SystemVue
2016.08

# Tutorials

KEYSIGHT
TECHNOLOGIES

## Notices

## Restricted Rights Legend

If software is for use in the performance of a U.S. Government prime contract or subcontract, Software is delivered and licensed as "Commercial computer software" as defined in DFAR 252.227-7014 (June 1995), or as a "commercial item" as defined in FAR 2.101(a) or as "Restricted computer software" as defined in FAR 52.227-19 (June 1987) or any equivalent agency regulation or contract clause.

Use, duplication or disclosure of Software is subject to Keysight Technologies' standard commercial license terms, and non-DOD Departments and Agencies of the U.S. Government will receive no greater than Restricted Rights as defined in FAR 52.227-19(c)(1-2) (June 1987). U.S. Government users will receive no greater than Limited Rights as defined in FAR 52.227-14 (June 1987) or DFAR 252.227-7015 (b)(2) (November 1995), as applicable in any technical data.

**Portions of this software are licensed by third parties including open source terms and conditions. For detail information on third party licenses, see Notice.**

# Contents

# Tutorials

# Simulation Control and Scripting

## Simulation Control and Scripting

In this tutorial section, you will learn how to control simulation from within the SystemVue application and from external programs such as LabVIEW and MATLAB.

- Controlling SystemVue from External Programs
- Optimizing a Simulation
- Using MATLAB Script For Sequence Control
- Performing a Monte Carlo Analysis on a Design
- Running a Yield Analysis on a Design
- Sweeping a Simulation

## Controlling SystemVue from External Programs

### Controlling SystemVue from External Programs

You can automate SystemVue use using the COM interface together with the VB scripting capabilities that are built into the product. The documentation for scripting is available at User's Guide > Using SystemVue > Scripts.

- Exploring the Workspace Using Visual Basic
- Running Scripts using Visual Basic
- Running a BER Analysis Controlled From LabVIEW, MATLAB, or C#

### Exploring the Workspace Using Visual Basic

### Exploring the Workspace Using Visual Basic

#### VBBrowser

The VBBrowser communicates to SystemVuethrough the COM interface. The source code for the VBBrowser is located in "Examples\Tutorials\Simulation_Control_and_Scripting\Scripting\Visual Basic\Browser\MainForm.vb"

The executable is available in "Examples\Tutorials\Simulation_Control_and_Scripting\Scripting\Visual Basic\Browser\VBBrowser.exe"

The files VBBrowser.exe and Interop.GENESYS.dll were created following the instructions found in ReadMe.txt located in the same directory. This application will let you explore an opened workspace, with it, you can find the path to the items in your workspace to use in your automation scripts.

## SystemVue Browser

The VBBrowser is used to browse objects in SystemVue. This is an interactive program that allows a user to see what functions are available to call within the script processor. The program communicates with one active instance of the SystemVue program. The browser looks at the current workspace and retrieves objects and items from it.

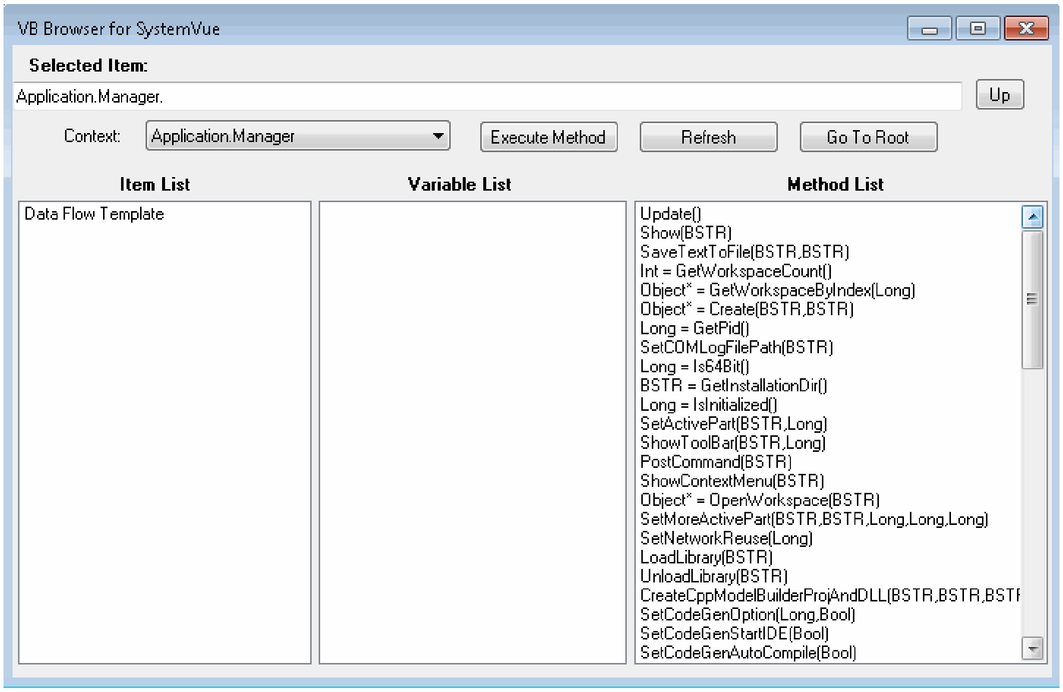
## Running the VBBrowser

There are two ways to launch the VBBrowser

1. Run the VBBrowser while you have an instance of SystemVue running. If multiple instances of SystemVue are open, the VBBrowser will attach to the first instance.
2. Launch the VBBrower without SystemVue running. The VBBrowser will launch as well as SystemVue.

**NOTE** If you load another workspace in SystemVue while the VBBrowser is running it is best to click the Go To Root button to avoid errors. Clicking the Refresh or Up button will throw an error and then load the root.

## Contents of the VBBrowser



Selected Item: Application.Manager.

Context: Application.Manager

Item List:

Data Flow Template

Method List:

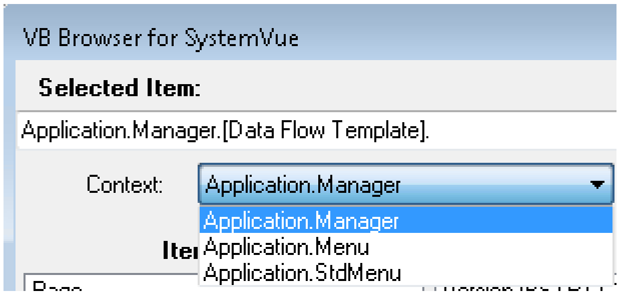
```
Update()
Show(BSTR)
SaveTextToFile(BSTR,BSTR)
Int = GetWorkspaceCount()
Object* = GetWorkspaceByIndex(Long)
Object* = Create(BSTR,BSTR)
Long = GetPid()
SetCOMLogFilePath(BSTR)
Long = Is64Bit()
BSTR = GetInstallationDir()
Long = IsInitialized()
SetActivePart(BSTR,Long)
ShowToolBar(BSTR,Long)
PostCommand(BSTR)
ShowContextMenu(BSTR)
Object* = OpenWorkspace(BSTR)
SetMoreActivePart(BSTR,BSTR,Long,Long,Long)
SetNetworkReuse(Long)
LoadLibrary(BSTR)
UnloadLibrary(BSTR)
CreateCppModelBuilderProjAndDLL(BSTR,BSTR,BSTI
SetCodeGenOption(Long,Bool)
SetCodeGenStartIDE(Bool)
SetCodeGenAutoCompile(Bool)
```

**CAUTION** The workspace name reported at the root level of VBBrowser can be different than the actual file name if the user has changed the file name manually. When creating syntax using the workspace name, you may need to use the name of the workspace file on the hard disk and not the name reported by the VBBrowser.

### General

The Selected Item box contains the syntax for the script that you can execute by clicking the Execute Method button.

The Context drop box contains three items

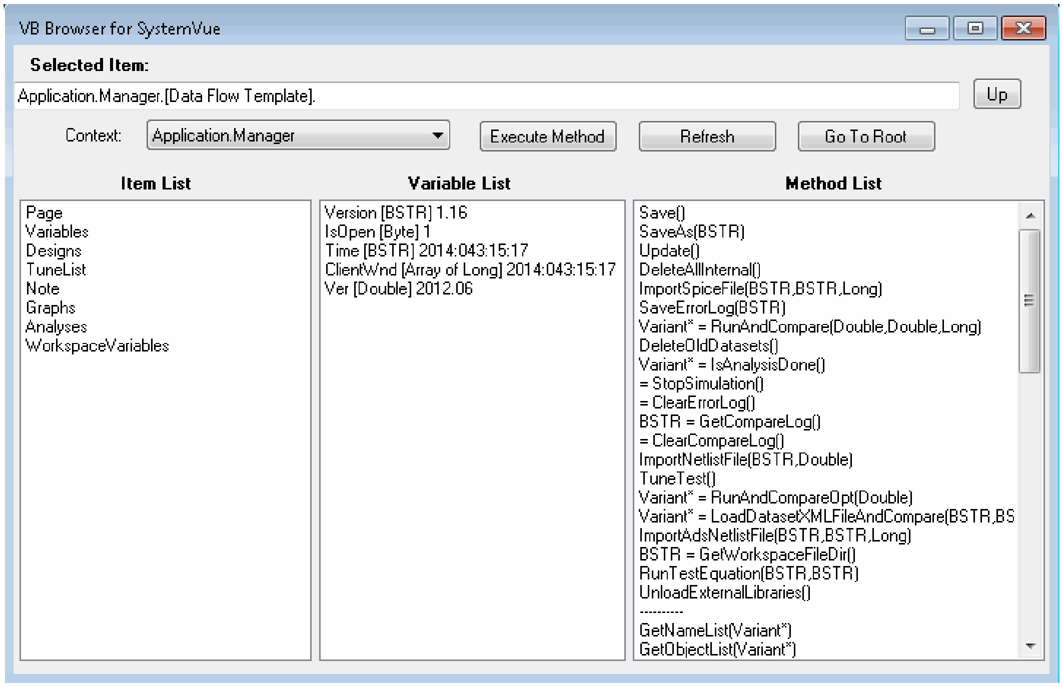


Selected Item: Application.Manager.[Data Flow Template].

Context: Application.Manager

1. Application.Manager (default) Sets the Item List to the context of the workspace tree.
2. Application.Menu Sets the Item List to the context of the current Menu Bar in SystemVue
3. Application.StdMenu - Sets the Item List to the context of the standard Menu Bar in SystemVue

### Lists



Selected Item: Application.Manager.[Data Flow Template].

Context: Application.Manager

Item List:

```
Page
Variables
Designs
TuneList
Note
Graphs
Analyses
WorkspaceVariables
```

Variable List:

```
Version [BSTR] 1.16
IsOpen [Byte] 1
Time [BSTR] 2014:043:15:17
ClientWnd [Array of Long] 2014:043:15:17
Ver [Double] 2012.06
```

Method List:

```
Save()
SaveAs(BSTR)
Update()
DeleteAllInternal()
ImportSpiceFile(BSTR,BSTR,Long)
SaveErrorLog(BSTR)
Variant* = RunAndCompare(Double,Double,Long)
DeleteOldDatasets()
Variant* = IsAnalysisDone()
= StopSimulation()
= ClearErrorLog()
BSTR = GetCompareLog()
= ClearCompareLog()
ImportNetlistFile(BSTR,Double)
TuneTest()
Variant* = RunAndCompareOpt(Double)
Variant* = LoadDatasetXMLFileAndCompare(BSTR,BS
ImportAdsNetlistFile(BSTR,BSTR,Long)
BSTR = GetWorkspaceFileDir()
RunTestEquation(BSTR,BSTR)
UnloadExternalLibraries()
----------
GetNameList(Variant*)
GetObjectList(Variant*)
```

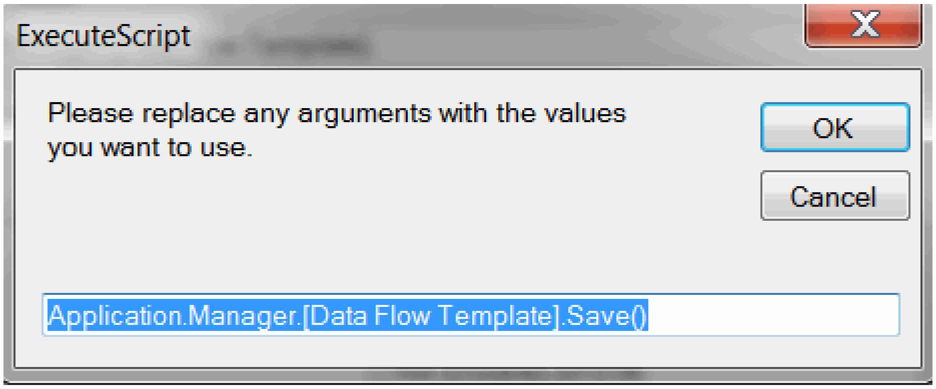
*Item List* - The window contains a list of all the items found in the current context. If nothing appears in the window you can click the Refresh button to refresh the context. Clicking on an item in this list will show you a list of sub items. Note that the sub items correspond to the items inside the opened workspace. Notice that as

you click items, the text in the selected item box changes. The first thing you should see (in the default context) in the Item list is the name of the workspace(s) that are loaded in SystemVue. In the example above you would see
Data Flow Template

as the first item on the list.

*Variable List* - The window contains a list of properties, variables, or parameters that are associated with the current item. Items in this list can be called as a property to an item.

*Method List* - The window contains a list of the methods that can be used with the current item. Notice that by double clicking on a method the ExecuteScript window pops up with current syntax of the method you have selected. This syntax is generated from the Selected Item text box and the method you have clicked. This is what would pop up if you double clicked the Save() method.



ExecuteScript

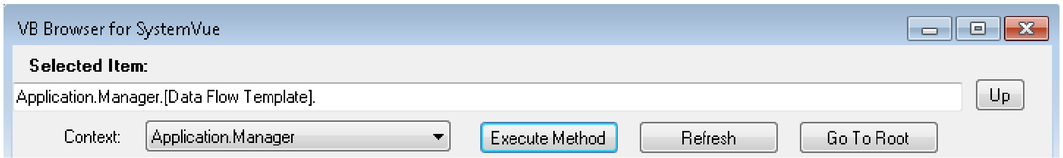
Please replace any arguments with the values you want to use.

Application.Manager.[Data Flow Template].Save()

NOTE

You can execute this one line script by clicking on OK. A script processor window will not pop up in SystemVue, so you may not always know if it worked or not. If you need to execute many lines it is suggested to use a script. The ExecuteScript window is best used as a guide to get the correct syntax for writing your own script.

## Buttons



VB Browser for SystemVue

Selected Item: Application.Manager.[Data Flow Template].

Context: Application.Manager

*Up* - The button sets the Item List to the parent item of the current Item List window

*Execute Method* The button will bring up the ExecuteScript window that shows the syntax for the current Selected Item and gives the option to run it or not.

*Refresh* - The button reloads the items in the three lists.

*Go To Root* - The button sets the Item List to the top most parent.

### Example 1: Set a variable using the VBBrowser

In this example, you will control a SystemVue variable from the VBBrowser. We will be using a similar technique in the C#, LabVIEW and MATLAB examples in the following tutorials. To learn more about this example, see Introduction: SystemVue Eb/N0 Sweep for BER.

1. Close all, but keep one SystemVue instance running on your PC.
2. Open the example "Comms\BER\QPSK_BER_Coded_Viterbi.wsv"
3. Open the Equation1 equations.
4. Start VBBrowser.exe, located in your Examples directory: "Tutorials\Simulation_Control_and_Scripting\Scripting\Visual Basic\Browser\VBBrowser.exe", position this window so that you can see it and the Equation1 equations in SystemVue.
5. In the Item List, you should see "QPSK_BER_Coded_Viterbi", double click on it and traverse to the EbN0 value (the first item you see in the Item List is the WorkspaceVariables; once you double click on it, you will see its item list; double click on VarBlock and then you will see the variable EbN0). The VBBrowser should report that your selected item is "Application.Manager. QPSK_BER_Coded_Viterbi.WorkspaceVariables.VarBlock.\[EbN0\].".
6. In the Variable List, click on Data. In the Method List, double click on Set (BSTR).
7. The VBBrowser will open an ExecuteScript dialog. In this dialog, change BSTR to 8. As you are clicking the OK button in this dialog, look at the NDensity Variable in Equation1. You will see it change as the new EbN0 is applied.

### Example 2: Run a simulation using the VBBrowser

In this example, you will run a simulation using the VBBrowser using the variable you set above. It is assumed that you have SystemVue and VBBrowser still open.

1. In the VBBrowser, click on the Go To Root button. This will reset the path to the top.
2. In the Item List, traverse to the Uncoded_QPSK_BER_Analysis analysis. The VBBrowser should report that your selected item is "Application.Manager. QPSK_BER_Coded_Viterbi.Analyses.Uncoded_QPSK_BER_Analysis.".
3. In the Method List, double click on RunAnalysis().
4. The VBBrowser will open an ExecuteScript dialog. In this dialog, click the OK button.

### Running Scripts Using Visual Basic

## Running Scripts Using Visual Basic

### Running a Script from Microsoft Excel

Microsoft Excel has a Visual Basic for an Applications (VBA) engine that you can use to script other applications that support a COM interface. In the case of SystemVue, this means that a script can be written in Microsoft Excel that opens SystemVue, does something such as load a workspace and run simulations, collects data, and processes the data. For information on accessing the VBScript development editor in Microsoft Excel, see your version of Excel's Help. For Microsoft Excel 2010, see Getting Started with VBA in Excel 2010.

The global windows name for SystemVue's COM server is GENESYS. To use the COM server SystemVue, you must register it with the Windows operating system by name so that a script can access it (including run an instance of it). To register or unregister the COM server refer toRegister COM Interface

**NOTE** The bitness of the Excel executable must match the bitness of the COM server. For instance, you need to install 64-bit Excel to call 64-bit SystemVue.

The following code shows a simple script which simply opens an instance of SystemVue:

```
Sub myScript()
  Dim comServer As Object  ' Declare variable that
references our COM server
  Set comServer = CreateObject("Genesys.Application")
' Open an instance of the application
  comServer.Application.Visible = True  ' By default
the application is hidden, we make it visible using this
 command
End Sub
```

**NOTE** The example instructions below are for Microsoft Excel 2010.

### Example 1:

Create a VBA macro to run the Bluetooth example workspace In this example, you will create a more involved VBA script which opens SystemVue, opens the example workspace named "Comms\Bluetooth.wsv", runs a particular analysis that is located in the workspace, gets data from the dataset, and imports the data into an excel spreadsheet. # Open a new workbook in Microsoft Excel

Create a new Microsoft Excel VBA macro called `RunBluetoothExample`, refer to the Getting Started with VBA in Excel 2010 to assist in following the steps below.

1. Enabled the *Developer Ribbon* in Microsoft Excel.
2. Click on *Macros* button in the *Developer Ribbon*.

3. In the *Macro Dialog*, enter the new macro name `RunBluetoothExample` and hit the *Create* button.
4. Copy and paste the following code into the new macro:

```
    Dim comServer As Object  ' Declare variable that
references our COM server
    Set comServer = CreateObject("Genesys.Application")
' Open an instance of the application
    comServer.Application.Visible = True  ' By default
the application is hidden, we make it visible using this
 command
    Dim path As String
    comServer.Manager.FileOpenExample "Comms\Bluetooth.
wsv"
    path = comServer.Manager.GetExeDir()
    Set WsDoc = comServer.Manager.GetWorkspaceByIndex(0)
    WsDoc.SetEqnWorkingDir (path)
    WsDoc.Analyses.DF1.RunAnalysis
    S1 = WsDoc.Analyses.DF1_Data.Eqns.VarBlock.S1.
GetValue()
    Dim oXL As Excel.Application
    Dim oWB As Excel.Workbook
    Dim oSheet As Excel.Worksheet
    Dim oRng As Excel.Range
    Dim iNumQtrs As Integer
    Set oXL = Excel.Application  ' Activate Excel
    oXL.Visible = True
    ' Set active Workbook
    Set oWB = oXL.Workbooks.Application.ActiveWorkbook
    ' Set active Sheet
    Set oSheet = oWB.ActiveSheet
    ' output first 2500 datapoints to excel
    For i = 0 To 2499
        oSheet.Cells(i + 1, 1).Value = S1(i)
    Next i
    Exit Sub
Err_Handler:
      MsgBox Err.Description, vbCritical, "Error: " &
Err.Number
```

1. Add the reference to the COM interface, by selecting Tools > References.. menu pick, and select GENESYS.
   Note, if you don't see GENESYS - make sure you have registered the SystemVue COM interface.
2. Run the macro by clicking on the green play button.

### Example 2:

Extend the VBA macro to extract additional data and plot the results In this example, you will extend the macro above to extract S1, S2, S1_Time and S2_Time. Additionally, you will create a simple GUI to run the macro and plot the results.

1. Edit the `RunBluetoothExample` macro created in the prior example.
2. Add commands to define and import S1_Time, S2, and S2_Time.
3. Add commands to define column names:
   - Column A: S1_Time
   - Column B: S1
   - Column C: S2_Time
   - Column D: S2
4. Add commands to copy the imported S1, S1_Time, S2, and S2_Time data into the appropriate columns spreadsheet. Note, as we have added labels - the data must start in row 2.
5. Verify your changes by running the macro.
6. Insert a *Scatter* line chart into Sheet1.
7. Right click on the chart and choose the Select Data... command.
8. Right click the Add button in the Select Data Source dialog, define:
   - Series Name: S1
   - Series X Values: =Sheet1!$A2:$A2501
   - Series Y Values: =Sheet1!$B2:$B2501
9. Right click the Add button in the Select Data Source dialog, define:
   - Series Name: S2
   - Series X Values: =Sheet1!$C2:$C2501
   - Series Y Values: =Sheet1!$D2:$D2501
10. Add a button to Sheet1, by clicking using Insert menu in *Developer* ribbon. Have this button execute the `RunBlueToothMacro`.
11. Right click on the button and Edit Text. Relabel button to RunBlueToothMacro.
12. Optional: Use Record Macro in the *Developer* ribbon to create a macro to clear columns A-D. Add a button to run this macro and label it ClearBluetoothData.

### Running a Script from a Visual Basic Program

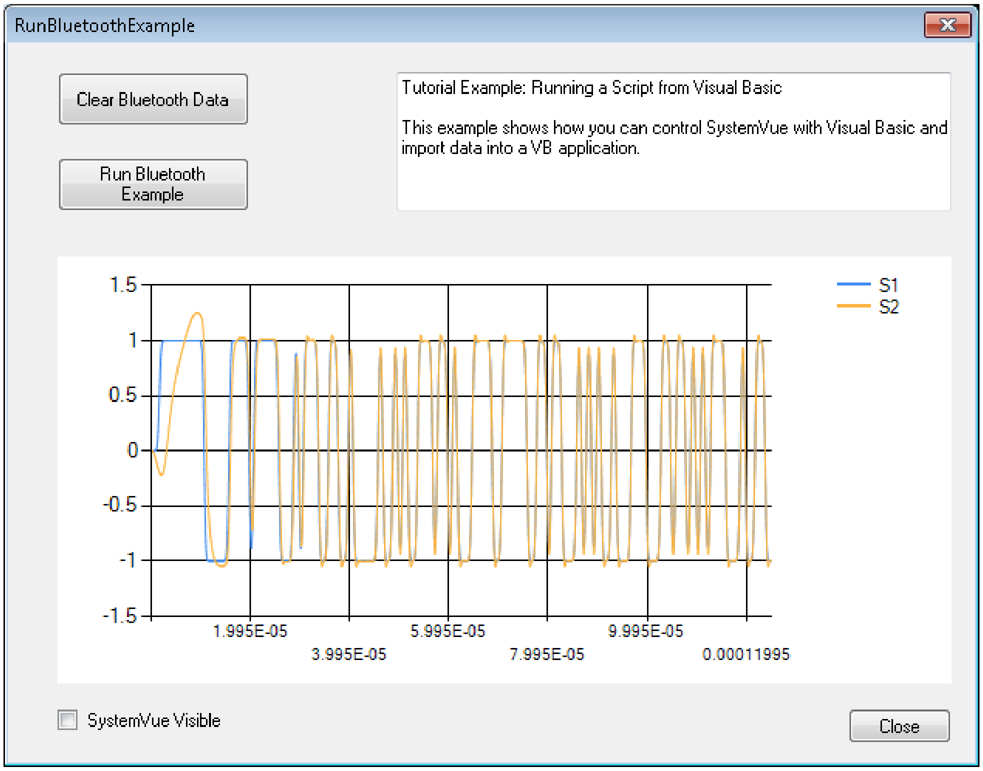
This example is similar to the previous two examples that use VBA in Microsoft Excel. For this example, we use a standalone executable written in Visual Basic to open the Comms/Bluetooth.wsv workspace, run the analysis, and extract and display data. The solution to this example is in: Examples/Tutorials

/Simulation_Control_and_Scripting/Scripting/Visual Basic/Visual Studio /RunBluetoothExample.sln. To customize it, you can use Visual Studio Visual Basic Express Edition (free from Microsoft).

### Example 3:

Run the Visual Basic program

1. Run the RunBluetoothExample.exe located at: Examples/Tutorials /Simulation_Control_and_Scripting/Scripting/Visual Basic/Visual Studio /RunBluetoothExample.exe.
   The following window will be displayed, but without data.



Tutorial Example: Running a Script from Visual Basic

This example shows how you can control SystemVue with Visual Basic and import data into a VB application.

2. Hit the Run Bluetooth Example button. This will launch SystemVue in the background. If the SystemVue Visible option has been checked then SystemVue will become visible once it has loaded.
3. Once SystemVue is loaded, the visual basic program calculates the analysis DF1, extracts the results from the dataset, and plots the data in a chart as seen above.
4. Clear the data using the Clear Bluetooth Data button.
5. Close or exit the SystemVue_Visual_Basic.exe. Notice that the instance of SystemVue will be closed also.

## Running a BER Analysis Controlled From LabVIEW MATLAB or C Sharp

## Running a BER Analysis Controlled From LabVIEW, MATLAB, or C#

In this section, we will review the COM interface examples that ship with SystemVue. All except the last example in this section perform the following steps, native in each environment:

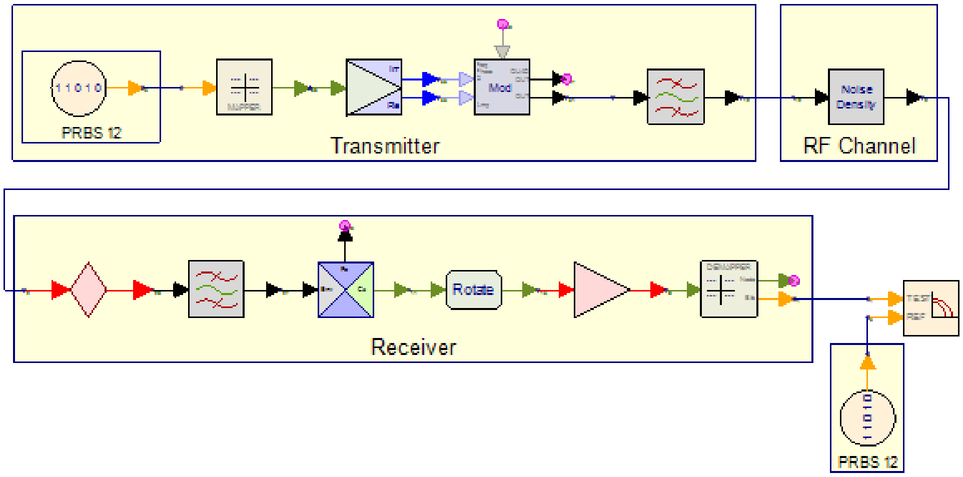
1. Launch SystemVue
2. Open a workspace
3. Sweep a variable
4. Run a simulation
5. Retrieve the result

To simplify the use of the COM interface of SystemVue, we have created an example NET DLL component, SystemVueNET.dll.

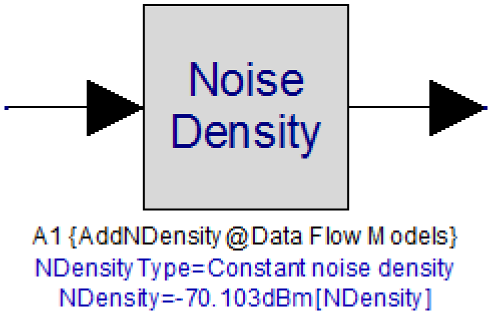
## Introduction: SystemVue Eb/N0 Sweep for BER

In this section, we review the workspace used in the first three COM interface examples. In each of these examples, we will be performing a bit error rate (BER) analysis by sweeping the Eb/N0 parameter. We can implement this sweep natively in SystemVue using a parameter sweep. The workspace example is located in "Examples\Comms\BER\QPSK_BER_Coded_Viterbi.wsv". In this workspace, we will be sweeping the Uncoded_QPSK_Design over multiple parameters Eb/N0 values.

Below is the schematic, note the four distinct sections, transmitter, channel, receiver, and BER measurement:



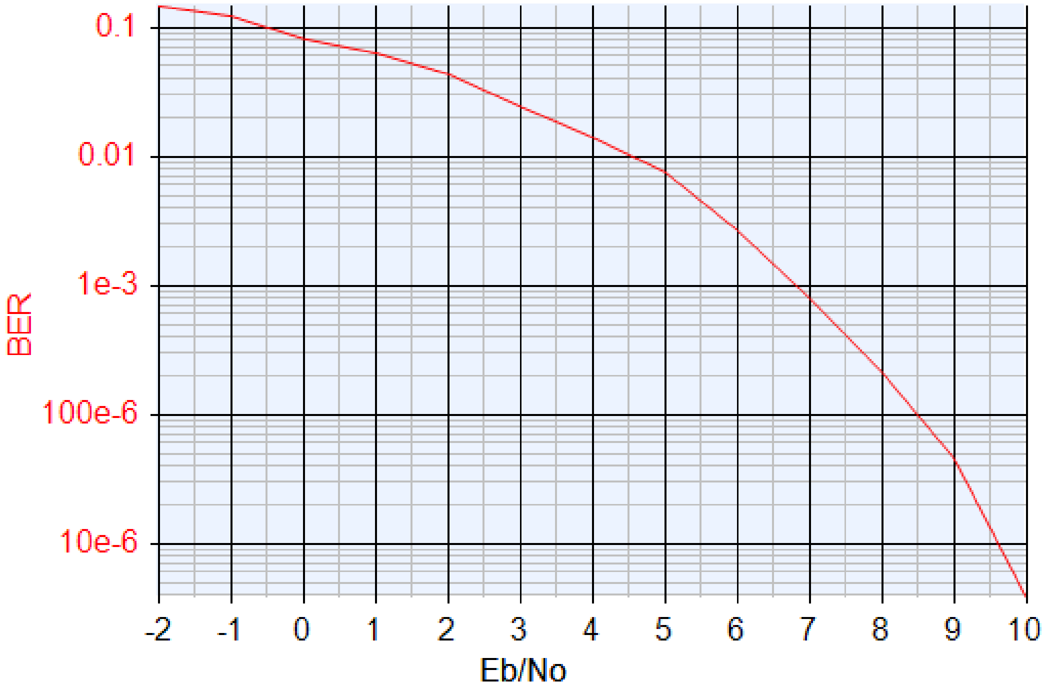
As we perform the BER analysis for an Eb/N0 value, we calculate and modify the value of noise density (NDensity) of the channel:



A1 {AddNDensity@Data Flow Models}
NDensityType=Constant noise density
NDensity=-70.103dBm[NDensity]

Below is the parameter sweep in SystemVue, we will be reimplementing this control for the COM interface examples. If you hit calculate now, you can zoom into the channel and see the NDensity as it is being updated for each sweep point.

Finally, after we calculate the Eb/N0 sweep in SystemVue, we can see the BER waterfall plot:



To accomplish this sweep, we first define an equation block declaring that Eb/N0 will be swept:

```
% Eb/No = energy per bit / noise density

% EbN0 is defined in a separate equation block to enable
% updating the variable using external control.  See the
% Automation section in the notes.

EbN0=tune(3)
```

In another equation block, we calculate the NDensity using the swept Eb/N0 value:

```
ModPower_dBm = 13 % modulator output power in dBm
SymbolRate = 51.2e+6
ModPower_W = 10^( (ModPower_dBm-30)/10 );
ModPower_Vrms = sqrt( 50*ModPower_W);
ModCarrier = 300e6;
ModAmpSensitivity = ModPower_Vrms*sqrt(2);
SymbolTime = 1/SymbolRate
BitsPerSymbol = 2
% Eb/No = energy per bit / noise density
Eb_dBm = ModPower_dBm - 10*log10( SymbolRate *
BitsPerSymbol )
No_dBm = Eb_dBm - EbN0
NDensity = No_dBm
```

Note, since we are using the COM interface, we must declare Eb/N0 in a separate equation block. By doing so, as we change Eb/N0 over COM, the second equation block will be automatically calculated before the simulation is run.

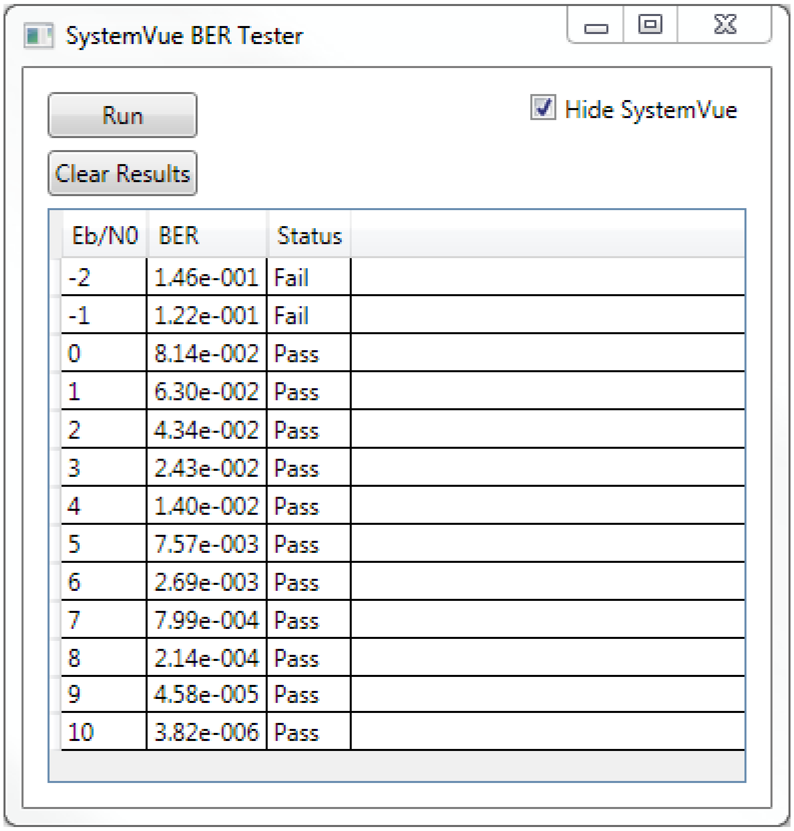
In this example, we swept Eb/N0 and displayed the BER results. In the following sections, we will use the SystemVue COM interface to implement the sweep in the following environments:

- Visual C#
    - Simplifying the COM Interface using NET DLL component
    - Performing the BER Analysis
- LabVIEW
- MATLAB

## Visual C#

In this example, we use Visual C# to perform the Eb/N0 sweep. The executable is provided at: "Examples\Tutorials\Simulation_Control_and_Scripting\Scripting\C#\QPSK_BER.exe"

When you start it, you will see:



☑ Hide SystemVue

| Eb/N0 | BER | Status |
|---|---|---|
| -2 | 1.46e-001 | Fail |
| -1 | 1.22e-001 | Fail |
| 0 | 8.14e-002 | Pass |
| 1 | 6.30e-002 | Pass |
| 2 | 4.34e-002 | Pass |
| 3 | 2.43e-002 | Pass |
| 4 | 1.40e-002 | Pass |
| 5 | 7.57e-003 | Pass |
| 6 | 2.69e-003 | Pass |
| 7 | 7.99e-004 | Pass |
| 8 | 2.14e-004 | Pass |
| 9 | 4.58e-005 | Pass |
| 10 | 3.82e-006 | Pass |

This custom application enables you to:

- Hit the Run button to perform the sweep
- Hide and unhide the visibility using the check box provided.

To see the sweep in action, unhide SystemVue, zoom into the channel, and watch the NDensity parameter update as each sweep point is evaluated.

The Visual Studio solution is supplied in the "Examples\Scripting\C#\Visual Studio" directory. To customize it, you can use Visual Studio 2010 C# Express Edition (free from Microsoft).

### Simplifying the COM Interface using NET DLL component

To help with all of the Eb/N0 examples, we supply an example NET DLL component, named SystemVueNET.dll. This DLL allows us to simplify the management of the COM interface for the QPSK BER examples implemented in C#, LabVIEW, and MATLAB.

In this DLL, we define a class called SystemVue, the file located in "Examples\Tutorials\Simulation_Control_and_Scripting\Scripting\C#\Visual Studio\SystemVueNET\SystemVue.cs":

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using Microsoft.Win32;

namespace SystemVueExample
{
    public class SystemVue
    {

        // Instance of SystemVue application
        GENESYS.Application m_app;

        // Constructor, called when a instance of this
class is created
        public SystemVue()
        {
            try
            {
                // Start a new instance of SystemVue
                m_app = new GENESYS.Application();
            }
            catch
            {
                // If we have a exception, the COM
server is probably not registered.
                // Register it by running SystemVue.exe
/regserver
                m_app = null;
            }

        }

        // Member boolean to track visibility
        bool m_bVisible = false;

        // Method to set/get Visible property of
SystemVue - by default SystemVue will start hidden
under COM control
        public bool Visible
        {
            get { return m_bVisible; }
            set
            {
                m_bVisible = value;
                if (m_app != null)
                {
                    m_app.Application.Visible =
m_bVisible;
                }
            }
        }

        // Some external environments need a separate
method to set visibility
        public void SetVisible(bool bVisible)
        {
            Visible = bVisible;
        }

        // Destructor
        ~SystemVue()
        {
            // Close and save all workspaces
            try
            {
                for (int i = 0; i < m_app.Manager.
GetWorkspaceCount(); i++)
                {
                    GENESYS.Workspace workspace = m_app.
Manager.GetWorkspaceByIndex(i);
                    // COM interface does not support
quitting without saving, so save to temp file, and then
delete it
                    string file = Path.
GetTempFileName();
                    workspace.SaveAs(file);
                    File.Delete(file);
                }
            }
            catch
            {
            }

            // Quit the application
            if (m_app != null)
                m_app.Quit();
        }

        // Run a VB script command
        public bool RunScript(string csScript)
        {
            bool bStatus = true;

            try
            {
                // Run a script, assuming Visual Basic
                m_app.Application.RunScript(csScript,
GENESYS.ScriptLanguage.genLangVBScript);
            }
            catch
            {
                bStatus = false;
            }

            return bStatus;
        }

        // Open a workspace, given the path
        public bool OpenWorkspace(string sPath)
        {
            string sCommand;

            sCommand = "OpenWorkspace(\"";
            sCommand += sPath.Replace('/', '\\'); //
SystemVue 2011.10 and earlier must have backslashes
            sCommand += "\")";

            return RunScript(sCommand);

        }

        public bool OpenExampleWorkspace(string sPath)
        {
            string sCommand;

            sCommand = "FileOpenExample(\"";
            sCommand += sPath.Replace('/', '\\'); //
SystemVue 2011.10 and earlier must have backslashes
            sCommand += "\")";

            return RunScript(sCommand);

        }

        private void SetAutoCalcOff(string sParamPath)
        {
            // See if this is a varblock and turn off
automation
            try
            {
                int iIndex = sParamPath.IndexOf(".
VarBlock"); // if not found, exception thrown and
caught below
                string sVarBlockPath = sParamPath.Remove
(iIndex);
                GENESYS.IItem equationBlock = GetItem
(sVarBlockPath);
                RunScript("autocalc=false\r\n" +
sVarBlockPath + ".SetProperty \"AutoCalc\", autocalc");
            }
            catch
            {
                // If not found, igore as it is not
equation block
            }
        }

        // Set a scalar double parameter
        public bool SetStringParameter(string
sParamPath, string sParamValue)
        {
            bool bSuccess = true;

            SetAutoCalcOff(sParamPath);

            bSuccess = RunScript(sParamPath + ".Set(
\"'" + sParamValue + "\" )");

            return bSuccess;
        }

        // Set a scalar double parameter
        public bool SetParameter(string sParamPath, doub
le sParamValue)
        {
            bool bSuccess = true;

            SetAutoCalcOff(sParamPath);

            bSuccess = RunScript(sParamPath + ".Set( "
+ sParamValue + " )");

            return bSuccess;
        }

        // Get data from dataset, assuming double
        public double GetData(string sDataName)
        {
            GENESYS.IItem item = GetItem(sDataName);

            double data = 0.0;

            if (item != null)
            {
                for (int i = 0; i < item.GetVarCount();
i++)
                {
                    string itemName = item.GetVarName
(i);
                    if (itemName == "Data")
                    {
                        data = (double)(((GENESYS.IItem)
item).GetVarValue(i));
                        break;
                    }
                }
            }

            return data;
        }

        // Find a item in a Genesys item
        public GENESYS.IItem GetItem(string sItemName)
        {
            GENESYS.IItem me = null;
            if (m_app != null)
            {
                me = (GENESYS.IItem)m_app.Manager;
                me = GetItem(me, sItemName);
            }
            return me;
        }

        // Find a item, given a path
        static GENESYS.IItem GetItem(GENESYS.IItem
parent, string sItemName)
        {
            GENESYS.IItem item = parent;

            string[] path = sItemName.Split('.');
            try
            {
                foreach (string itemName in path)
                {
                    if (item != null)
                        item = item.GetItemByName
(itemName);
                }
            }
            catch
            {
                item = null;
            }
            return item;
        }
    }
}
```

### Performing the BER Analysis

In the Visual Studio solution, the QPSK_BER project defines the GUI and control for the BER sweep. Most of the implementation of this application are in the "Examples\Tutorials\Simulation_Control_and_Scripting\Scripting\C#\Visual Studio\QPSK_BER\QPSK_BER.cs" file. The RunAnalysis method (shown below) performs the sweep. We use SystemVueNET.dll created in the previous section to interface to the SystemVue COM interface.

```
public void RunAnalysis()
{
    // Create a new instance only if needed
    if (systemVue == null)
    {
        // Start a new instance of SystemVue
        systemVue = new SystemVueExample.SystemVue();
        // Open the workspace
        systemVue.OpenExampleWorkspace("Comms/BER
/QPSK_BER_Coded_Viterbi.wsv");

        systemVue.Visible = Visible;

    }

    // Sweep Eb/N0 -2 to 10 and calculate the BER
    for (int EbN0 = -2; EbN0 <= 10; EbN0++)
    {
        // Set the NDenstity parameter
        systemVue.SetParameter("QPSK_BER_Coded_Viterbi.
WorkspaceVariables.VarBlock.[EbN0]", EbN0);

        // Run the analysis
        systemVue.RunScript(
            "QPSK_BER_Coded_Viterbi.Analyses.
Uncoded_QPSK_BER_Analysis.RunAnalysis");

        // Read BER from dataset
        double BER = systemVue.GetData(
            "QPSK_BER_Coded_Viterbi.Analyses.
Uncoded_QPSK_BER_Data.Eqns.VarBlock.B11_BER");

        QPSK_BER.SimulationResult newSim = new QPSK_BER.
SimulationResult();
        newSim.BER = BER;
        newSim.EbN0 = EbN0;
        newSim.Test = BER > .1 ? "Fail" : "Pass";
        m_SimulationResults.Add(newSim);
   }
}
```

### Example 1: Run the C# QPSK_BER program

In this example, you will run the QPSK_BER program that was installed with SystemVue. By running this, you will be able to observe how SystemVue can be controlled by an external program.

1. Under your SystemVue install directory, open the directory: Examples\Tutorials\Simulation_Control_and_Scripting\Scripting\C#
2. Double click on QPSK_BER.exe
3. Click on the *Run* button, it will take a few seconds for SystemVue to load. As SystemVue is running under COM control, the SystemVue GUI will be hidden.
4. After the run is complete, uncheck the checkbox *Hide SystemVue*.
5. Open the *Designs/Uncoded_QPSK_Design* design and zoom into the *Noise Density* part so you can easily see the *NDensity* parameter. The C# program modifies the Eb/N0 value which is used to calculate the *NDensity* variable.

6. Arrange the SystemVue and the *SystemVue BER Tester* windows so you can see both on your screen.
7. Click the *Clear Results* button in the *SystemVue BER Tester*.
8. Click the *Run* and observe the *NDensity* parameter update as the sweep points are calculated.
9. Open the *EbN0* and *Equation1* equations in the workspace, arrange the windows so you can see both. Click *Run* again in the *SystemVue BER Tester*. To enable this control, the *EbN0* value is required to be set in a different set of equations from where it is used. The *Auto Calculate* property of *Equation1* equations forces the reevaluate of *NDensity* as the *EbN0* value is changed by the external program.

### Example 2: Create a Custom C# Console Application

In this example, you will build a custom C# program that will run the BER example on the command line (also referred to as a *Console Application*). To customize this, you will need to have Visual Studio C# Express Edition installed.

1. Copy the Examples\Tutorials\Simulation_Control_and_Scripting\Scripting\C#\Visual Studio
2. In the copied directory, double click on *SystemVueNET.sln*. In the Solution Explorer tab, you will see two projects:
   a. *QPSK_BER*: Defines BER sweep and BER tester GUI.
   b. *SystemVue_NET*: Defines a simple utility DLL that controls the SystemVue COM automation layer. You will be using the DLL in the standalone executable.
3. To build and run the example *BER Tester*GUI:
   a. Right click on *QPSK_BER* project in the *Solution Explorer* and select *Debug > Start New Instance*.
   b. Explore the *QPSK_BER/QPSK_BER.cs* and the *SystemVueNET /SystemVue.cs* files to see how they work by setting breakpoints in the `RunAnalysis()` method in *QPSK_BER.cs*.
   c. After you are finished exploring, exit the BER tester GUI.
4. Now you will create a new standalone executable that can run the BER test as a command line executable.
5. In the *Solution Explorer*, right-click on *Solution 'SystemVueNET'* in the *Solution Explorer* and select *Add > New Project...*.
6. In the *Add New Project* dialog:
   a. Select the *Visual C# > Windows > Console Application* project.
   b. Set the name of your project to *My_QPSK_BER*.
   c. Click the *OK* button.

7. In the *Solution Explorer*, right-click on the new project *My_QPSK_BER* and select *Add Reference...*.

8. In the *Add Reference* dialog, select the *Projects* tab and select the *SystemVueNET* project. This will allow you to call into the methods in the methods of the *SystemVueNET* library.

9. Edit the *Program.cs* file in the newly created *My_QPSK_BER* project.

10. In the `Main`method, add the following code:

```
// Start a new instance of SystemVue
SystemVueExample.SystemVue systemVue = new
SystemVueExample.SystemVue();

// SystemVue by default will come up hidden, unhide
SystemVue
systemVue.Visible = true;


// Wait for user to hit a key to exit the program
Console.WriteLine("Press any key to exit...");
Console.ReadKey();
```

11. In the *Solution Explorer*, right-click on the new project *My_QPSK_BER* and select *Set as StartUp Project*. This will set this project to be the default project built and debugged by Visual Studio.

12. Hit the *F5* function key to compile and run *My_QPSK_BER*. (Alternatively, you can right-click on the *My_QPSK_BER* project and select *Debug > Start New Instance*.) SystemVue will be started and unhidden by the C# code. The application will then wait until you press a key to exit.

13. Add the following code to the `Main` method, after the line that unhides SystemVue.

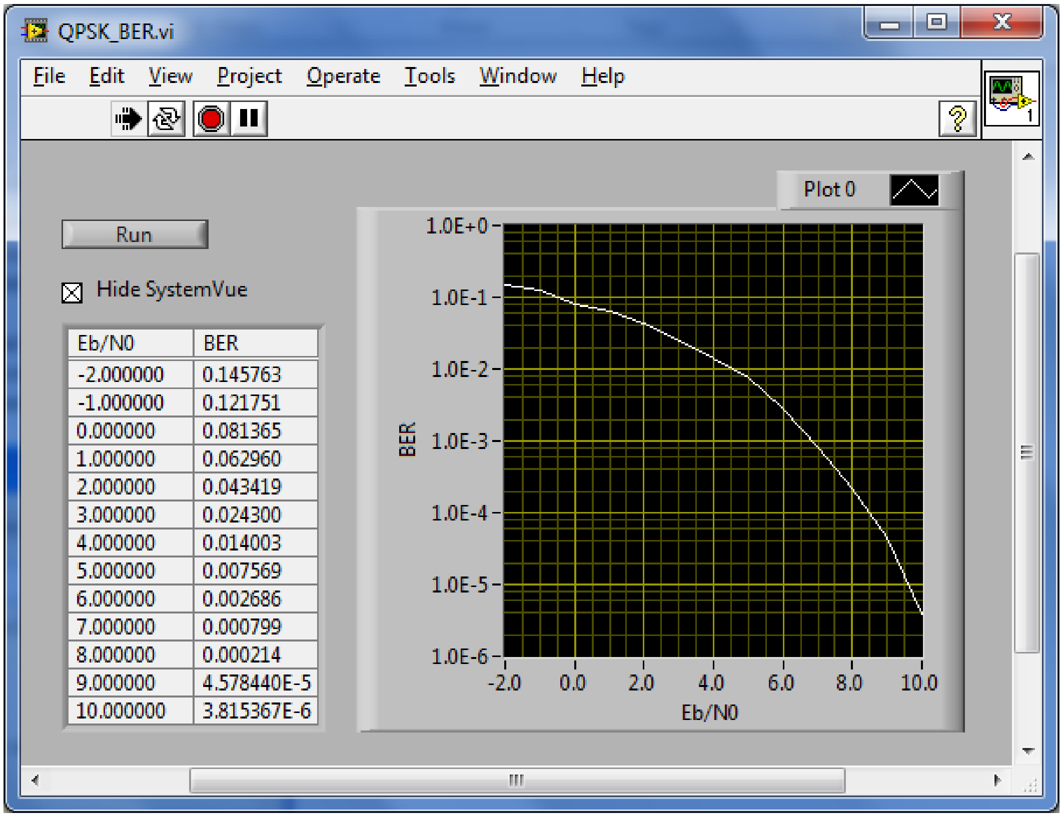
```
// Open the workspace
systemVue.OpenExampleWorkspace("Comms/BER
/QPSK_BER_Coded_Viterbi.wsv");

// Sweep Eb/N0 -2 to 10 and calculate the BER
for (int EbN0 = -2; EbN0 <= 10; EbN0++)
{
    // Set the NDenstity parameter
    systemVue.SetParameter("QPSK_BER_Coded_Viterbi.
WorkspaceVariables.VarBlock.[EbN0]", EbN0);

    // Run a simple VB command in SystemVue to run
the appropriate analysis
    systemVue.RunScript("QPSK_BER_Coded_Viterbi.
Analyses.Uncoded_QPSK_BER_Analysis.RunAnalysis()");
```

```
    // Read BER from dataset
    double BER = systemVue.GetData("QPSK_BER_Coded_V
iterbi.Analyses.Uncoded_QPSK_BER_Data.Eqns.VarBlock.
B11_BER");

    string result = "Eb/N0 = " + EbN0 + "\tBER = "
+ BER;
    Console.WriteLine(result);
}
```

14. Hit the *F5* function key to compile and run *My_QPSK_BER*.

## LabVIEW

In this example, we use LabVIEW to implement the BER analysis.



| Eb/N0 | BER |
|---|---|
| -2.000000 | 0.145763 |
| -1.000000 | 0.121751 |
| 0.000000 | 0.081365 |
| 1.000000 | 0.062960 |
| 2.000000 | 0.043419 |
| 3.000000 | 0.024300 |
| 4.000000 | 0.014003 |
| 5.000000 | 0.007569 |
| 6.000000 | 0.002686 |
| 7.000000 | 0.000799 |
| 8.000000 | 0.000214 |
| 9.000000 | 4.578440E-5 |
| 10.000000 | 3.815367E-6 |

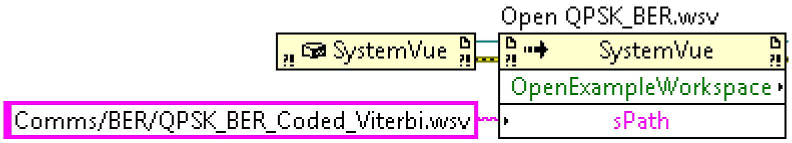
To run it, you will need to install LabVIEW Run-Time Engine available free from National Instruments.

As in the previous example, we use SystemVueNET.dll to manage the SystemVue COM interface.

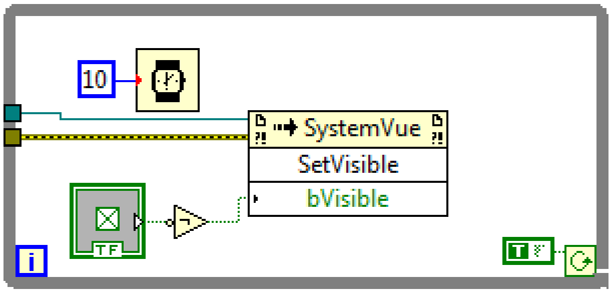
The LabVIEW vi is defined in the "Examples\Tutorials\Simulation_Control_and_Scripting\Scripting\LabVIEW\QPSK_BE vi" file. You will need LabVIEW to open the vi file.

The LabVIEW application provides the implementation for:

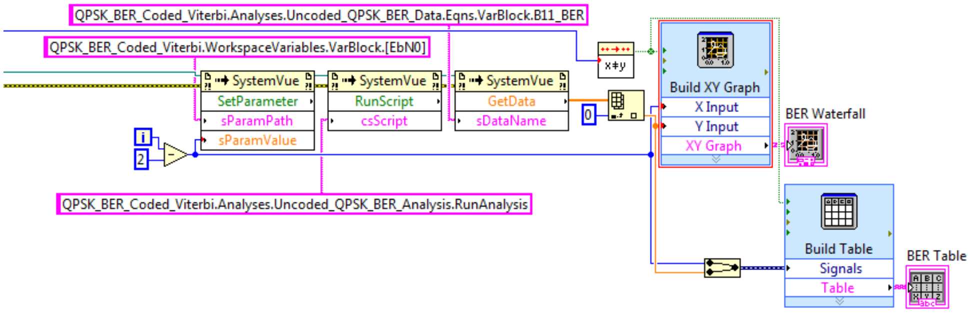
- Starting SystemVue and loading the workspace:



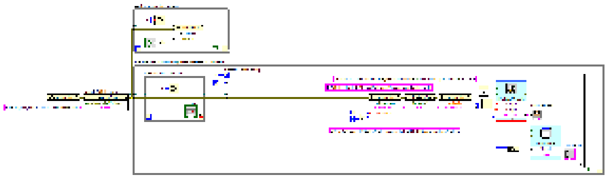
- Toggling the visibility of SystemVue:



Sweeping over the Eb/N0 and displaying the resultant BER:


QPSK_BER_Coded_Viterbi.Analyses.Uncoded_QPSK_BER_Data.Eqns.VarBlock.B11_BER
QPSK_BER_Coded_Viterbi.WorkspaceVariables.VarBlock.[EbN0]
SystemVue
SetParameter
sParamPath
sParamValue
SystemVue
RunScript
csScript
SystemVue
GetData
sDataName
x+y
0
i
2
Build XY Graph
X Input
Y Input
XY Graph
BER Waterfall
QPSK_BER_Coded_Viterbi.Analyses.Uncoded_QPSK_BER_Analysis.RunAnalysis
Build Table
Signals
Table
BER Table

To see the full LabView implementation, click on the image below:



Note, LabVIEW 2011 does not automatically recognize .NET DLLs compiled with VS 2010. To use .NET DLLs compiled with VS 2010, a .config file must be supplied. For more information see "Loading .NET 4.0 Assemblies in LabVIEW".

### Example 3: Run the compiled LabVIEW Run-Time QPSK_BER program

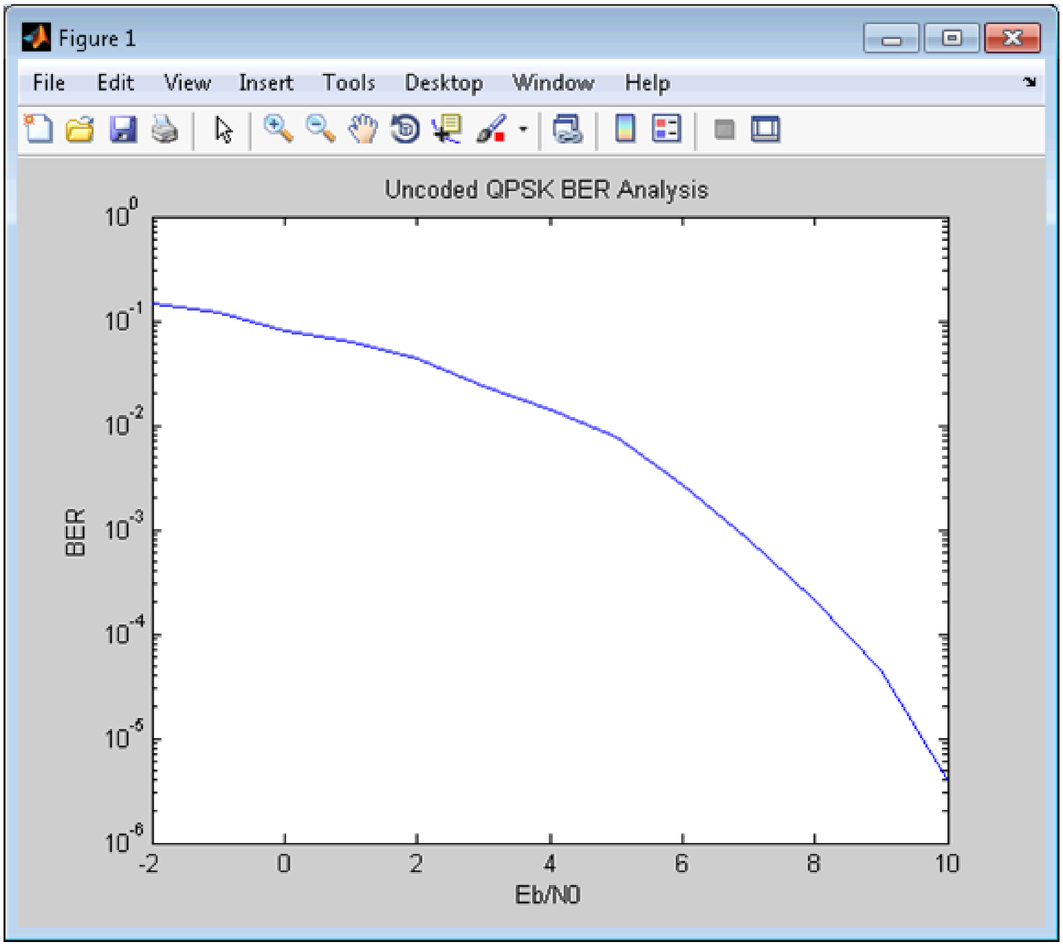
1. Double click on the LabVIEW vi file located in your SystemVue install directory at: "Examples\Tutorials\Simulation_Control_and_Scripting\Scripting\LabVIEW\QF vi."
2. Open the Operate tab and choose the Change to Run Mode.
3. Open the Operate tab again and choose Run.

4. Optionally, rerun the example with SystemVue unhidden by running the steps in the prior C# example.

5. In the "
"Examples\Tutorials\Simulation_Control_and_Scripting\Scripting\LabVIEW\" directory, double clock on the *QPSK_BER_vi.png* file. This image shows you the implementation of the LabVIEW control of SystemVue, compare the LabVIEW implementation with the C# RunAnalysis code. You can also view the LabVIEW implementation by open the Window tab and click Show Block Diagram.

### MATLAB

In this example, we use MATLAB to implement the BER analysis. The MATLAB script is defined in the "Examples\Tutorials\Simulation_Control_and_Scripting\Scripting\MATLAB\QPSK_BE m" file. When you run the script, you will see:



As in the previous example, we use SystemVueNET.dll created above to interface to the SystemVue COM interface.

```
% Find the directory path where this file is located
pathToDLL = fileparts( mfilename('fullpath'));
```

```
% Load the assembly in this directory (source code in
C# example area)
%NET.addAssembly([pathToDLL '/SystemVueNET.dll']);

% Open SystemVue and the workspace that we are
interested in
if exist( 'systemVue') == false
    % Start a new instance of SystemVue
    systemVue = SystemVueExample.SystemVue();

    % Hide SystemVue
    systemVue.Visible = false;

    % Open the workspace
    systemVue.OpenExampleWorkspace('Comms/BER
/QPSK_BER_Coded_Viterbi.wsv');
end

% Index into results matrix
i = 1;

% Sweep Eb/N0 -2 to 10 and calculate the BER
for j =-2:10,

    % Set EbN0
    EbN0(i) = j;
    systemVue.SetParameter('QPSK_BER_Coded_Viterbi.
WorkspaceVariables.VarBlock.[EbN0]', EbN0(i));

    % Run the analysis
    systemVue.RunScript('QPSK_BER_Coded_Viterbi.
Analyses.Uncoded_QPSK_BER_Analysis.RunAnalysis');

    % Read BER from dataset
    data = systemVue.GetData('QPSK_BER_Coded_Viterbi.
Analyses.Uncoded_QPSK_BER_Data.Eqns.VarBlock.B11_BER');
    BER(i) = data;

    % Display NDensity and BER on the console window
    disp(['Eb/N0 = ' num2str(EbN0(i)), ' BER = ',
num2str(BER(i))]);

    % Increment index into results matrix
    i = i+1;

end

%plot the results
semilogy(EbN0,BER)
xlabel('Eb/N0')
ylabel('BER')
title('Uncoded QPSK BER Analysis')
```

### Example 4: Run the MATLAB QPSK_BER program

1. Start MATLAB.
2. Drag and drop the QPSK_BER.m file located in the SystemVue install directory at "Examples\Tutorials\Simulation_Control_and_Scripting\Scripting\MATLAB\" into the MATLAB *Command Window*. This will run the BER test.
3. Double click the QPSK_BER.m file to open it in an editor, compare the MATLAB implementation with the C# RunAnalysis code.
4. Optionally, rerun the example with SystemVue unhidden by running the steps in the prior C# example.

**CAUTION** This example requires MATLAB 2011a or higher to run because SystemVueNET.dll is built using .NET 4.0.

## Optimizing a Simulation

### Optimizing a Simulation

This section has a series of tutorials that demonstrate how to optimize designs created in Spectrasys or Data Flow.

### Optimizing Spectrasys Designs

1. Intermod Optimization

### Optimizing Data Flow Designs

1. Peak to Average Power Ratio Optimization
2. EVM Optimization

### Intermod Optimization

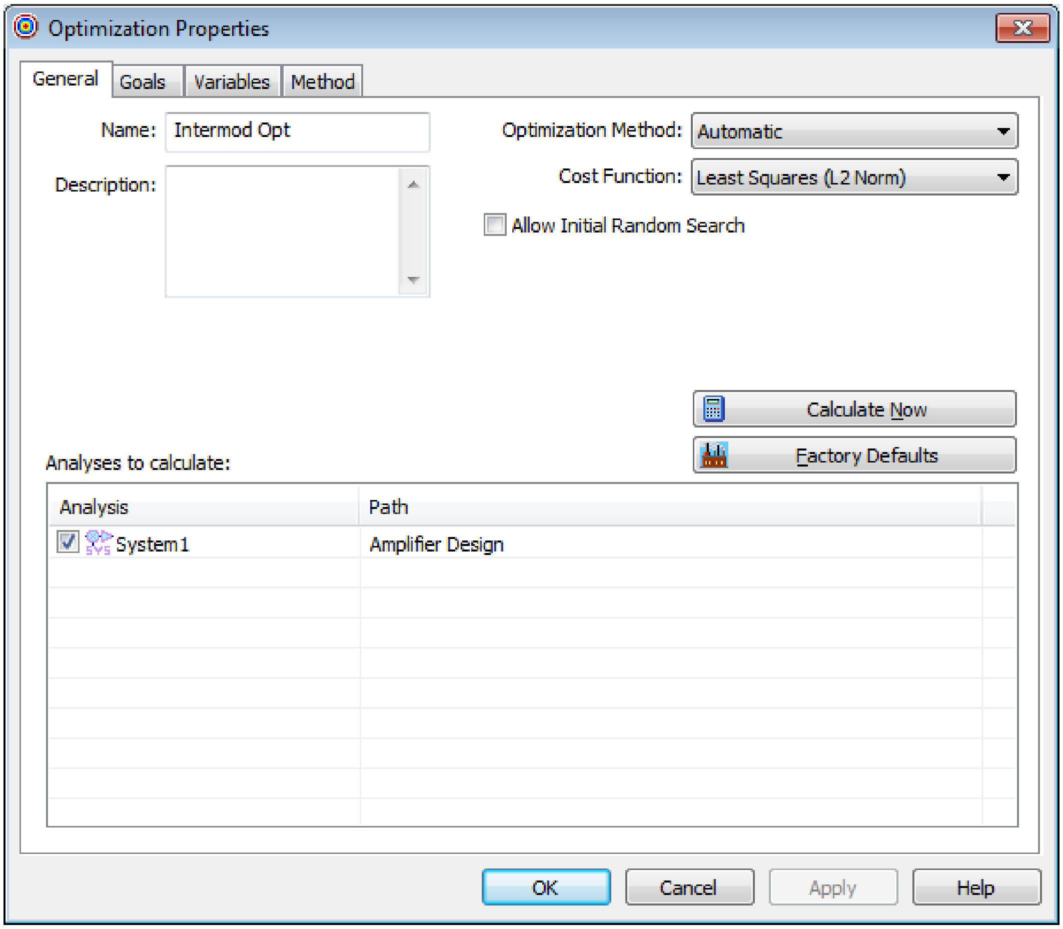
In this tutorial example, you will create and perform an optimization on path measurements in Spectrasys. An optimization tunes variables and recalculates one or more analyzes to evaluate data and determine if a set of goals is met. Any path measurement in Spectrasys can be optimized. The purpose of the example below is to reduce the intermod channel power (TIMCP3) while maintaining the cascaded gain (CGAIN).

Open the workspace *OptimizationUsingSpectrasys.wsv* under <SystemVue Installation Directory>\Examples\Tutorials\Simulation_Control_and_Scripting.

### Creating a New Optimization

1. Create a new optimization by adding a new item and then selecting Add Optimization... from the Evaluations submenu.

2. For the optimization to work, it must have at least one analysis to run. The analyzes that the optimization can run are shown on the General tab. An optimization can run multiple analyzes, but in this example only one analysis is available. Select the System1 analysis.



Name: Intermod Opt

Description:

Optimization Method: Automatic

Cost Function: Least Squares (L2 Norm)

☐ Allow Initial Random Search

Analyses to calculate:

| Analysis | Path |
|---|---|
| ☑ System1 | Amplifier Design |

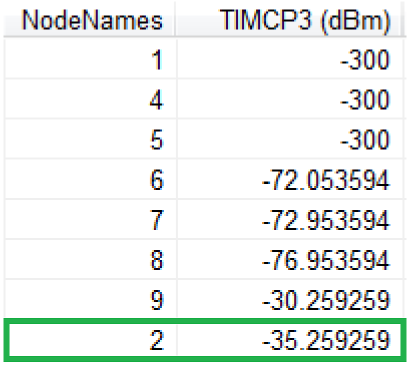
### Creating Goals

To use a Spectrasys path measurement as an optimization goal, you must use the extractsweptdata function to get the measurement at a particular node.

1. Set the Default Dataset to *System1_Data_Path1* since we will create our goals from measurements in this dataset.
2. Using the extractsweptdata data function, define a measurement that will return the TIMCP3 value at the output node of the design (node 2). The syntax should look like the following:

```
extractsweptdata( TIMCP3, NodeNames, '2' )
```

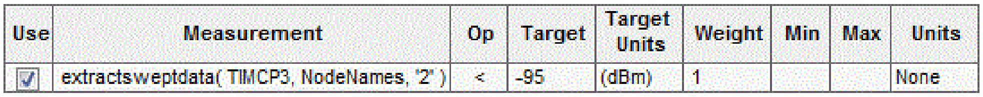
This function will return the TIMCP3 measurement at the node 2 in Path1.



| NodeNames | TIMCP3 (dBm) |
|---|---|
| 1 | -300 |
| 4 | -300 |
| 5 | -300 |
| 6 | -72.053594 |
| 7 | -72.953594 |
| 8 | -76.953594 |
| 9 | -30.259259 |
| 2 | -35.259259 |

3. For this example, we want TIMCP3 at the node 2 to be less than -95 dBm. To enter this as a goal select the less than operator (Op), enter -95 as the Target , and set the Target Units to (dBm).
4. Set the weight to 1 and leave the min and max values blank.

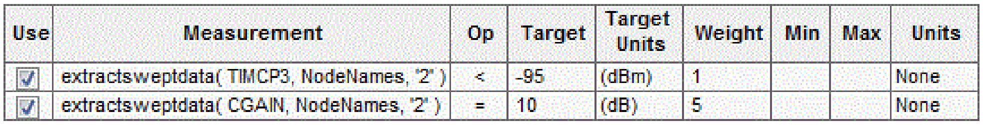
   The first goal should now look like the following:



| Use | Measurement | Op | Target | Target Units | Weight | Min | Max | Units |
|---|---|---|---|---|---|---|---|---|
| ☑ | extractsweptdata( TIMCP3, NodeNames, '2' ) | < | -95 | (dBm) | 1 | | | None |

5. The second goal is to hold the cascaded gain (CGAIN) to 10 dB. You do not want to reduce intermods at the cost of reducing the cascaded gain, so holding the cascaded gain is a more important priority. For optimization, goals with higher weights are considered more important. Try creating the next goal for the CGAIN measurement.

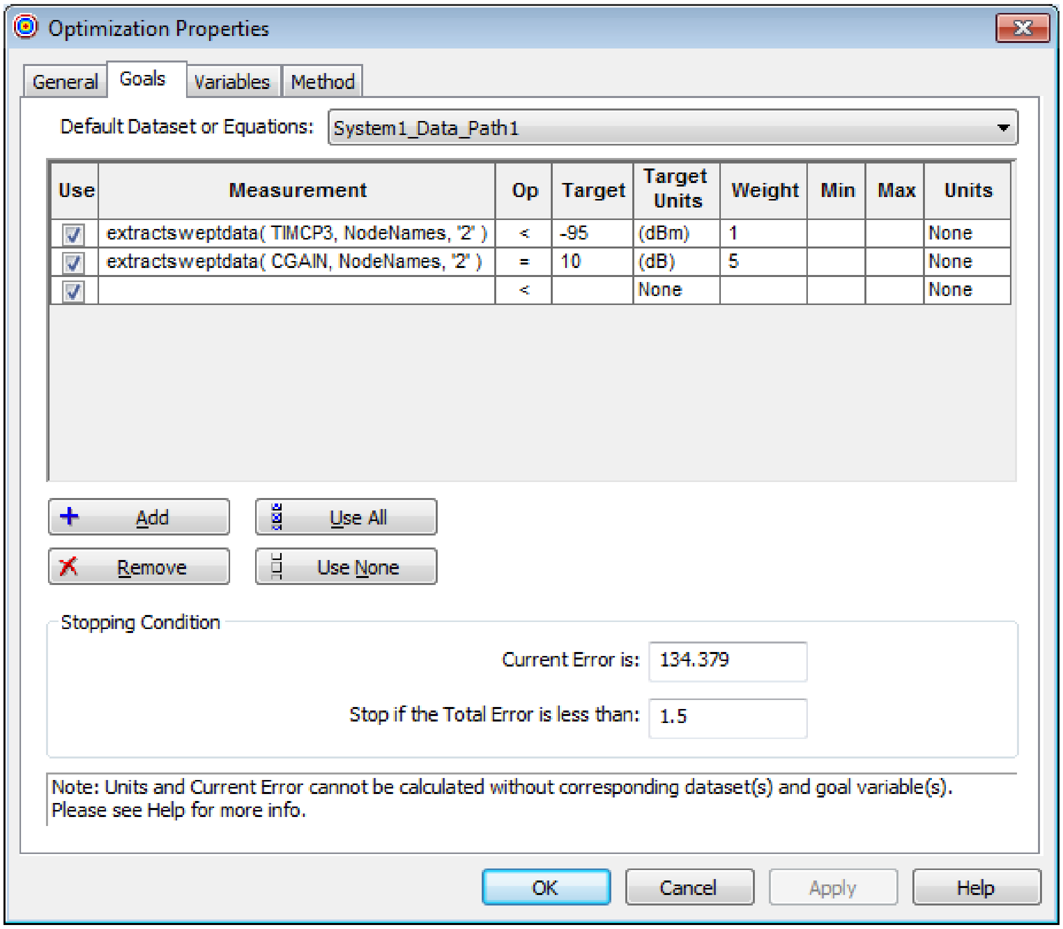
   The second goal might look like the following:



| Use | Measurement | Op | Target | Target Units | Weight | Min | Max | Units |
|---|---|---|---|---|---|---|---|---|
| ☑ | extractsweptdata( TIMCP3, NodeNames, '2' ) | < | -95 | (dBm) | 1 | | | None |
| ☑ | extractsweptdata( CGAIN, NodeNames, '2' ) | = | 10 | (dB) | 5 | | | None |

6. Note the Current error is calculated using current values of the measurements.
7. Enter 1.5 as the Stop if the Total Error is less than value. When running the optimization the Current error will be recalculated between each iteration, and the Best error will be updated whenever the current error is an improvement. Optimization will stop once the Best error is less than 1.5.

**NOTE** There is no maximum iteration limit. Optimization will continue to run so long as the stop condition is not met or until the user selects **Stop**.



Optimization Properties

General | Goals | Variables | Method

Default Dataset or Equations: System1_Data_Path1

| Use | Measurement | Op | Target | Target Units | Weight | Min | Max | Units |
|---|---|---|---|---|---|---|---|---|
| ☑ | extractsweptdata( TIMCP3, NodeNames, '2' ) | < | -95 | (dBm) | 1 | | | None |
| ☑ | extractsweptdata( CGAIN, NodeNames, '2' ) | = | 10 | (dB) | 5 | | | None |
| ☑ | | < | | None | | | | None |

Stopping Condition

Current Error is: 134.379

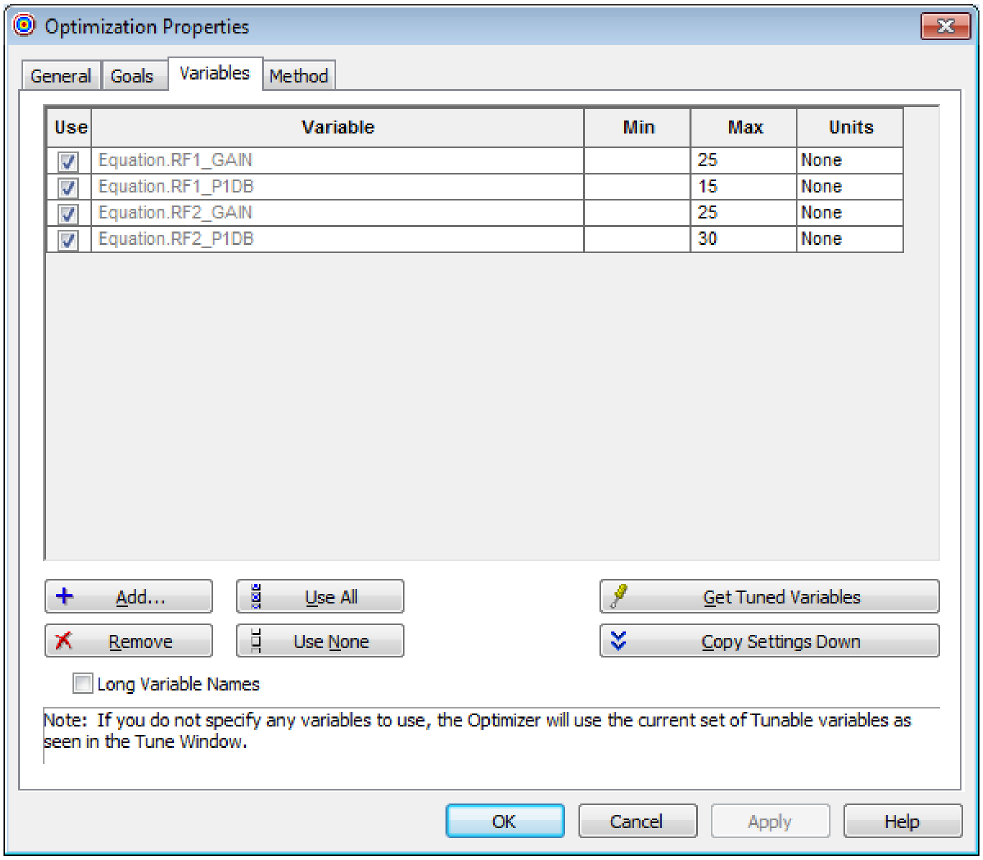
Stop if the Total Error is less than: 1.5

Note: Units and Current Error cannot be calculated without corresponding dataset(s) and goal variable(s).
Please see Help for more info.

### Selecting Variable to Tune

Tuned variables must exist for Optimization to work. Refer to Tuning Variable for how to create a tuned variable. In optimization properties on the Variables tab, we define which tuned variables the optimizer will use. If no tuned variables are specified, then the optimizer will use all available tuned variables by default.

1. In this example, four tuned variable have been defined in the equation block called *Equation*. Select the Get Tuned Variables button to load the tuned variables into the optimization variables tab. These are the variables that will be tuned to achieve the optimization goals.
2. By default, the range for each tuned variable is *0 < variable < infinity*. To reduce optimization time, it is good practice to define a range for each variable. Set the RF1_GAIN and RF2_GAIN max value to 25. Set the max value for RF1_P1DB and RF2_P1DB to 15 and 30 respectively.



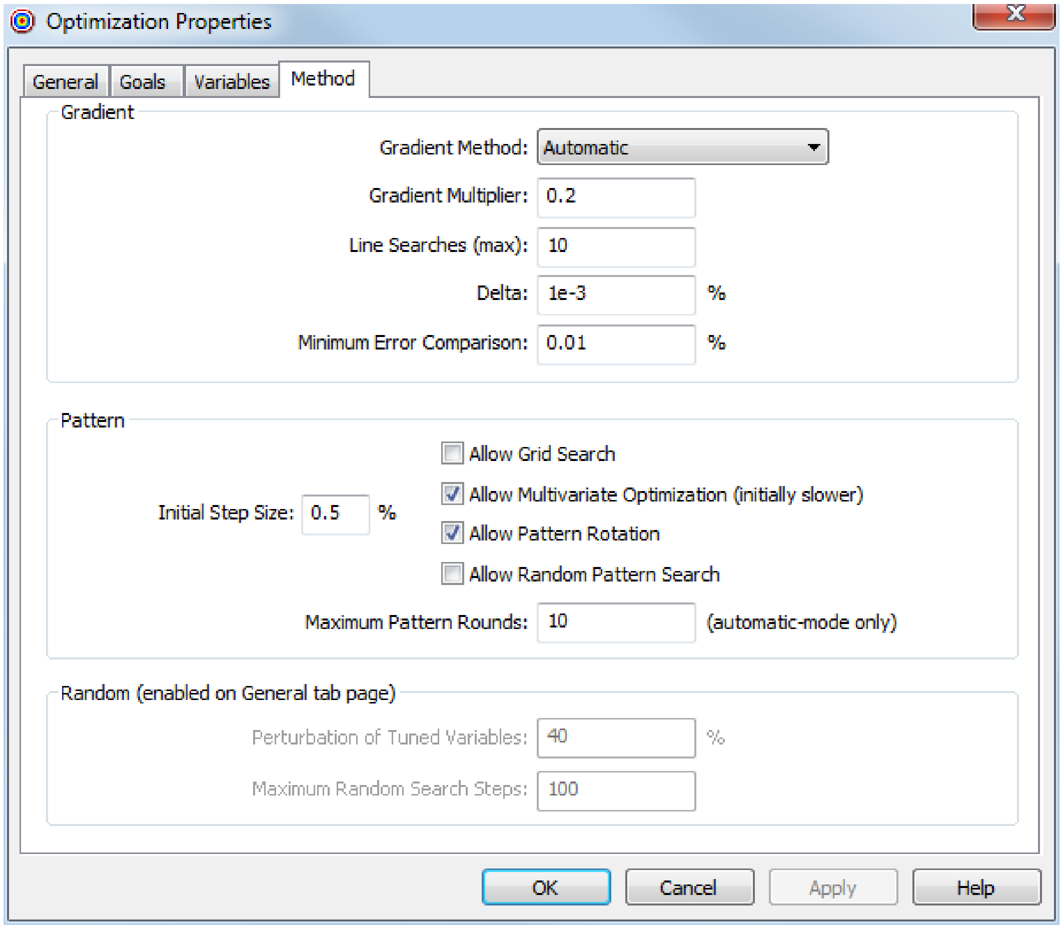
| Use | Variable | Min | Max | Units |
|---|---|---|---|---|
| ☑ | Equation.RF1_GAIN |  | 25 | None |
| ☑ | Equation.RF1_P1DB |  | 15 | None |
| ☑ | Equation.RF2_GAIN |  | 25 | None |
| ☑ | Equation.RF2_P1DB |  | 30 | None |

□ Long Variable Names

Note: If you do not specify any variables to use, the Optimizer will use the current set of Tunable variables as seen in the Tune Window.

## Selecting the Optimization Method

To setup the optimization gradient method and set the pattern options, open the optimization properties and select the Method tab.



Gradient

Gradient Method: Automatic

Gradient Multiplier: 0.2

Line Searches (max): 10

Delta: 1e-3 %

Minimum Error Comparison: 0.01 %

Pattern

Initial Step Size: 0.5 %

☐ Allow Grid Search

☑ Allow Multivariate Optimization (initially slower)

☑ Allow Pattern Rotation

☐ Allow Random Pattern Search

Maximum Pattern Rounds: 10 (automatic-mode only)

Random (enabled on General tab page)

Perturbation of Tuned Variables: 40 %

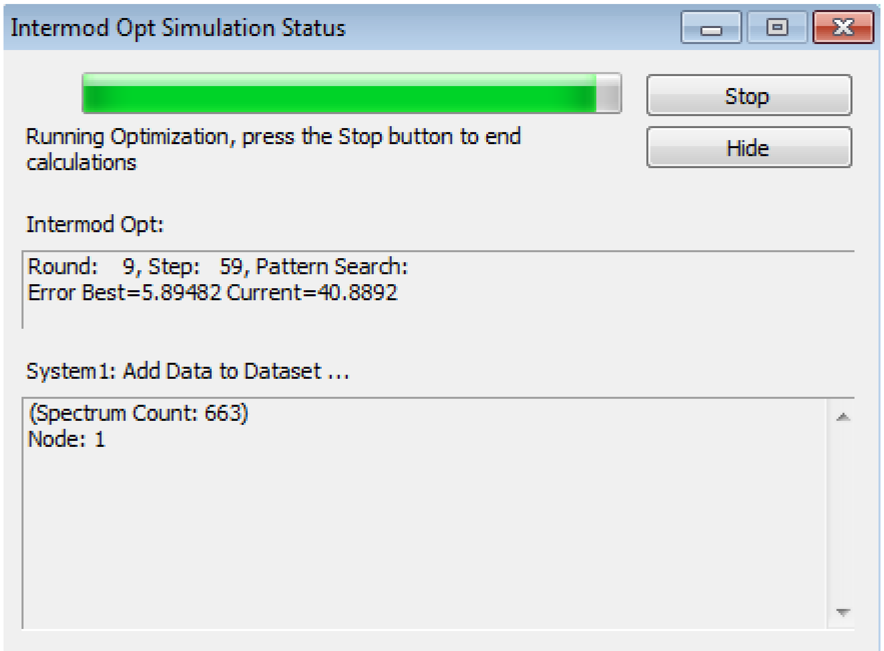
Maximum Random Search Steps: 100

For this example, the gradient method has been set to the default method of Automatic. The Automatic gradient method cycles through all available gradient algorithms until no further improvement is possible. The delta, which is the amount the tuned variables, will be perturbed, has been set to 1e-3. The delta can be increased if the optimization converges too slowly, or decreased it if the optimization fails to improve. The delta should fall within the range 0.1 > delta > 1e-8. For this example, the option Allow Multivariate Optimization has been turned on. All other options have been left as their default setting. For more information on the other options on this page, see Method Tab.

### Running the Optimization

1. To run the Optimization, right click the optimization on the workspace tree and select Run (Calculate).
2. Notice during the optimization in the Simulation Status window reports the Best error and Current error. If the Best error is less than the stopping condition defined in the Goals tab, the optimization will stop.

NOTE If an optimization is manually stopped, optimization will do a final calculation using the tuned values for the **Best** error solution.



Running Optimization, press the Stop button to end calculations

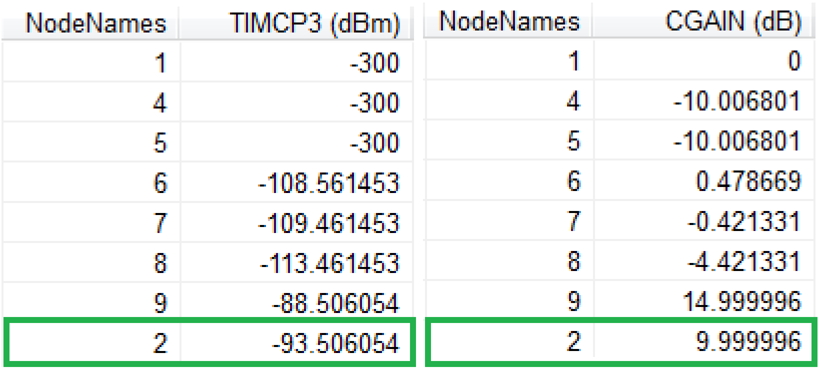
Intermod Opt:

```
Round:    9, Step:   59, Pattern Search:
Error Best=5.89482 Current=40.8892
```

System1: Add Data to Dataset ...

```
(Spectrum Count: 663)
Node: 1
```

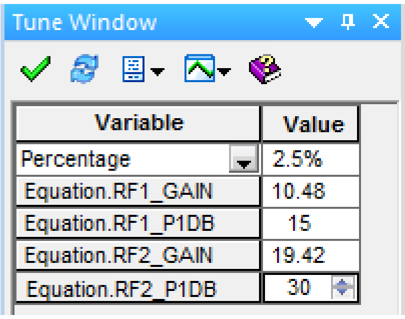
3. Optimization reaches an error of about 1.49 after about 30 seconds (time will vary per computer). Open the System1_Data_Path1 dataset and look at the CGAIN and TIMCP3 variables to confirm that the optimization has produced expected results. CGAIN is approximately equal to 10 dB, and the TIMCP3 is -93.5 dBm. The value -93.5 is not below the goal we original defined, but because we set the stopping error condition to 1.5 this is acceptable.



| NodeNames | TIMCP3 (dBm) |
|---|---|
| 1 | -300 |
| 4 | -300 |
| 5 | -300 |
| 6 | -108.561453 |
| 7 | -109.461453 |
| 8 | -113.461453 |
| 9 | -88.506054 |
| 2 | -93.506054 |

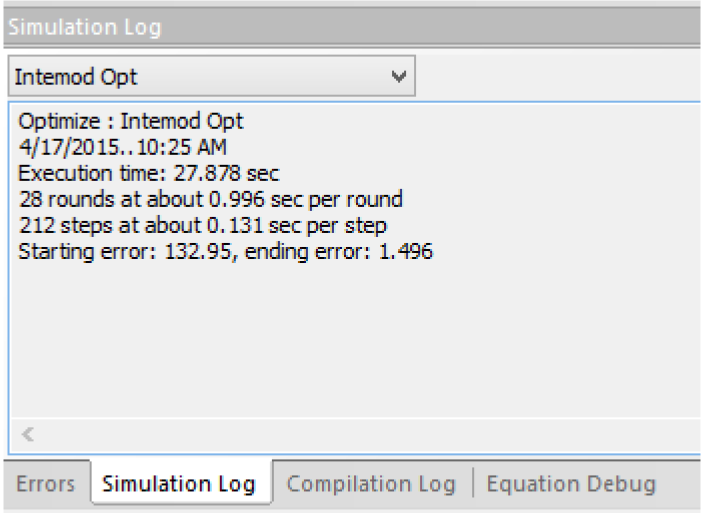
| NodeNames | CGAIN (dB) |
|---|---|
| 1 | 0 |
| 4 | -10.006801 |
| 5 | -10.006801 |
| 6 | 0.478669 |
| 7 | -0.421331 |
| 8 | -4.421331 |
| 9 | 14.999996 |
| 2 | 9.999996 |

4. On completion, the tuned values for the best error are calculated. View the tuned window or equation block to see that the tune variables have changed.



| Variable | Value |
|---|---|
| Percentage | 2.5% |
| Equation.RF1_GAIN | 10.48 |
| Equation.RF1_P1DB | 15 |
| Equation.RF2_GAIN | 19.42 |
| Equation.RF2_P1DB | 30 |

5. View the simulation log to see information about the optimization run.



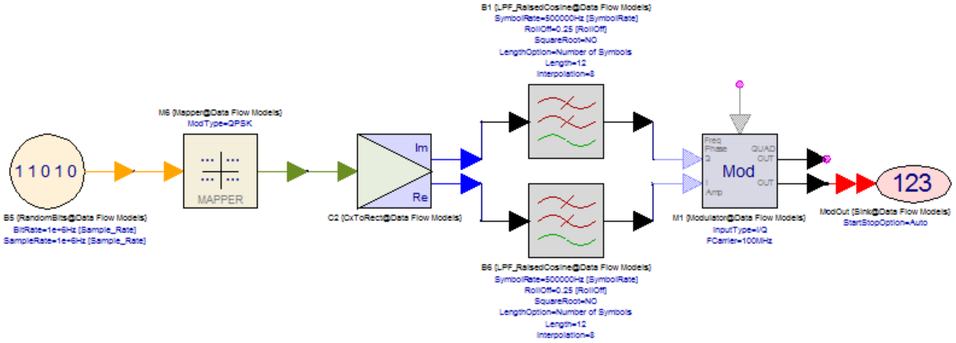
```
Optimize : Intemod Opt
4/17/2015..10:25 AM
Execution time: 27.878 sec
28 rounds at about 0.996 sec per round
212 steps at about 0.131 sec per step
Starting error: 132.95, ending error: 1.496
```

For more information, refer to the Optimization.

## Peak to Average Power Ratio Optimization

This tutorial example shows how to perform an optimization for a Data Flow design. The optimization setup is using one goal and one variable.

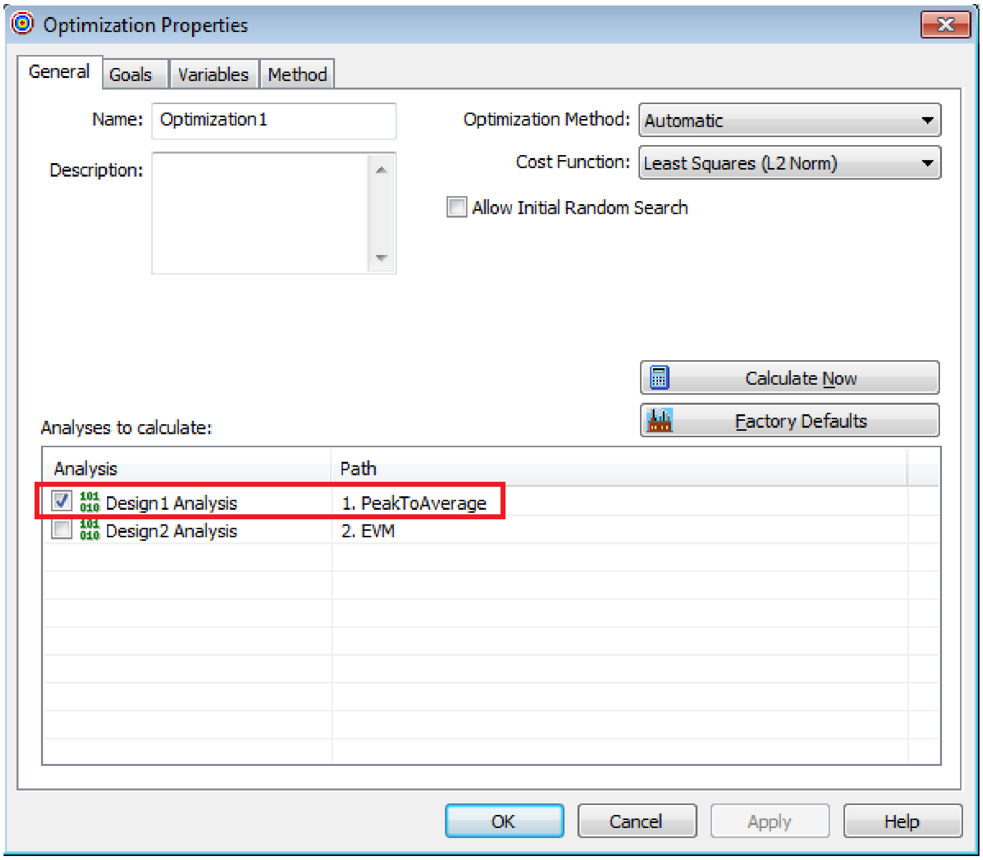
1. Open the example <SystemVue Installation Directory>\Examples\Tutorials\Simulation_Control_and_Scripting\Optimizatior wsv
2. Open *Design1* in the *1. PeakToAverage* folder. This design generates a QPSK signal with a symbol rate of 0.5 MHz sampled at 4 MHz.


B1 {LPF_RaisedCosine@Data Flow Models}
SymbolRate=500000Hz [SymbolRate]
RollOff=0.25 [RollOff]
SquareRoot=NO
LengthOption=Number of Symbols
Length=12
Interpolation=8
M6 {Mapper@Data Flow Models}
ModType=QPSK
1 1 0 1 0
MAPPER
Im
Re
Freq
Phase
Q
QUAD OUT
Mod
OUT
I
Amp
123
B5 {RandomBits@Data Flow Models}
BitRate=1e+6Hz [Sample_Rate]
SampleRate=1e+6Hz [Sample_Rate]
C2 {CxToRect@Data Flow Models}
M1 {Modulator@Data Flow Models}
InputType=I/Q
FCarrier=100MHz
ModOut {Sink@Data Flow Models}
StartStopOption=Auto
B6 {LPF_RaisedCosine@Data Flow Models}
SymbolRate=500000Hz [SymbolRate]
RollOff=0.25 [RollOff]
SquareRoot=NO
LengthOption=Number of Symbols
Length=12
Interpolation=8

3. Run *Design1 Analysis* and open the Equations page *PostProcessingEqns1* to see what the peak to average power ratio is (*PeakToAverageRatio* variable). The current system settings result in a peak to average power ratio of 5.46 dB. This may be too high for a power amplifier that may be connected to the output of the modulator so our goal is to set up an optimization to reduce the peak to average power ratio to less than 3 dB. The variable we are going to vary is the raised cosine filter *RollOff* factor.

4. Right-click the folder *1. PeakToAverage* and select *Add* > *Evaluations* > *Add Optimization*.

   In the *General* tab, select *Design1 Analysis* in the *Analyzes to calculate* the area.



Optimization Properties

Name: Optimization1

Description:

Optimization Method: Automatic

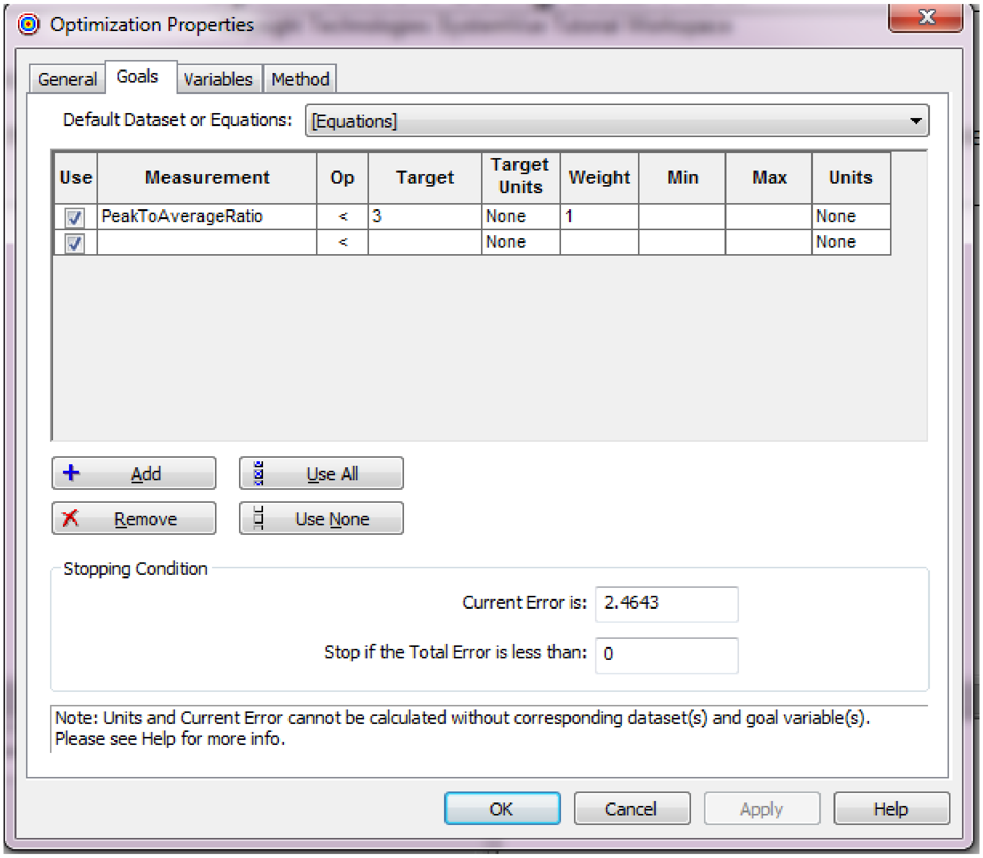
Cost Function: Least Squares (L2 Norm)

□ Allow Initial Random Search

Analyses to calculate:

| Analysis | Path |
|---|---|
| ☑ Design1 Analysis | 1. PeakToAverage |
| □ Design2 Analysis | 2. EVM |

In the *Goals* tab, set *Default Dataset or Equations* to [Equations] and in the grid below add the *PeakToAverageRatio* measurement with a target of < 3.



Optimization Properties

Default Dataset or Equations: [Equations]

| Use | Measurement | Op | Target | Target Units | Weight | Min | Max | Units |
|---|---|---|---|---|---|---|---|---|
| ☑ | PeakToAverageRatio | < | 3 | None | 1 | | | None |
| ☑ | | < | | None | | | | None |

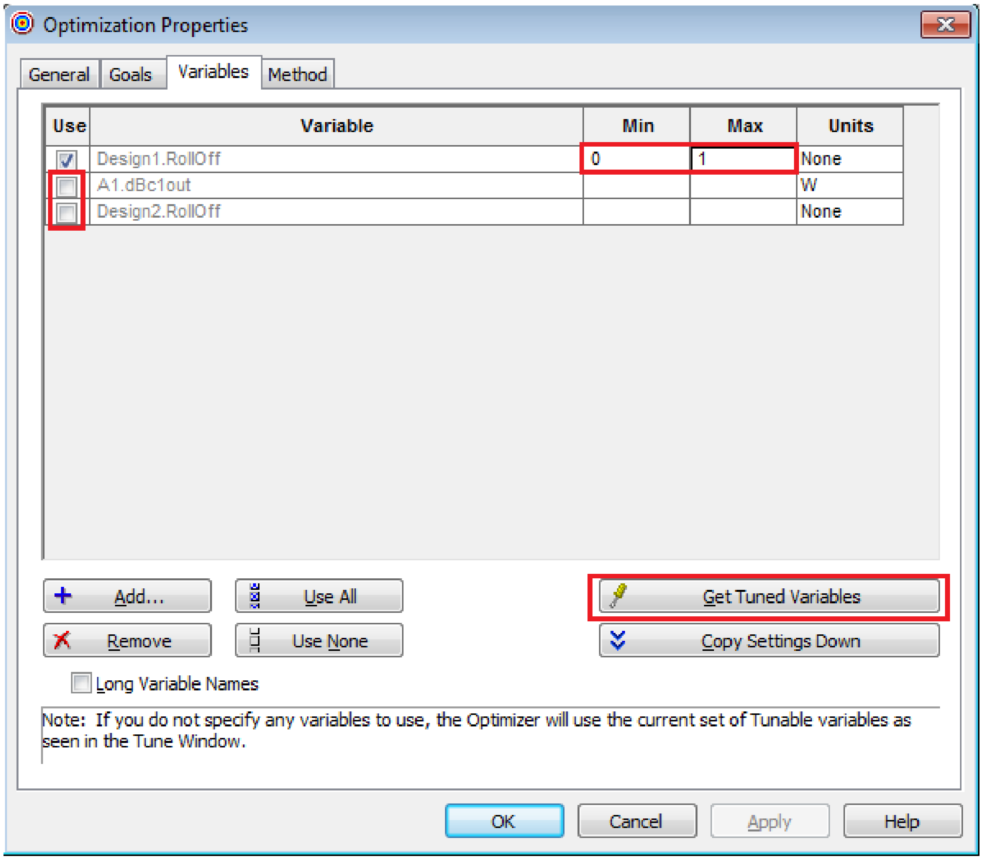
Stopping Condition

Current Error is: 2.4643

Stop if the Total Error is less than: 0

Note: Units and Current Error cannot be calculated without corresponding dataset(s) and goal variable(s). Please see Help for more info.

In the *Variables* tab, press the Get Tuned Variables button. This will populate the grid with all tuned variables (tuned variables are part parameters that have their Tune checkbox checked or variables in Equations pages that are assigned using the =tune operator; see Equations tab of *Design1*). Uncheck the checkboxes next to *A1.dBc1out* and *Design2.RollOff*. Finally define the Min and Max values for *Design1.RollOff*.



| Use | Variable | Min | Max | Units |
|---|---|---|---|---|
| ☑ | Design1.RollOff | 0 | 1 | None |
| ☐ | A1.dBc1out | | | W |
| ☐ | Design2.RollOff | | | None |

☐ Long Variable Names

Note: If you do not specify any variables to use, the Optimizer will use the current set of Tunable variables as seen in the Tune Window.

Leave everything in the *Method* tab in its default state and press the OK button to apply the changes you made.

5. Now run the optimization by right clicking on the *Optimization1* item on the workspace tree and selecting *Run (calculate now)*. The optimizer starts and after a few seconds, it achieves the goal of 3 dB or less peak to average ratio. Open the Equations page *PostProcessingEqns1* and verify that the *PeakToAverageRatio* is less than 3. The value should be close to 2.576. If you open *Desing1* and zoom in the filters area, you will see that the value of *RollOff* factor that achieved this 2.576 dB peak to average ratio is 0.773 (you can also see this value in the *Equations* tab of *Design1*).

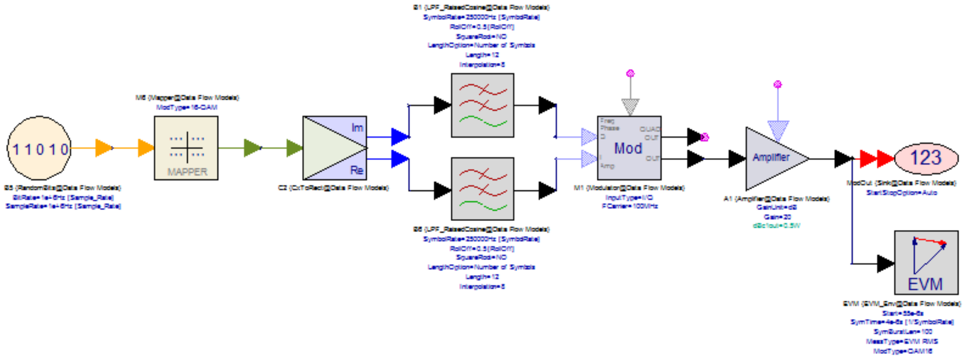
6. The *RollOff* factor of 0.773 gave us almost half a dB of safety guard from our 3 dB target. However, the higher value of *RollOff* factor results in higher bandwidth for the output signal. Let's try to reduce the *RollOff* factor but still meet the peak to average 3 dB target.

7. Double click on *Optimization1* and in the *Goals* tab add a second goal defined by *PeakToAverageRatio* > 2.75 with the weight of 1. Run the optimizer again and after it completes observe the peak to average ratio in *PostProcessingEqns1* (2.89) and the *RollOff* factor (0.709).

## EVM Optimization

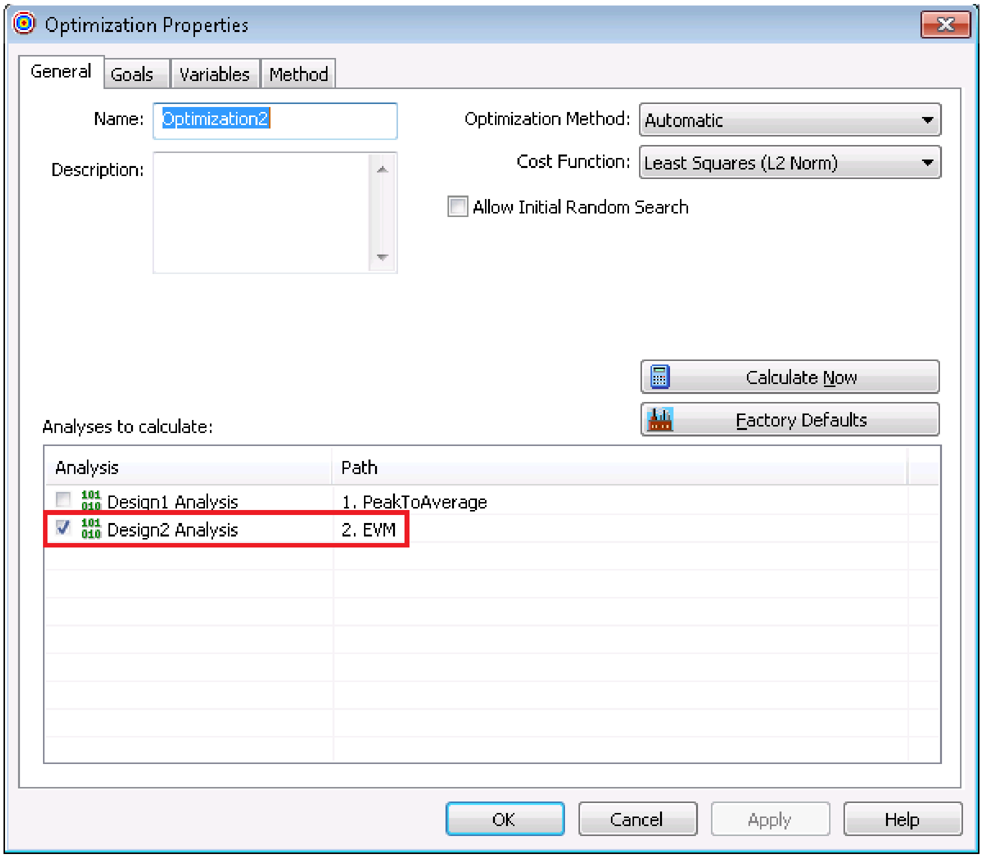
This tutorial example shows how to perform an optimization for a Data Flow design. The optimization setup is using one goal and one variable.

1. Open the example <SystemVue Installation Directory>\Examples\Tutorials\Simulation_Control_and_Scripting\Optimizatior wsv
2. Open *Design2* in the *2. EVM* folder. This is the same system as in *Design1* (it generates a QPSK signal with a symbol rate of 0.5 MHz sampled at 4 MHz) with the addition of a non-linear amplifier at the output of the modulator.


B1 (LPF_RaisedCosine@Data Flow Models)
SymbolRate= 250000Hz [SymbolRate]
RollOff= 0.5 [RollOff]
SquareRoot= NO
LengthOption= Number of Symbols
Length= 12
Interpolation= 8
M6 (Mapper@Data Flow Models)
ModType= 16-QAM
1 1 0 1 0
MAPPER
Im
Re
Freq
Phase
Q
I
Amp
QUAD OUT
OUT
Mod
Amplifier
123
B5 (RandomBits@Data Flow Models)
BitRate= 1e+6Hz [Sample_Rate]
SampleRate= 1e+6Hz [Sample_Rate]
C2 (CxToRect@Data Flow Models)
M1 (Modulator@Data Flow Models)
InputType= I/Q
FCarrier= 100MHz
A1 (Amplifier@Data Flow Models)
GainUnit= dB
Gain= 20
dBc1out= 0.5W
ModOut (Sink@Data Flow Models)
StartStopOption= Auto
B6 (LPF_RaisedCosine@Data Flow Models)
SymbolRate= 250000Hz [SymbolRate]
RollOff= 0.5 [RollOff]
SquareRoot= NO
LengthOption= Number of Symbols
Length= 12
Interpolation= 8
EVM
EVM (EVM_Env@Data Flow Models)
Start= 55e-6s
SymTime= 4e-6s [1/SymbolRate]
SymBurstLen= 100
MeasType= EVM RMS
ModType= QAM16

3. Run *Design2 Analysis* and observe the EVM (*AmpOut_EVM_RMS* variable) in the dataset *Design2_Data*. The measured EVM is 7.6%. Let's set up an optimization to reduce EVM to less than 3% by varying the amplifier's *dBc1out* parameter.
4. Right-click on the folder *2. EVM* and select *Add* > *Evaluations* > *Add Optimization...*.

   In the *General* tab, select *Design2 Analysis* in the *Analyzes to calculate* area.



Optimization Properties

Name: Optimization2

Description:

Optimization Method: Automatic

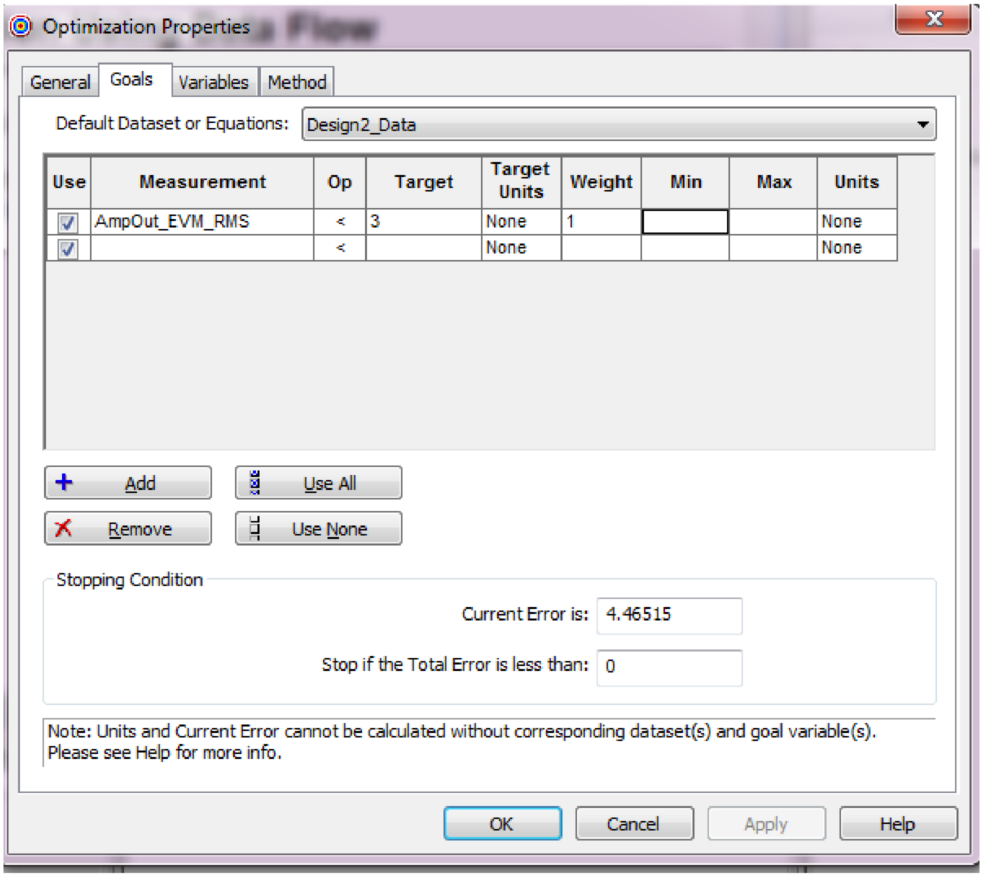
Cost Function: Least Squares (L2 Norm)

□ Allow Initial Random Search

Analyses to calculate:

| | Analysis | Path |
|---|---|---|
| □ | Design1 Analysis | 1. PeakToAverage |
| ☑ | Design2 Analysis | 2. EVM |

In the *Goals* tab, set *Default Dataset or Equations* to *Desing2_Data* and in the grid below add the *AmpOut_EVM_RMS* measurement with a target of < 3.



Default Dataset or Equations: Design2_Data

| Use | Measurement | Op | Target | Target Units | Weight | Min | Max | Units |
|---|---|---|---|---|---|---|---|---|
| ☑ | AmpOut_EVM_RMS | < | 3 | None | 1 | | | None |
| ☑ | | < | | None | | | | None |

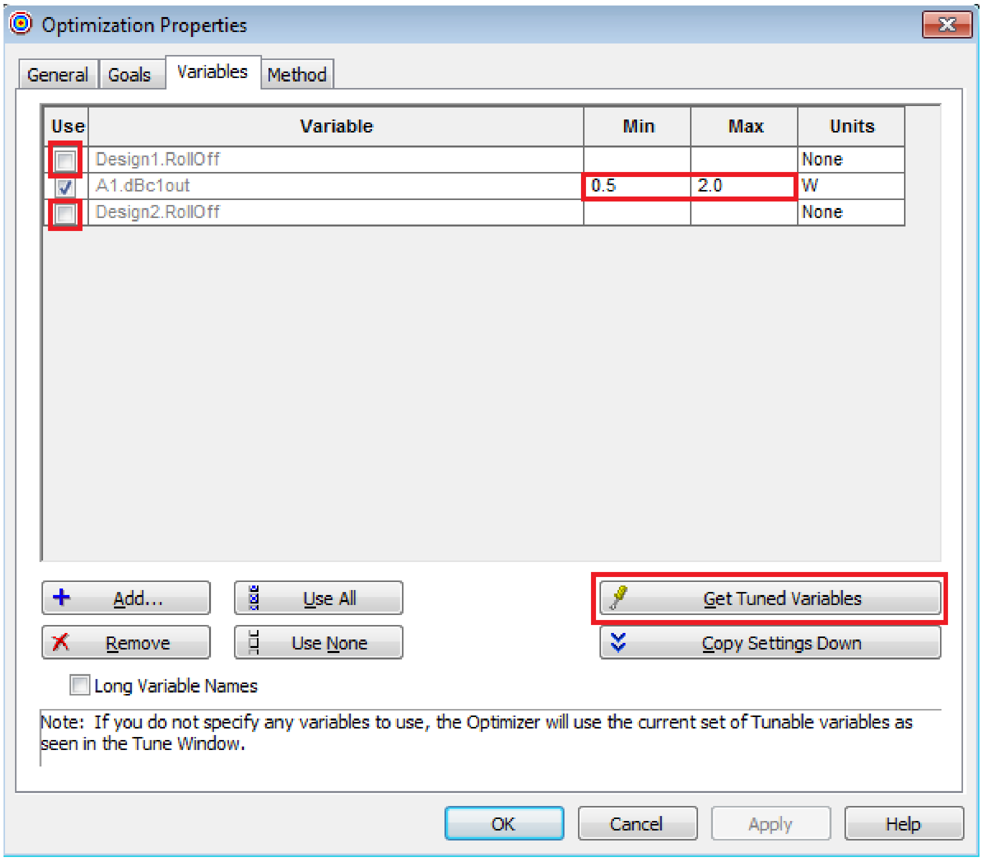
Stopping Condition

Current Error is: 4.46515

Stop if the Total Error is less than: 0

Note: Units and Current Error cannot be calculated without corresponding dataset(s) and goal variable(s). Please see Help for more info.

In the *Variables* tab, press the Get Tuned Variables button. This will populate the grid with all tuned variables (tuned variables are part parameters that have their Tune checkbox checked or variables in Equations pages that are assigned using the =? operator; see Equations tab of *Design1*). Uncheck the checkboxes next to *Design1.RollOff* and *Design2.RollOff*. Finally define the Min and Max values for *A1.dBc1out*.



| Use | Variable | Min | Max | Units |
|---|---|---|---|---|
| ☐ | Design1.RollOff | | | None |
| ☑ | A1.dBc1out | 0.5 | 2.0 | W |
| ☐ | Design2.RollOff | | | None |

☐ Long Variable Names

Note: If you do not specify any variables to use, the Optimizer will use the current set of Tunable variables as seen in the Tune Window.

Leave everything in the *Method* tab in its default state and press the OK button to apply the changes you made.

5. Now run the optimization and after it completes open the dataset *Desing2_Data* and verify that the *AmpOut_EVM_RMS* is less than 3%. The value should be close to 2.0%. If you open *Desing2* and zoom in the amplifier area, you will see that the value of *dBc1out* parameter that achieved the desired EVM is 1.546 W.

## Using MATLAB Script For Sequence Control

### Overview

Many applications require running multiple simulations sequentially. For example, in an LTE Bit Error Rate (BER) measurement over a device, one simulation can generate waveform(s) that will be downloaded into RF Signal Synthesizer(s) to modulate the RF signals that will stimulate the device. Another simulation will then use measurement equipment such as the Keysight Technologies MXA's to capture the output RF signal from the device and feed the measured data back into the simulation to be demodulated for BER analysis.

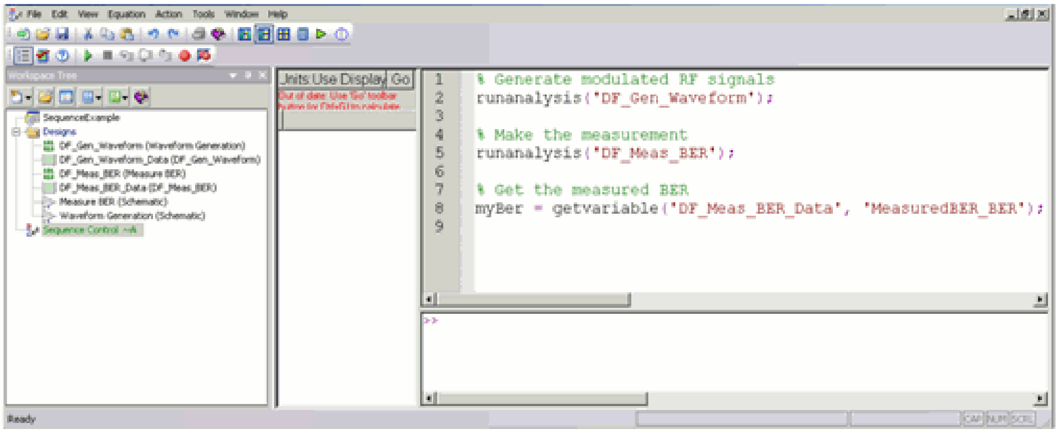
Furthermore, in order to characterize the device's performance, it might be necessary to adjust certain settings of some instruments several times and make the measurements after each instrument adjustment. For example, it might be necessary to change a DC bias level and see how the BER is impacted by it.

These are the applications where sequence control can be used.

SystemVue provides a powerful and flexible sequence control mechanism that is based on MATLAB scripting.

Refer to Using Command Expert In MATLAB Script on the typical way of controlling instrument from MATLAB Script scripting environment. For LXI-compliant instruments, you can alternatively use Tcpip.

## A Simple Sequence


File
Edit
View
Equation
Action
Tools
Window
Help
Workspace Tree
SequenceExample
Designs
DF_Gen_Waveform (Waveform Generation)
DF_Gen_Waveform_Data (DF_Gen_Waveform)
DF_Meas_BER (Measure BER)
DF_Meas_BER_Data (DF_Meas_BER)
Measure BER (Schematic)
Waveform Generation (Schematic)
Sequence Control ~A
Units Use Display
Go
Out of date. Use 'Go' toolbar
Ready
CAP
NUM
SCRL

```
% Generate modulated RF signals
runanalysis('DF_Gen_Waveform');

% Make the measurement
runanalysis('DF_Meas_BER');

% Get the measured BER
myBer = getvariable('DF_Meas_BER_Data', 'MeasuredBER_BER');

```

```
>>
```

In the above example, there are two simulations:

- DF_Gen_Waveform(Waveform Generation) that performs a Data Flow Simulation over the Waveform Generation(Schematic) design
- DF_Meas_BER(Measure BER) that performs a Data Flow Simulation over the Measure BER (Schematic) design

The critical MATLAB Script built-in functions used are:

- runanalysis - executes the specified Data Flow Simulation
- getvariable - gets the simulation result data

Obviously, the BER result is stored in a variable named MeasuredBER_BER inside the simulation results of DF_Meas_BER_Data(DF_Meas_BER).

CAUTION Notice that the **Sequence Control ~ A** MATLAB Script Equation page, i.e. the script, is located at the same level on the workspace tree as the workspace (i.e. project) name.

## How to Run the Sequence

You can use either of the following two ways to run the sequence (the sequence MATLAB Script Equation page must be open):

- click the GREEN triangle button (the 4th icon) on the second toolbar
- click the Go button next to the Equation editor area

### Example of a more Advanced Sequence

In the following sequence, we will vary the DC bias (provided by an LXI compliant DC supply), measure the BER at each of the different bias levels, and finally put the measured BER results into the simulation results.
The additional MATLAB Script function used in this example are:

- setvariable - brings the value stored in a variable into the measurement results storage area (i.e. Data Set) of the simulation
- num2str - converts a number to a string
- fprintf- writes a string to the opened tcpip port

NOTE The use of the **[]** operation to concatenate the strings when creating the **dcCmdStr** command string

CAUTION How the accumulated BER results are stored in the **myBers** variable and how this variable is **transposed** with the **'** operator when calling **setvariable(...)** on the last line.

```
% 5 DC Levels starting at 3.5V at a step of 0.5V
DCLevels = (3.5:0.5:5.0);

% Number of DC's
numDCs = length(DCLevels);

% Place holder for the 5 BER's to be measured
myBers = zeros(1, numDCs);

% Generate modulated RF signals
runanalysis('DF_Gen_Waveform');

% Create tcpip communication with DC supply
dcSply = tcpip('111.222.333.444', 5025);
fopen(dcSply);

% Loop the DC levels and make the measurement
% at each level
for idx = 1:1:numDCs
    dcCmdStr = [':VOLT ' num2str(DCLevels(idx))];
    fprintf(dcSply, dcCmdStr);
    % make sure the DC is settled
    fprintf(dcSply, '*OPC?');
    statusRes = fgets(dcSply);

    % Measure BER at this DC bias
    runanalysis('DF_Meas_BER');

    % Get the measured BER and store it away
    myBers(idx) = getvariable('DF_Meas_BER_Data', 'Measu
redBER_BER');
end

% Now close communication with DC supply
fclose(dcSply);

% Now bring the stored 5 BER's into the simulation
results
% and name the variable AllBers
setvariable( 'DF_Meas_BER_Data', 'AllBers', myBers' );
```

## Performing a Monte Carlo Analysis on a Design

### Performing a Monte Carlo Evaluation on a Design

This section has a series of tutorials that demonstrate how to run a Monte Carlo evaluation on designs created in Data Flow or Spectrasys.

### Monte Carlo for Data Flow Designs

1. Monte Carlo Data Flow Example

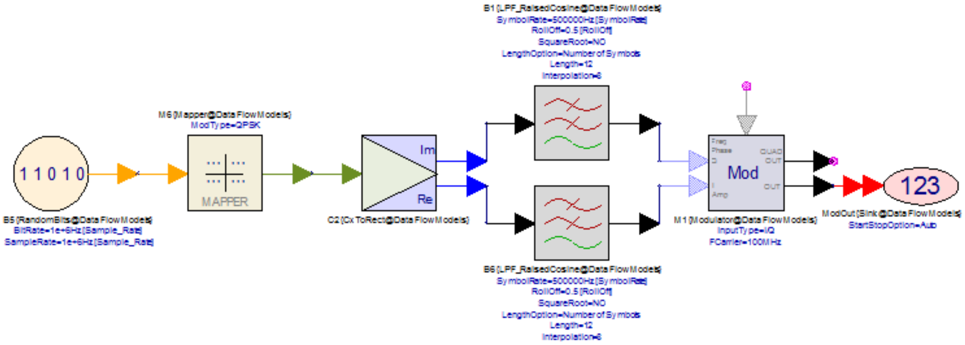
### Monte Carlo for Spectrasys Designs

1. Monte Carlo Spectrasys Example

### Monte Carlo Data Flow Example

This tutorial example shows how to perform a Monte Carlo evaluation on a Data Flow design.

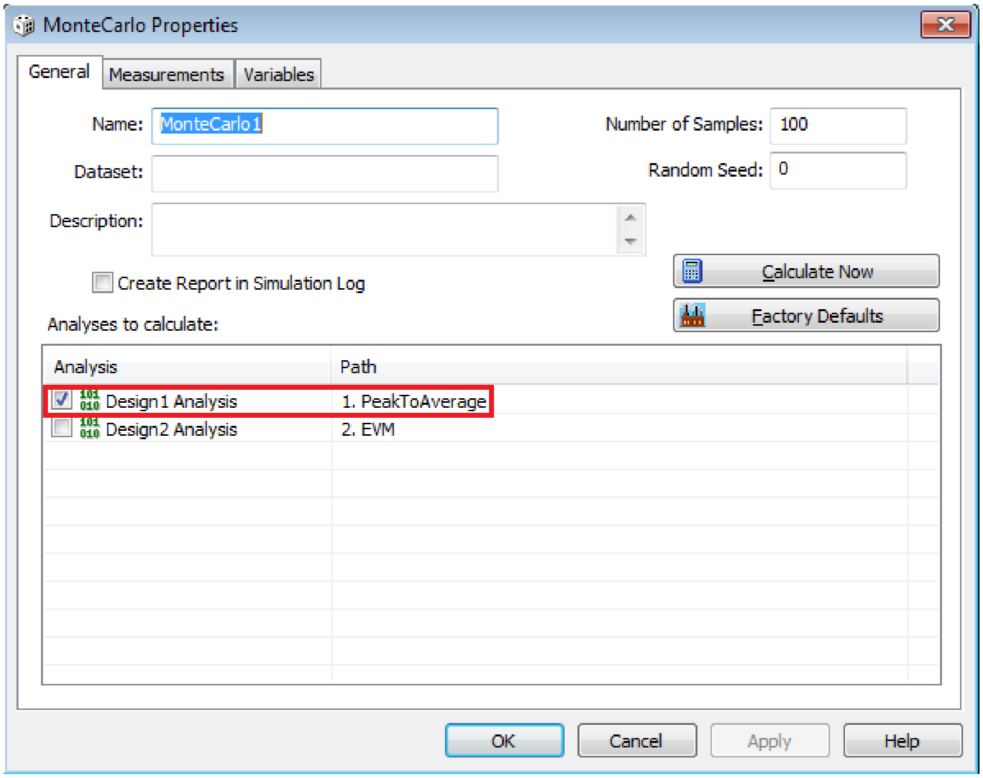
1. Open the example <SystemVue Installation Directory>\Examples\Tutorials\Simulation_Control_and_Scripting\MonteCarlo
wsv
2. Open *Design1* in the *1. PeakToAverage* folder. This design generates a QPSK signal with a symbol rate of 0.5 MHz sampled at 4 MHz.


B1 {LPF_RaisedCosine@Data Flow Models}
SymbolRate=500000Hz [SymbolRate]
RollOff=0.5 [RollOff]
SquareRoot=NO
LengthOption=Number of Symbols
Length=12
Interpolation=8
M6 {Mapper@Data Flow Models}
ModType=QPSK
1 1 0 1 0
MAPPER
Im
Re
Freq
Phase
Q
I
Amp
QUAD OUT
OUT
Mod
123
B5 {RandomBits@Data Flow Models}
BitRate=1e+6Hz [Sample_Rate]
SampleRate=1e+6Hz [Sample_Rate]
C2 {CxToRect@Data Flow Models}
M1 {Modulator@Data Flow Models}
InputType=IQ
FCarrier=100MHz
ModOut {Sink@Data Flow Models}
StartStopOption=Auto
B6 {LPF_RaisedCosine@Data Flow Models}
SymbolRate=500000Hz [SymbolRate]
RollOff=0.5 [RollOff]
SquareRoot=NO
LengthOption=Number of Symbols
Length=12
Interpolation=8

3. Run *Design1 Analysis* and open the Equations page *PostProcessingEqns1* to see what the peak to average power ratio is (*PeakToAverageRatio* variable). The current system settings result in a peak to average ratio of 3.95 dB. With a Monte Carlo simulation, you can see how a certain measurement varies when we statistically vary a system parameter. In this example, we are going to vary (based on a uniform distribution from 0 to 1) the raised cosine filter *RollOff* factor and see how the peak to average power ratio is affected.

4. Right-click on the folder *1. PeakToAverage* and select *Add* > *Evaluations* > *Add Monte Carlo Evaluation...*.

   In the *General* tab, select *Design1 Analysis* under *Analyses to calculate*.



MonteCarlo Properties

Name: MonteCarlo1

Dataset:

Description:

Number of Samples: 100

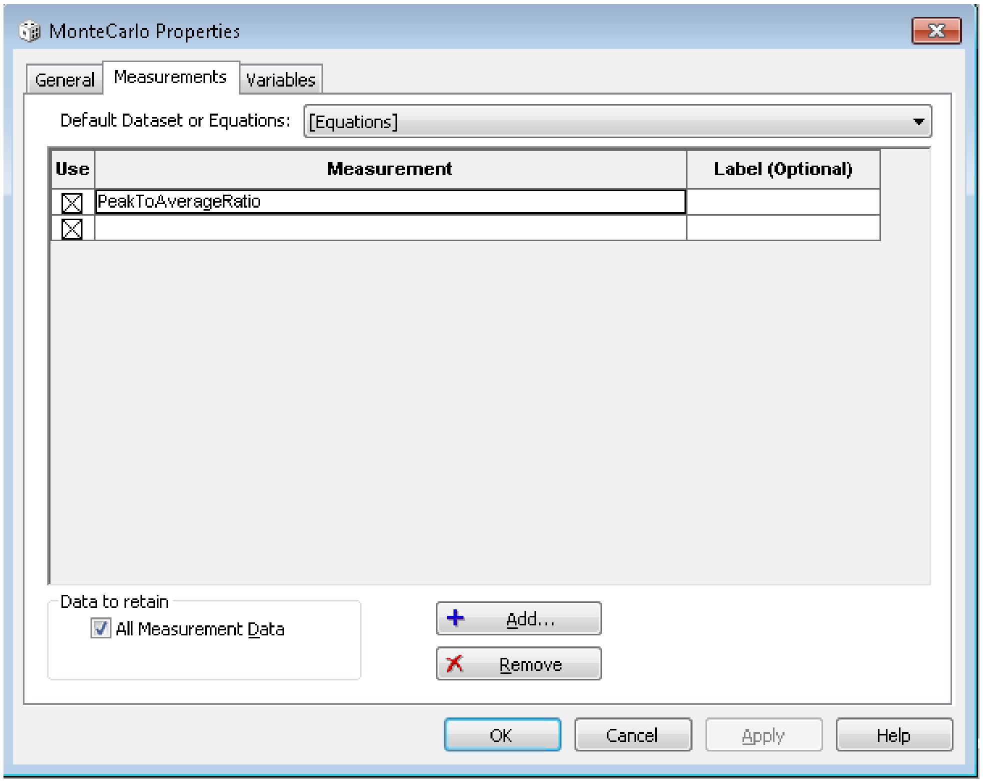
Random Seed: 0

☐ Create Report in Simulation Log

Analyses to calculate:

| Analysis | Path |
|---|---|
| ☑ Design1 Analysis | 1. PeakToAverage |
| ☐ Design2 Analysis | 2. EVM |

   In the *Measurements* tab, set *Default Dataset or Equations* to [Equations] and in the grid below add the *PeakToAverageRatio* measurement.



Default Dataset or Equations: [Equations]

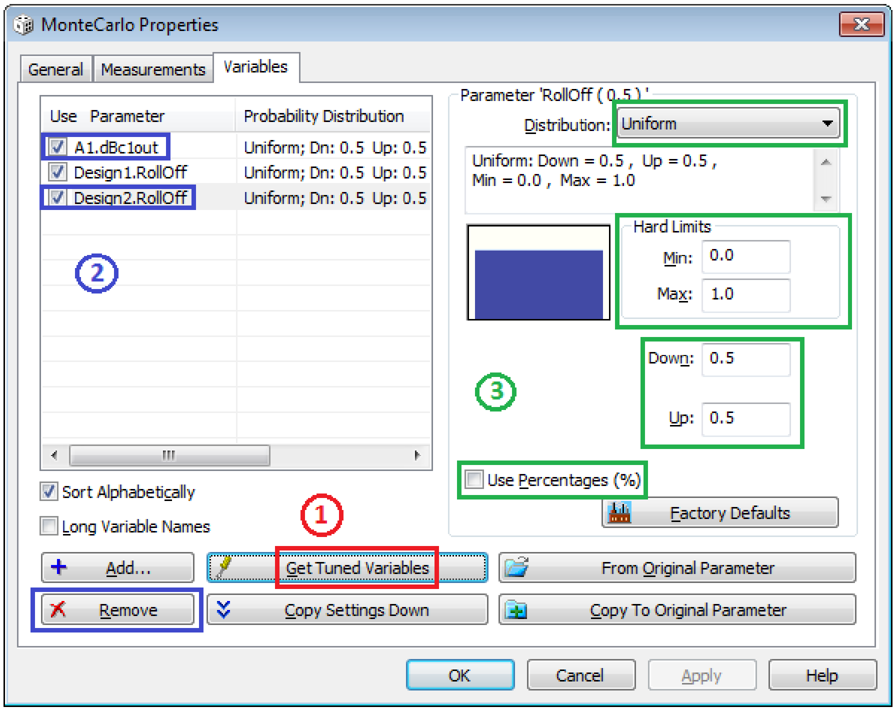
| Use | Measurement | Label (Optional) |
|---|---|---|
| ☒ | PeakToAverageRatio | |
| ☒ | | |

Data to retain

☑ All Measurement Data

In the *Variables* tab

- Press the Get Tuned Variables button. This will populate the grid in the top left area with all tuned variables (tuned variables are part parameters that have their Tune checkbox checked or variables in Equations pages that are assigned using the =tune operator; see Equations tab of *Design1*).
- Select each of the rows with *A1.dBc1out* and *Design2.RollOff* and press the Remove button to remove them.
- Select the remaining row (where *Design1.RollOff* factor is defined) and set its distribution (top right corner) to Uniform. Uncheck the Use Percentages (%) checkbox. Now define by how much the variable will be perturbed (down and up) from its nominal value (shown in the top right area just above the Distribution drop-down menu). For our example the nominal value of *Design1.RollOff* is 0.5 so in order to perturb it based on a uniform distribution between 0 and 1, we need to set the *Down* field to 0.5 and the *Up* field to 0.5. We can also set the *Min* and *Max Hard Limits* (this applies more to distributions that can generate arbitrarily big magnitude values like the *Normal*, *Beta*, *Lognormal*).



MonteCarlo Properties

General | Measurements | Variables

| Use | Parameter | Probability Distribution |
|---|---|---|
| ☑ | A1.dBc1out | Uniform; Dn: 0.5 Up: 0.5 |
| ☑ | Design1.RollOff | Uniform; Dn: 0.5 Up: 0.5 |
| ☑ | Design2.RollOff | Uniform; Dn: 0.5 Up: 0.5 |

Parameter 'RollOff ( 0.5 )'

Distribution: Uniform

Uniform: Down = 0.5 , Up = 0.5 ,
Min = 0.0 , Max = 1.0

Hard Limits

Min: 0.0

Max: 1.0

Down: 0.5

Up: 0.5

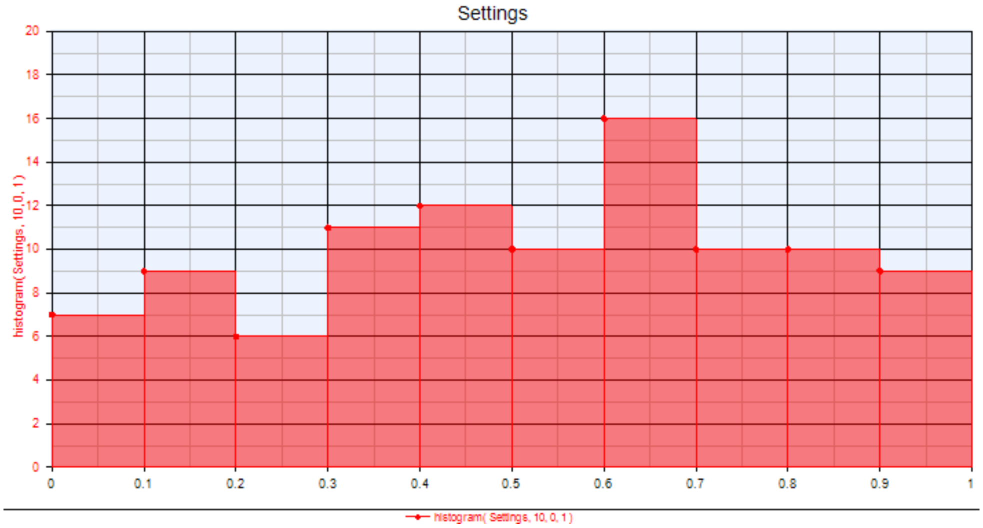
☐ Use Percentages (%)

☑ Sort Alphabetically

☐ Long Variable Names

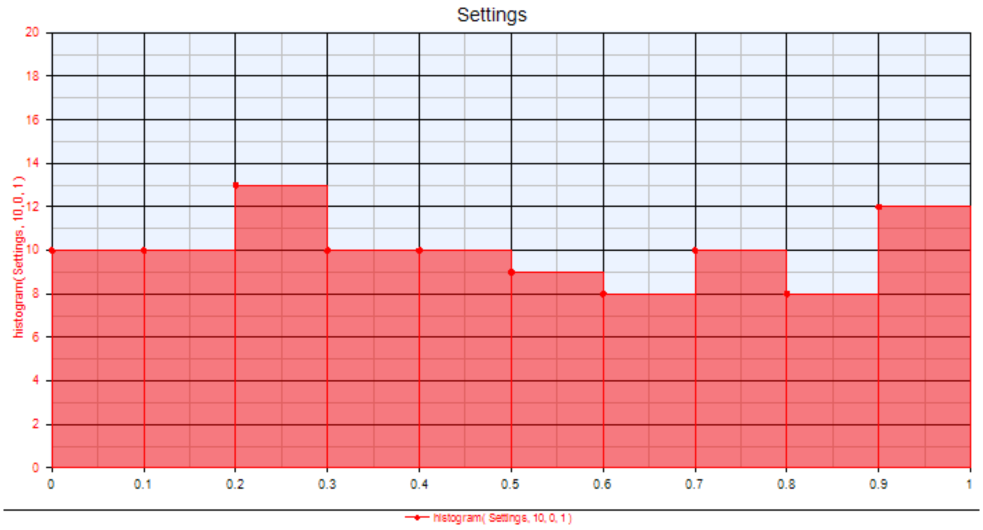
5. Press the OK button to apply the changes you made. Now we are ready to run the Monte Carlo analysis. Right click on *MonteCarlo1* and select Run (calculate now). A dataset *MonteCarlo1_Data* is generated.

6. It is a good idea to check whether we have a good distribution for the perturbed variable *Design1.RollOff*. To do this, open the dataset *MonteCarlo1_Data*, right click on the *Settings* variable and select *Add to Graph > New Graph Series Wizard...*. In the *Graph Series Wizard* window that opens select *Histogram* in the *Type of Series Selected* area and then press the OK button. In the *MonteCarlo1_Settings Properties* window change the equation in the Variable column from histogram( Settings ) to histogram( Settings, 10, 0, 1 ). In the *Y-Axis* tab, uncheck the *Auto-Scale* checkbox and set the *Min* and *Max* values to 0 and 20 respectively. Finally, press the OK button. A histogram of the perturbed variable *Design1.RollOff* is created.

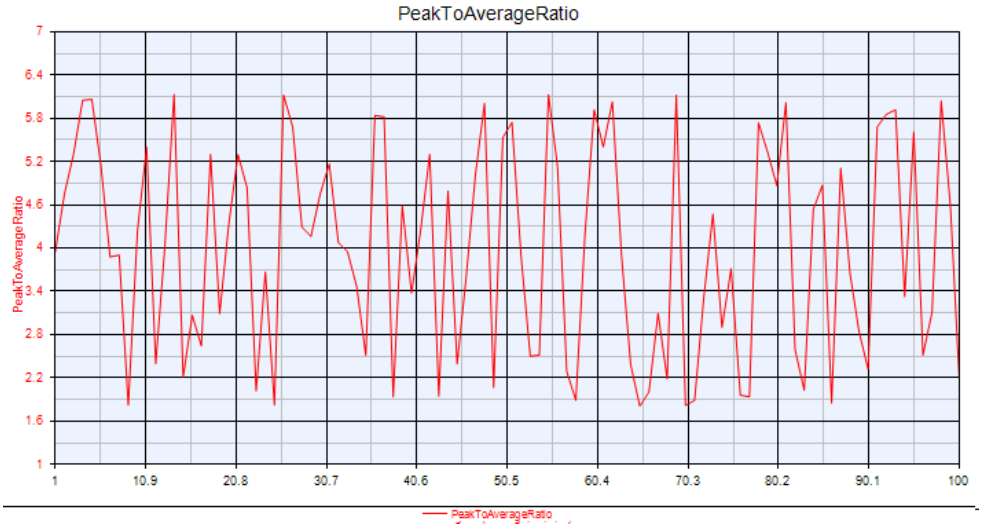


As you can see, the *Random Seed* value of 0 (in the *General* tab of *MonteCarlo1*) does not generate a good distribution for 100 runs (see *Number of Samples* field in the *General tab of _MonteCarlo1*).

7. Try a few different *Random Seed* values until you get a good distribution. The higher the number of runs (*Number of Samples*) the more likely you will get a good distribution for an arbitrary seed value. The *Random Seed* 4351, seems to give a reasonably good uniform distribution.

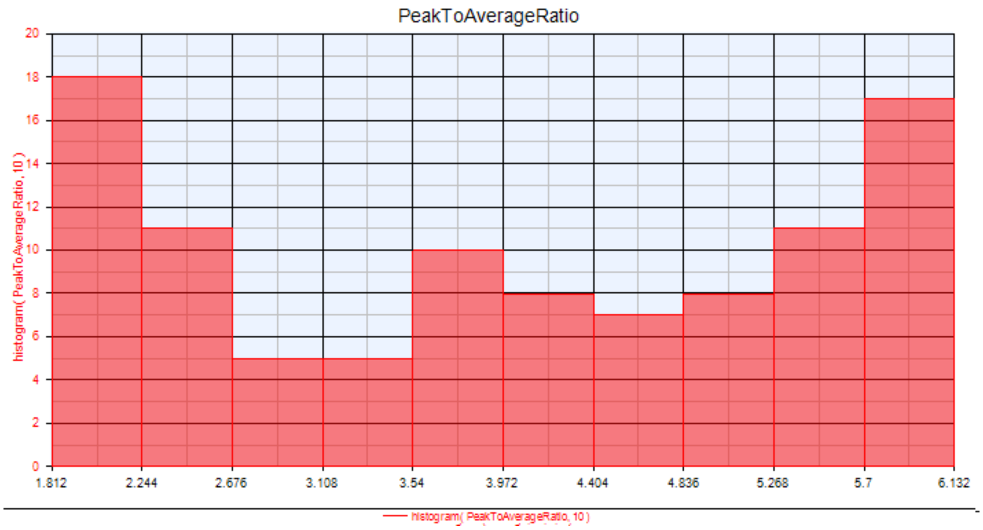


8. Now we are ready to look at the results. Open the *MonteCarlo1_Data* dataset, right-click on the *PeakToAverageRatio* variable and select *Add to Graph > New Graph*. This creates a plot of the *PeakToAverageRatio* value versus the index of the 100 runs we performed.



9. You can examine this graph for any extreme values, see which run produced them (move the mouse over the point in the graph and the pop up will show the independent and dependent values; the independent value is the run index), and go back in the dataset to find out what the value of the perturbed variable (*Settings* variable) was for that run.

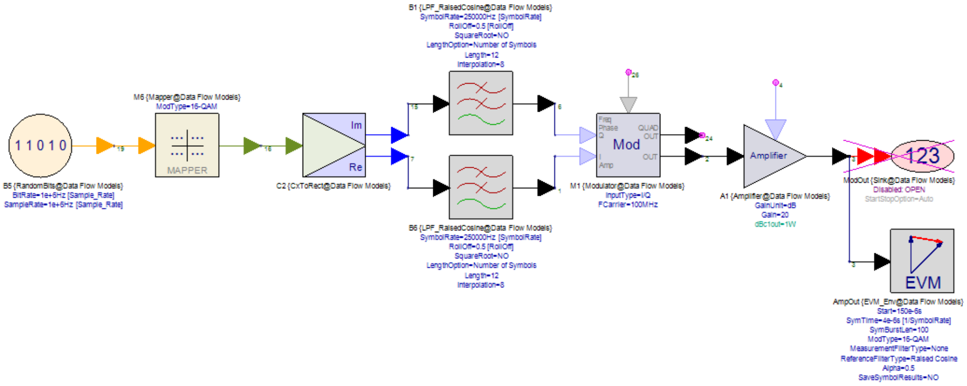
10. Now let's create a histogram of the *PeakToAverageRatio* values. Double-click the *PeakToAverageRatio* graph and change the equation in the Variable column from PeakToAverageRatio to histogram( PeakToAverageRatio, 10 ). In the *Y-Axis* tab, uncheck the *Auto-Scale* checkbox and set the *Min* and *Max* values to 0 and 20 respectively. Finally, press the OK button. The histogram of *PeakToAverageRation* values is created.



11. By examining this histogram you can identify sensitive areas in your design, that is, areas where small variations in the design variable *Desing1.RollOff* can cause large changes in the systems behavior (*PeakToAverageRatio* measurement).

Next we will set up a Monte Carlo analysis for an EVM simulation.

1. Open *Design2* in folder *2. EVM*. This design generates a 16-QAM signal with 0.25 MHz symbol rate sampled at 2 MHz.

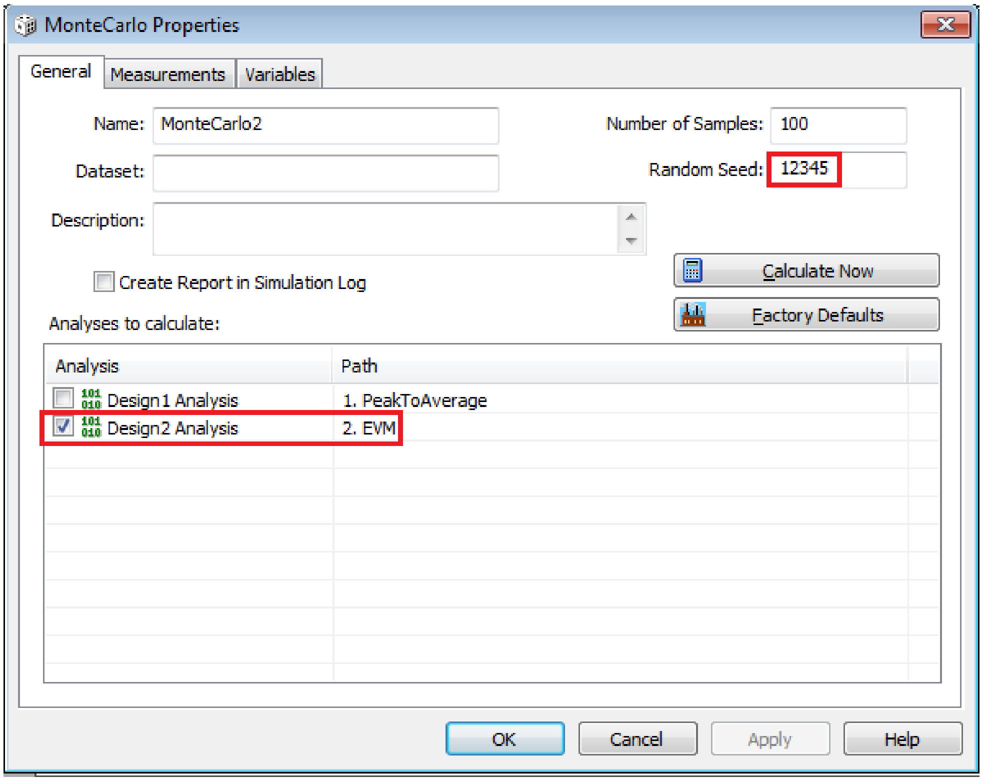

B1 {LPF_RaisedCosine@Data Flow Models}
SymbolRate=250000Hz [SymbolRate]
RollOff=0.5 [RollOff]
SquareRoot=NO
LengthOption=Number of Symbols
Length=12
Interpolation=8
M6 {Mapper@Data Flow Models}
ModType=16-QAM
1 1 0 1 0
MAPPER
Im
Re
B5 {RandomBits@Data Flow Models}
BitRate=1e+6Hz [Sample_Rate]
SampleRate=1e+6Hz [Sample_Rate]
C2 {CxToRect@Data Flow Models}
B6 {LPF_RaisedCosine@Data Flow Models}
SymbolRate=250000Hz [SymbolRate]
RollOff=0.5 [RollOff]
SquareRoot=NO
LengthOption=Number of Symbols
Length=12
Interpolation=8
Freq
Phase
Q
I
Amp
Mod
QUAD OUT
OUT
M1 {Modulator@Data Flow Models}
InputType=I/Q
FCarrier=100MHz
Amplifier
A1 {Amplifier@Data Flow Models}
GainUnit=dB
Gain=20
dBc1out=1W
123
ModOut {Sink@Data Flow Models}
Disabled: OPEN
StartStopOption=Auto
EVM
AmpOut {EVM_Env@Data Flow Models}
Start=150e-6s
SymTime=4e-6s [1/SymbolRate]
SymBurstLen=100
ModType=16-QAM
MeasurementFilterType=None
ReferenceFilterType=Raised Cosine
Alpha=0.5
SaveSymbolResults=NO

2. Run *Design2 Analysis* and observe the *AmpOut_EVM_RMS* variable in the *Design2_Data* dataset. The current system settings result in an EVM of 3.278%.

3. Now let's set up a Monte Carlo simulation to see how EVM varies when we vary the amplifiers *dBc1out* parameter (based on a uniform distribution from 0.5 W to 2 W).

4. Right-click on the folder *2. EVM* and select *Add* > *Evaluations* > *Add Monte Carlo Analysis...*.

   In the *General* tab, select *Design2 Analysis* in the *Analyses to calculate* area and set *Random Seed* to 12345.



Name: MonteCarlo2

Number of Samples: 100

Dataset:

Random Seed: 12345

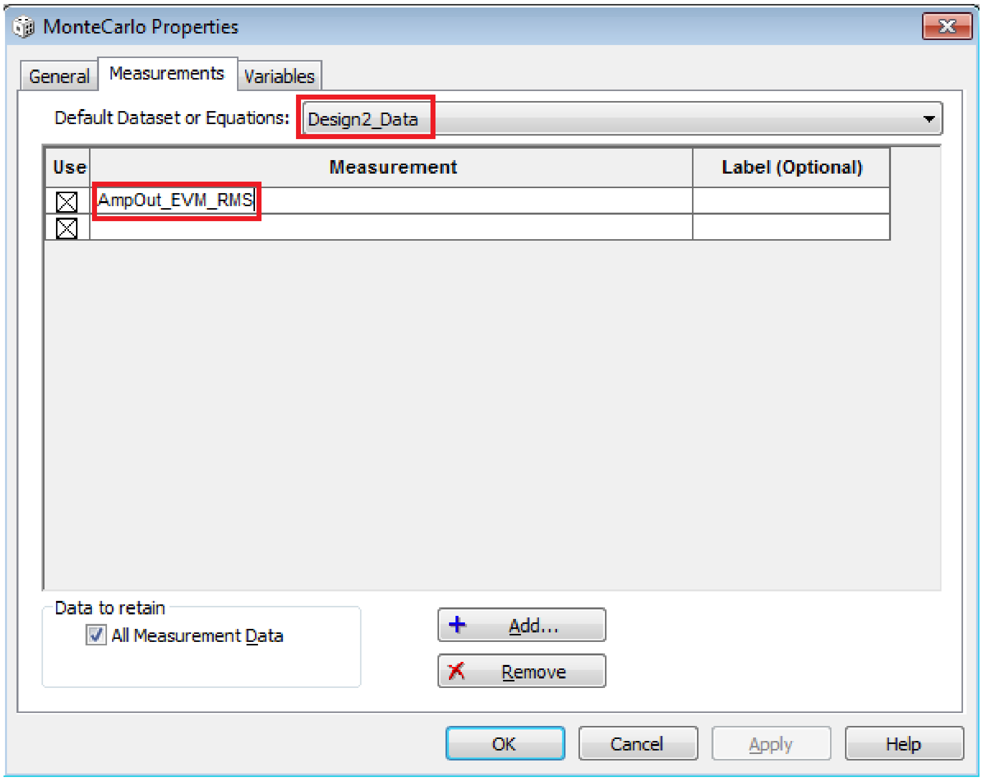
Description:

□ Create Report in Simulation Log

Analyses to calculate:

| Analysis | Path |
|---|---|
| □ Design1 Analysis | 1. PeakToAverage |
| ☑ Design2 Analysis | 2. EVM |

In the *Measurements* tab, set *Default Dataset or Equations* to *Design2_Data* and in the grid below add the *AmpOut_EVM_RMS* measurement.



Default Dataset or Equations: Design2_Data

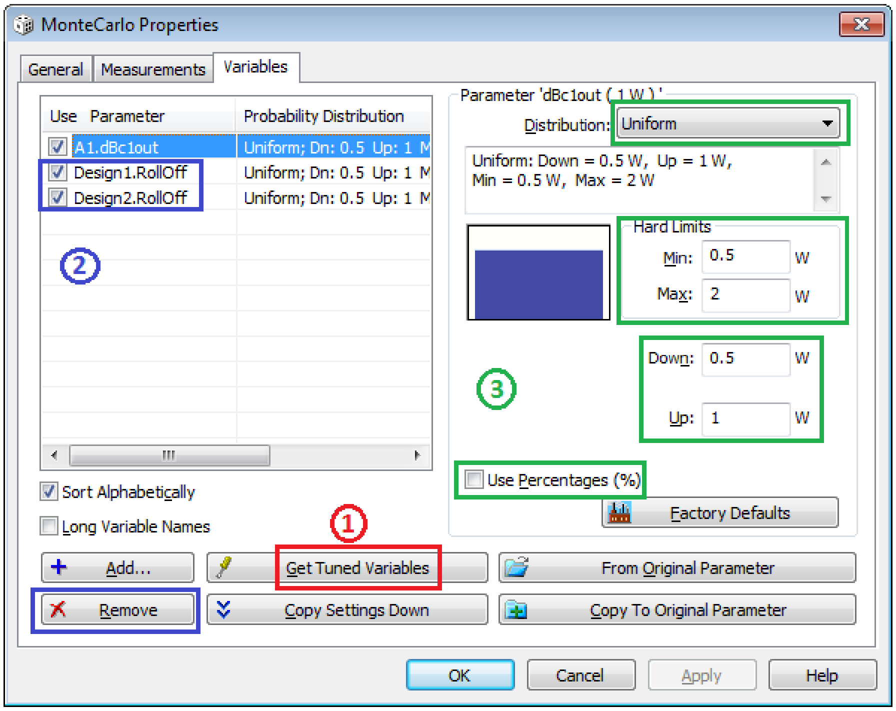
| Use | Measurement | Label (Optional) |
|---|---|---|
| ☒ | AmpOut_EVM_RMS | |
| ☒ | | |

Data to retain

☑ All Measurement Data

In the *Variables* tab

- Press the Get Tuned Variables button. This will populate the grid in the top left area with all tuned variables (tuned variables are part parameters that have their Tune checkbox checked or variables in Equations pages that are assigned using the =? operator; see Equations tab of *Design2*).
- Select each of the rows with *Design1.RollOff* and *Design2.RollOff* and press the Remove button to remove them.
- Select the remaining row (where *A1.dBc1out* is defined) and set its distribution (top right corner) to Uniform. Uncheck the Use Percentages (%) checkbox. Now define by how much the variable will be perturbed (down and up) from its nominal value (shown in the top right area just above the Distribution drop-down menu). For our example the nominal value of *A1.dBc1out* is 1 so in order to perturb it based on a uniform distribution between 0.5 and 2, we need to set the *Down* field to 0.5 and the *Up* field to 1. We can also set the *Min* and *Max Hard Limits* (this applies more to distributions that can generate arbitrarily big magnitude values like the *Normal*, *Beta*, *Lognormal*).



MonteCarlo Properties

General | Measurements | Variables

| Use | Parameter | Probability Distribution |
|---|---|---|
| ☑ | A1.dBc1out | Uniform; Dn: 0.5 Up: 1 M |
| ☑ | Design1.RollOff | Uniform; Dn: 0.5 Up: 1 M |
| ☑ | Design2.RollOff | Uniform; Dn: 0.5 Up: 1 M |

☑ Sort Alphabetically

□ Long Variable Names

Parameter 'dBc1out ( 1 W )'

Distribution: Uniform

Uniform: Down = 0.5 W, Up = 1 W,
Min = 0.5 W, Max = 2 W

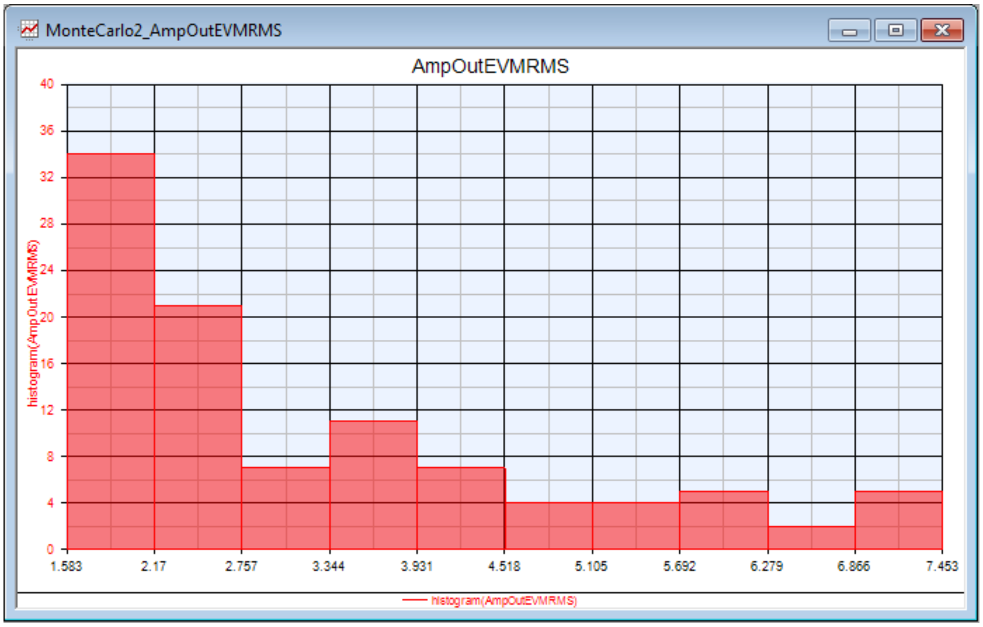
Hard Limits
Min: 0.5 W
Max: 2 W

Down: 0.5 W
Up: 1 W

□ Use Percentages (%)

5. Press the OK button to apply the changes you made and run the Monte Carlo analysis (right-click on it and select *Run (calculate now)*).

6. When the analysis completes, open the *MonteCarlo2_Data* dataset, right-click on the *AmpOut_EVM_RMS* variable and select *Add to Graph > New Graph Series Wizard...*. In the *Graph Series Wizard* window that opens select *Histogram* in the *Type of Series Selected* area and then press the OK button. In the *MonteCarlo2_AmpOut_EVM_RMS Properties* window go to the *Y-Axis* tab, uncheck the *Auto-Scale* checkbox and set the *Min* and *Max* values to 0 and 40 respectively. Finally, press the OK button. A histogram of the *AmpOut_EVM_RMS* values is created.



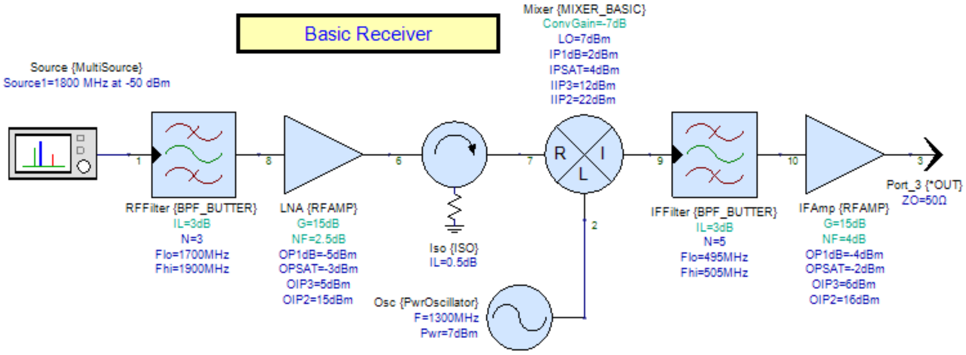
7. By examining this histogram you can identify sensitive areas in your design, that is, areas where small variations in the design variable *dBc1out* can cause large changes in the systems behavior (*AmpOut_EVM_RMS* measurement).

## Monte Carlo Example (Receiver Monte Carlo.wsv)

This example illustrates how a Monte Carlo evaluation is coupled with parameters statistics to determine the overall system performance of a basic receiver. The system parameters that will be examined are Cascaded Gain and Cascaded Noise Figure.

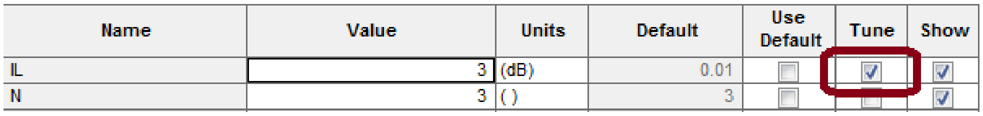
**NOTE** For this tutorial example, click Help / Open Example in SystemVue and open the Tutorials\Simulation_Control_and_Scripting\Receiver Monte Carlo.wsv workspace.

1. Create a schematic.



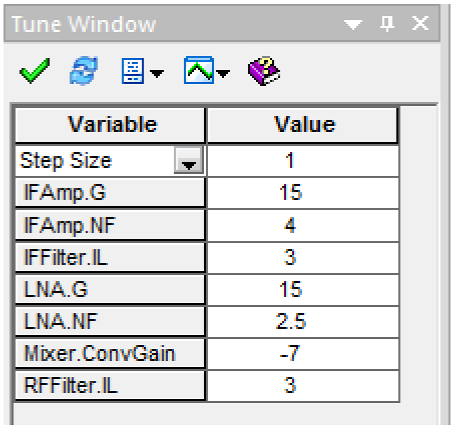
2. Create a System Analysis with a Path
The Monte Carlo analysis controls another analysis. In this case, it is a system analysis. Since the path measurements will be used in the Monte Carlo analysis a path must be added also. See Getting Started with Spectrasys for additional information on adding a system analysis and a path.

3. Create variables that the Monte Carlo analysis will modify.
Tuned variables are needed for the Monte Carlo analysis. A tuned variable can be created by checking the Tune check-box for the given part parameter.



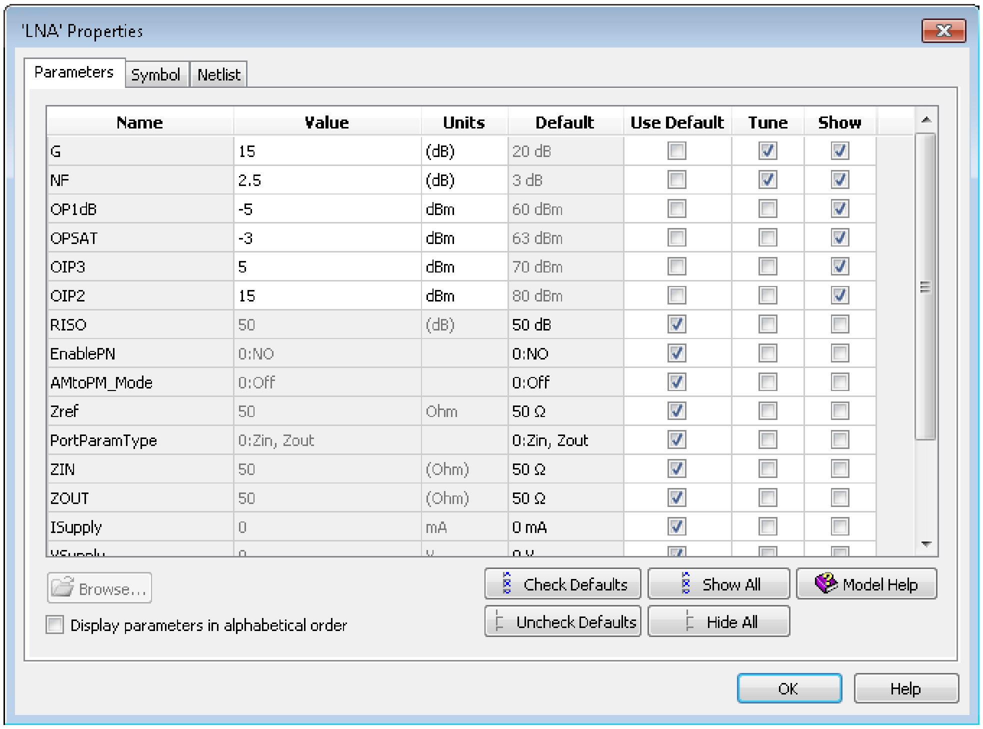
| Name | Value | Units | Default | Use Default | Tune | Show |
|---|---|---|---|---|---|---|
| IL | 3 | (dB) | 0.01 | ☐ | ☑ | ☑ |
| N | 3 | ( ) | 3 | ☐ | ☐ | ☑ |

In the schematic shown above the tuned variables are shown in teal .
A summary of all tuned variables is shown in the tune window.



| Variable | Value |
|---|---|
| Step Size | 1 |
| IFAmp.G | 15 |
| IFAmp.NF | 4 |
| IFFilter.IL | 3 |
| LNA.G | 15 |
| LNA.NF | 2.5 |
| Mixer.ConvGain | -7 |
| RFFilter.IL | 3 |

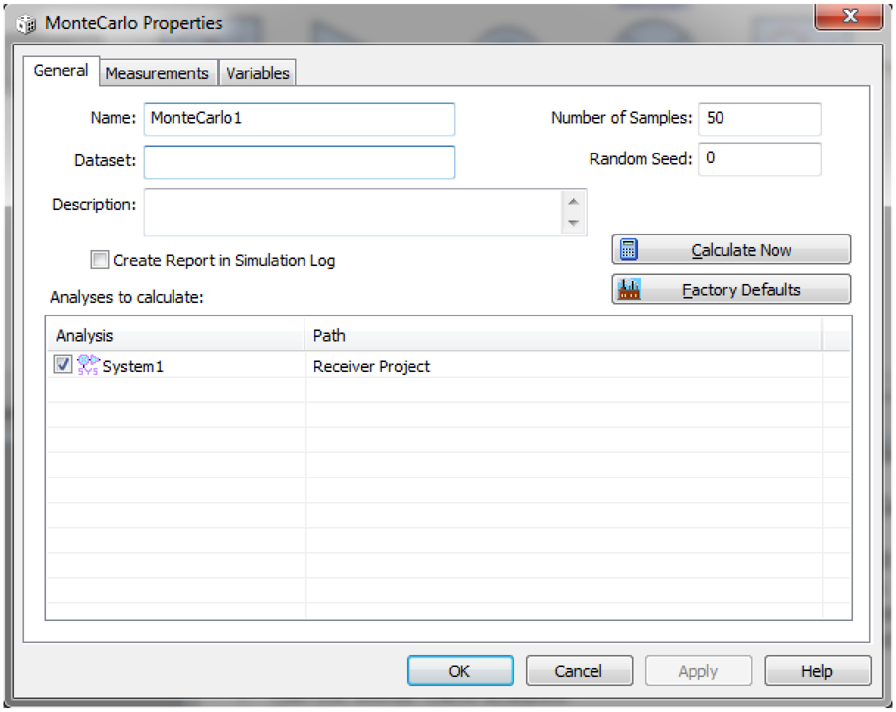
4. Assign statistical limits and distributions to variables as desired.
   Each tuned variable can have a particular statistical distribution assigned to it. See Parameter Statistics for additional information.
   The Parameters tab can be accessed by clicking the 'Advanced Options...' button ( Advanced Options... ) on the Part Parameters page. The following image depicts the statistical configuration for the LNA.



| Name | Value | Units | Default | Use Default | Tune | Show |
|---|---|---|---|---|---|---|
| G | 15 | (dB) | 20 dB | □ | ☑ | ☑ |
| NF | 2.5 | (dB) | 3 dB | □ | ☑ | ☑ |
| OP1dB | -5 | dBm | 60 dBm | □ | □ | ☑ |
| OPSAT | -3 | dBm | 63 dBm | □ | □ | ☑ |
| OIP3 | 5 | dBm | 70 dBm | □ | □ | ☑ |
| OIP2 | 15 | dBm | 80 dBm | □ | □ | ☑ |
| RISO | 50 | (dB) | 50 dB | ☑ | □ | □ |
| EnablePN | 0:NO |  | 0:NO | ☑ | □ | □ |
| AMtoPM_Mode | 0:Off |  | 0:Off | ☑ | □ | □ |
| Zref | 50 | Ohm | 50 Ω | ☑ | □ | □ |
| PortParamType | 0:Zin, Zout |  | 0:Zin, Zout | ☑ | □ | □ |
| ZIN | 50 | (Ohm) | 50 Ω | ☑ | □ | □ |
| ZOUT | 50 | (Ohm) | 50 Ω | ☑ | □ | □ |
| ISupply | 0 | mA | 0 mA | ☑ | □ | □ |
| VSupply | 0 | V | 0 V | ☑ | □ | □ |

□ Display parameters in alphabetical order

5. Create a Monte Carlo analysis.
   - Set the general information



Name: MonteCarlo1

Dataset:

Description:

Number of Samples: 50

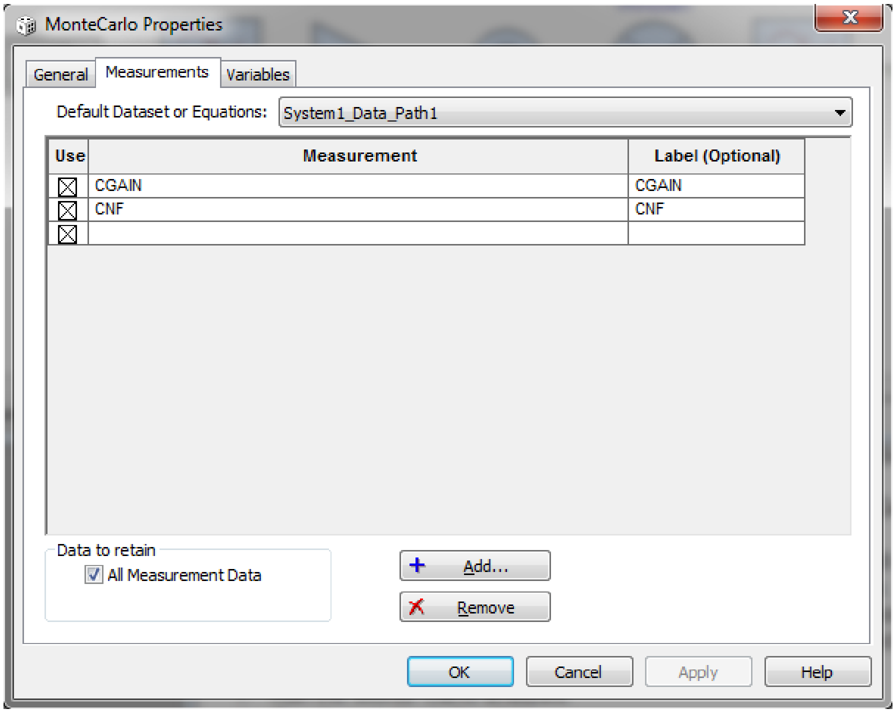
Random Seed: 0

☐ Create Report in Simulation Log

Analyses to calculate:

| Analysis | Path |
|---|---|
| ☑ System1 | Receiver Project |

- Add measurements



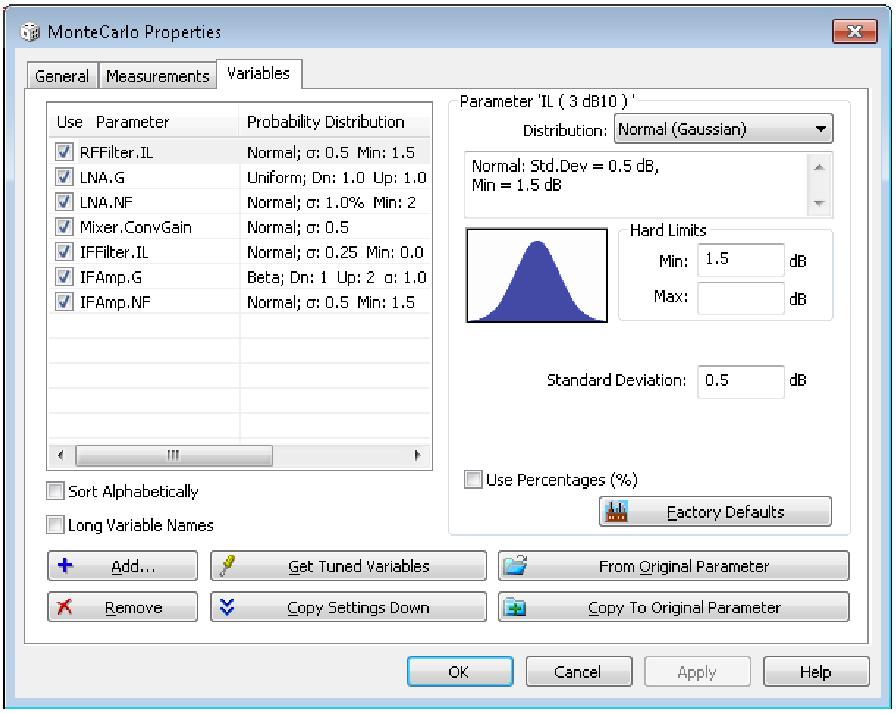
Default Dataset or Equations: System1_Data_Path1

| Use | Measurement | Label (Optional) |
|---|---|---|
| ☒ | CGAIN | CGAIN |
| ☒ | CNF | CNF |
| ☒ | | |

Data to retain

☑ All Measurement Data

Select the variables



| Use | Parameter | Probability Distribution |
|---|---|---|
| ☑ | RFFilter.IL | Normal; σ: 0.5 Min: 1.5 |
| ☑ | LNA.G | Uniform; Dn: 1.0 Up: 1.0 |
| ☑ | LNA.NF | Normal; σ: 1.0% Min: 2 |
| ☑ | Mixer.ConvGain | Normal; σ: 0.5 |
| ☑ | IFFilter.IL | Normal; σ: 0.25 Min: 0.0 |
| ☑ | IFAmp.G | Beta; Dn: 1 Up: 2 α: 1.0 |
| ☑ | IFAmp.NF | Normal; σ: 0.5 Min: 1.5 |

Parameter 'IL ( 3 dB10 ) '

Distribution: Normal (Gaussian)

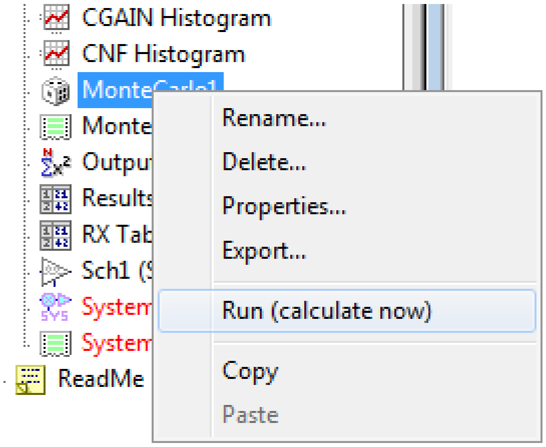
Normal: Std.Dev = 0.5 dB,
Min = 1.5 dB

Hard Limits
Min: 1.5 dB
Max: dB

Standard Deviation: 0.5 dB

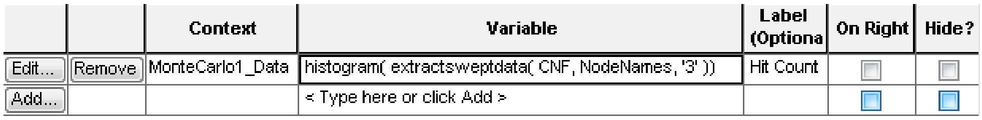
□ Use Percentages (%)

6. Run the Monte Carlo analysis



7. Plotting the results

The histogram function is used to plot the results.
When plotting the results of a system or Spectrasys path measurement an additional function is needed to extract Monte Carlo data at a particular node along the path. The extractsweptdata function is used for that purpose. After creating a new graph the measurement can be entered as follows. This measurement shows that a histogram will be plotted for the Cascaded Noise figure measurement for the node named '3'.

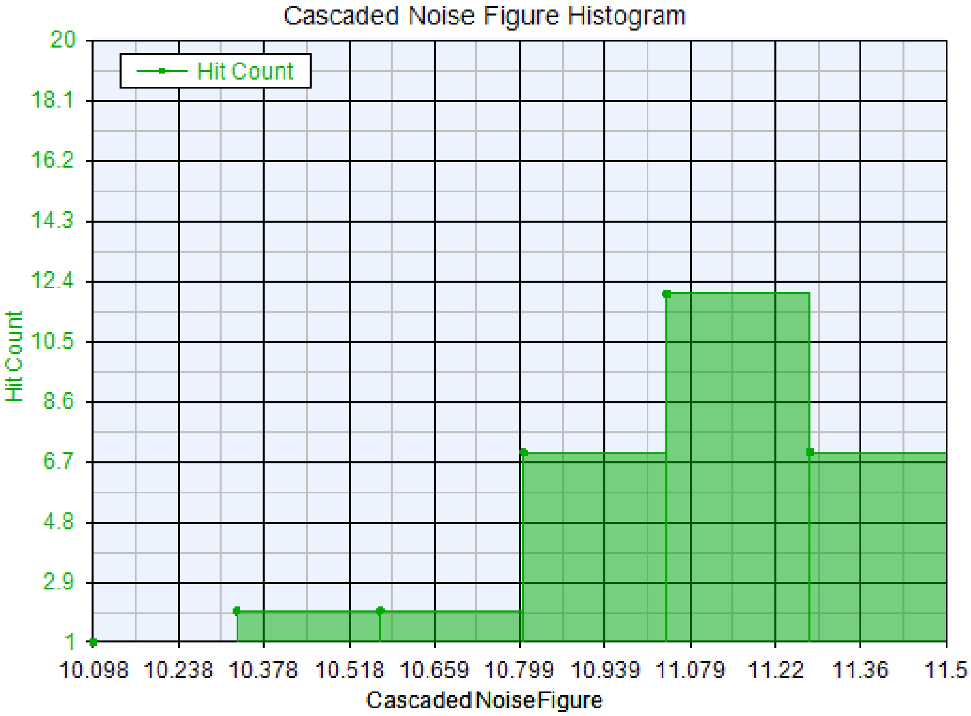


| | | Context | Variable | Label (Optiona | On Right | Hide? |
|---|---|---|---|---|---|---|
| Edit... | Remove | MonteCarlo1_Data | histogram( extractsweptdata( CNF, NodeNames, '3' )) | Hit Count | □ | □ |
| Add... | | | < Type here or click Add > | | □ | □ |

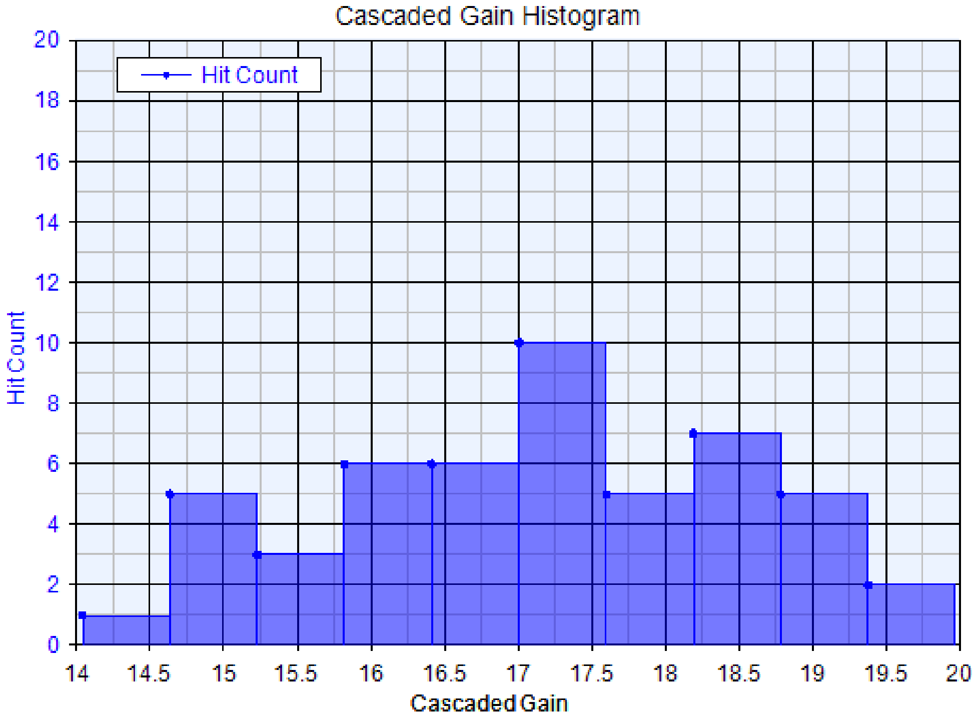
NOTE

Port numbers are automatically generated based on the order in which the ports (or sources) are placed in the schematic. In the schematic for this example, the LO is port 2 and the output is port 3. If the order of the parts on your schematic is different from this, you will need to select the correct node name in the measurement to display same results.

This is the histogram of the cascaded noise figure.



The setup for the cascaded gain graph is identical of that to the cascaded noise figure except the measurement of CGAIN (cascaded gain) is used instead of CNF (cascaded noise figure).



## Running a Yield Analysis on a Design

### Running a Yield Evaluation on a Design

This section has a series of tutorials that demonstrate how to run Yield evaluation on designs created in Spectrasys or Data Flow.

#### Yield for Data Flow Designs

Data Flow Yield Example

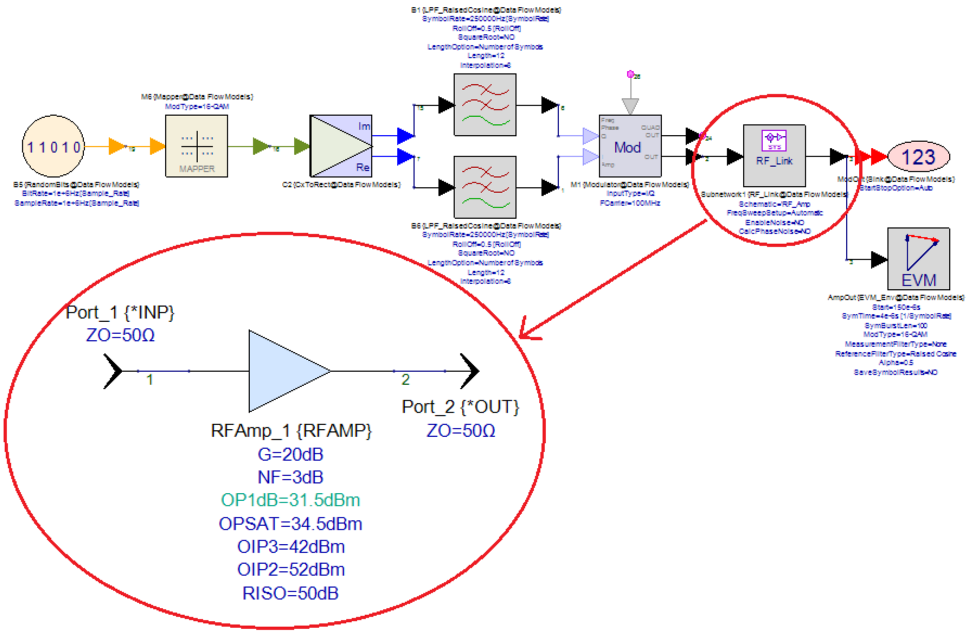
#### Yield for Spectrasys Designs

Spectrasys Yield Example

### Data Flow Yield Example

This tutorial example shows how to perform a Yield evaluation on a Data Flow design.

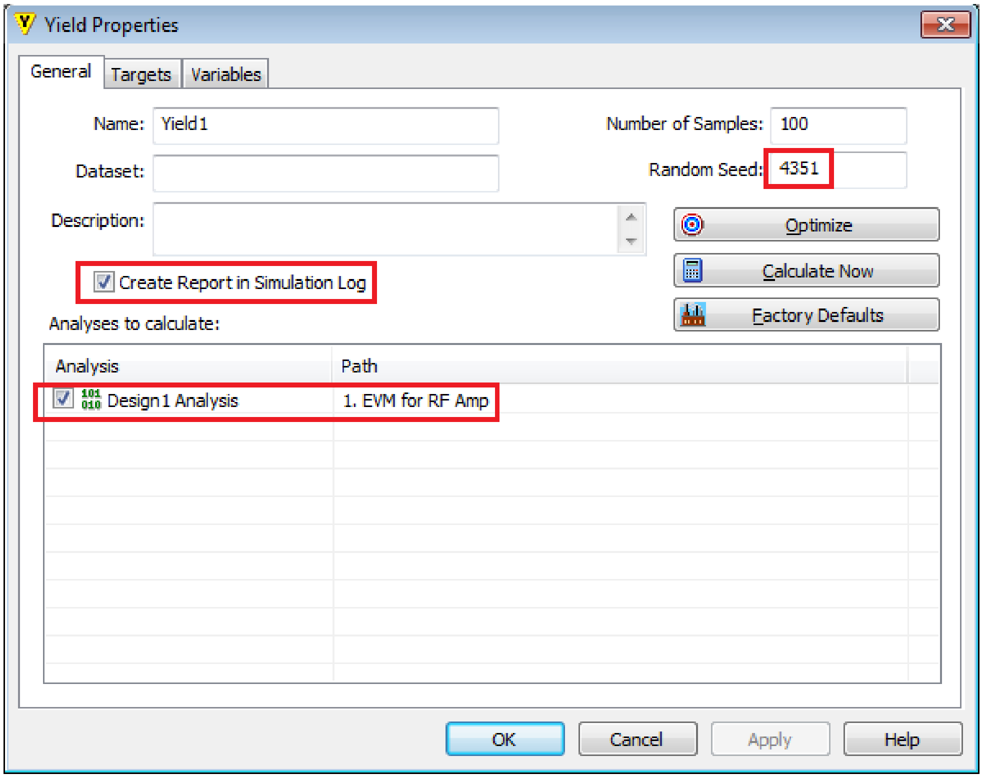
1. Open the example <SystemVue Installation Directory>\Examples\Tutorials\Simulation_Control_and_Scripting\YieldForDat wsv.
2. Open *Design1* in the *1. EVM for RF Amp* folder. This design generates a 16-QAM signal with a symbol rate of 0.25 MHz sampled at 2 MHz. The signal is amplified using an amplifier designed in Spectrasys (*RF_Amp*) using RF_Link.


B1 {LPF_RaisedCosine@Data Flow Models}
SymbolRate=250000Hz[SymbolRate]
RollOff=0.5 [RollOff]
SquareRoot=NO
LengthOption=Number of Symbols
Length=12
Interpolation=6
M6 {Mapper@Data Flow Models}
ModType=16-QAM
1 1 0 1 0
MAPPER
Im
Re
Mod
Freq
Phase
Q
I
Amp
QUAD OUT
OUT
RF_Link
123
B5 {RandomBits@Data Flow Models}
BitRate=1e+6Hz[Sample_Rate]
SampleRate=1e+6Hz[Sample_Rate]
C2 {CxToRect@Data Flow Models}
M1 {Modulator@Data Flow Models}
InputType=IQ
FCarrier=100MHz
Subnetwork1 {RF_Link@Data Flow Models}
Schematic=RF_Amp
FreqSweepSetup=Automatic
EnableNoise=NO
CalcPhaseNoise=NO
ModOut {Sink@Data Flow Models}
StartStopOption=Auto
B6 {LPF_RaisedCosine@Data Flow Models}
SymbolRate=250000Hz[SymbolRate]
RollOff=0.5 [RollOff]
SquareRoot=NO
LengthOption=Number of Symbols
Length=12
Interpolation=6
EVM
AmpOut {EVM_Env@Data Flow Models}
Start=150e-6s
SymTime=4e-6s [1/SymbolRate]
SymBurstLen=100
ModType=16-QAM
MeasurementFilterType=None
ReferenceFilterType=Raised Cosine
Alpha=0.5
SaveSymbolResults=NO
Port_1 {*INP}
ZO=50Ω
1
2
Port_2 {*OUT}
ZO=50Ω
RFAmp_1 {RFAMP}
G=20dB
NF=3dB
OP1dB=31.5dBm
OPSAT=34.5dBm
OIP3=42dBm
OIP2=52dBm
RISO=50dB

3. Run *Design1 Analysis*, open the *Design1_Data*dataset, and observe the *AmpOut_EVM_RMS* variable. The current system settings result in an EVM of 2.4%. In this example, we will set up a Yield analysis to find out the percentage of amplifiers that will pass an EVM test of less than 2.8%, when the output 1 dB compression point (*OP1dB*) varies based on a normal distribution with a mean of 31.5 dBm and a standard deviation of 0.5 dBm.

4. Right-click the folder *1. EVM for RF Amp* and select *Add* > *Evaluations* > *Add Yield Analysis...*.

   In the *General* tab, select *Design1 Analysis* in the *Analyses to calculate* area, set *Random Seed* to 4351 (see Monte Carlo Data Flow Example for how to select a good seed), and check the *Create Report in Simulation Log* checkbox.



Name: Yield1

Number of Samples: 100

Dataset:

Random Seed: 4351

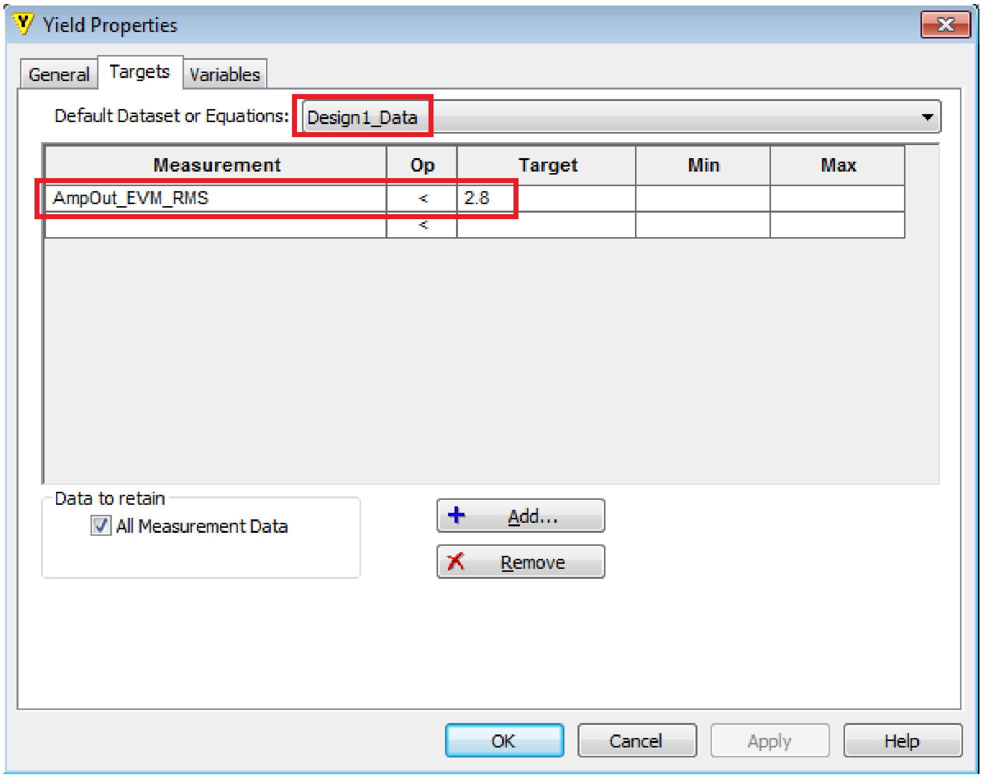
Description:

☑ Create Report in Simulation Log

Analyses to calculate:

| Analysis | Path |
|---|---|
| ☑ Design1 Analysis | 1. EVM for RF Amp |

In the *Targets* tab, set *Default Dataset or Equations* to *Design1_Data* and in the grid below add the *AmpOut_EVM_RMS* measurement with a target of < 2.8.



Default Dataset or Equations: Design1_Data

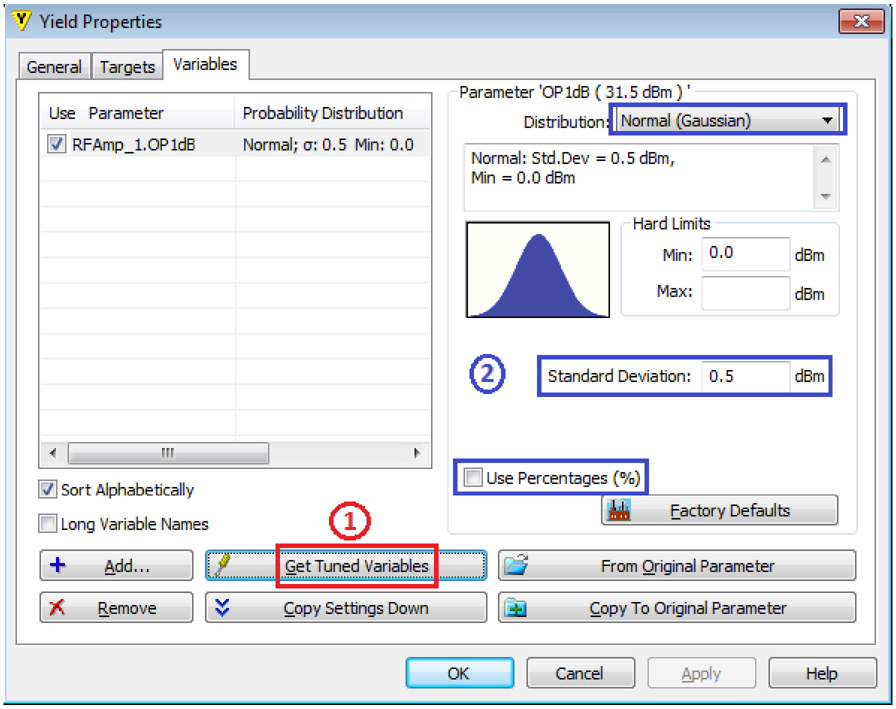
| Measurement | Op | Target | Min | Max |
|---|---|---|---|---|
| AmpOut_EVM_RMS | < | 2.8 | | |
| | < | | | |

Data to retain

☑ All Measurement Data

In the *Variables* tab

- Press the Get Tuned Variables button. This will populate the grid in the top left area with all tuned variables (tuned variables are part parameters that have their Tune checkbox checked or variables in Equations pages that are assigned using the =tune operator).
- Select the first row (where *RFAmp_1.OP1dB* is defined). By default the distribution (top right corner) is set to Normal. Uncheck the Use Percentages (%) checkbox and set the *Standard Deviation* to 0.5 dBm.



Yield Properties

General | Targets | Variables

| Use | Parameter | Probability Distribution |
|---|---|---|
| ☑ | RFAmp_1.OP1dB | Normal; σ: 0.5 Min: 0.0 |

☑ Sort Alphabetically

□ Long Variable Names

Parameter 'OP1dB ( 31.5 dBm ) '

Distribution: Normal (Gaussian)

Normal: Std.Dev = 0.5 dBm,
Min = 0.0 dBm

Hard Limits

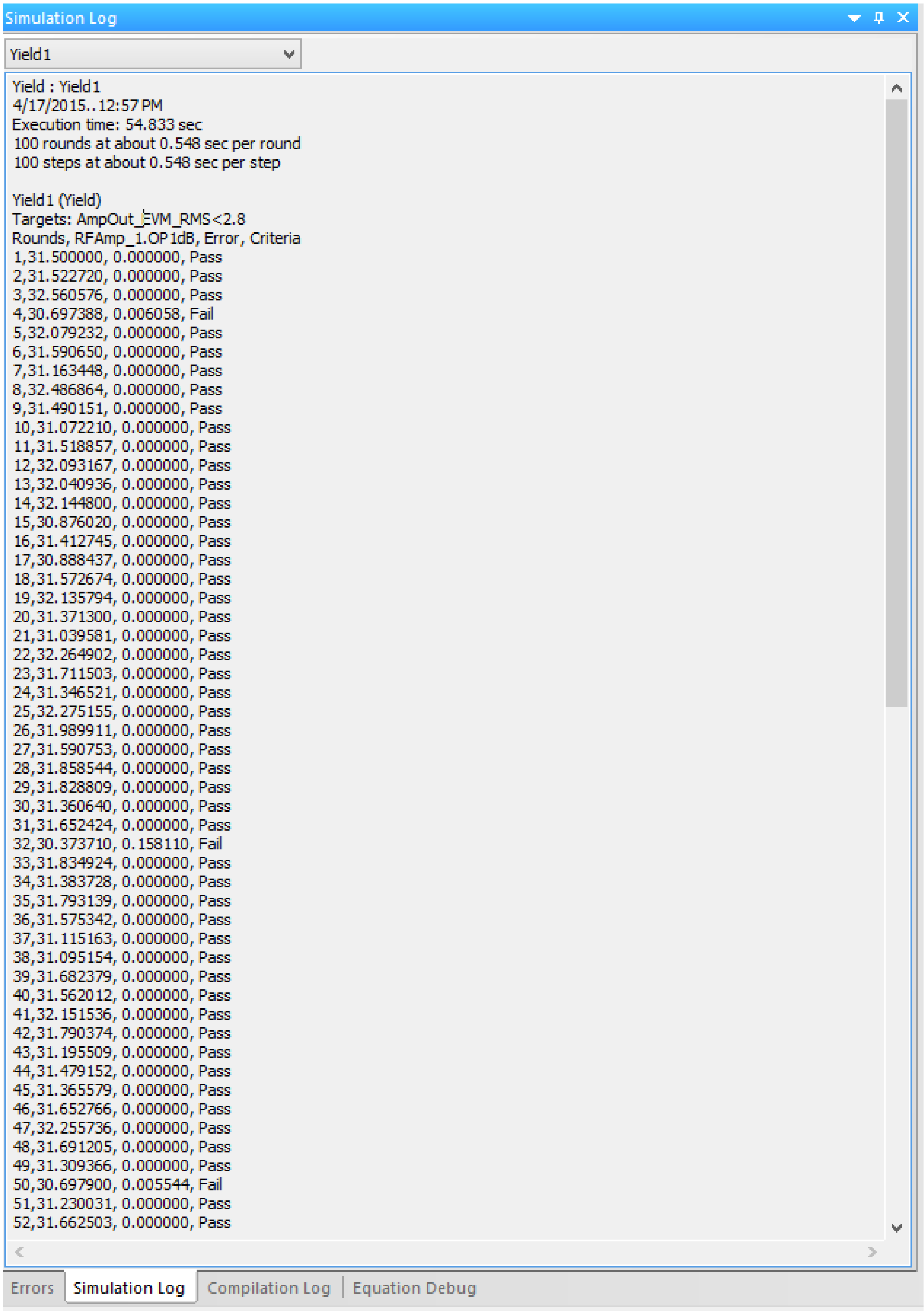
Min: 0.0 dBm

Max: dBm

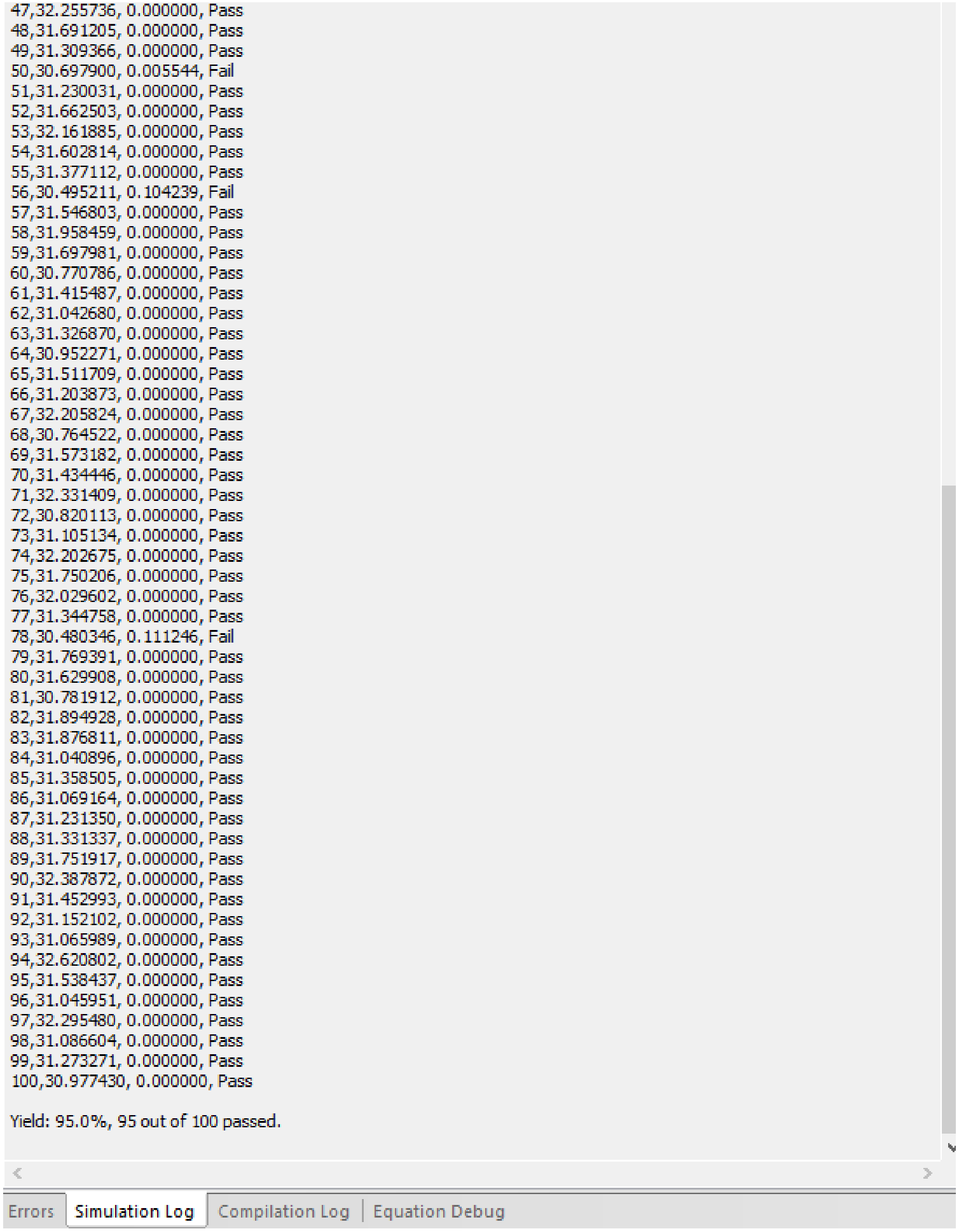
Standard Deviation: 0.5 dBm

□ Use Percentages (%)

5. Press the OK button to apply the changes you made and run the Yield analysis (right-click on it and select *Run (calculate now)*).
6. When the analysis completes, open the *Simulation Log* window (typically located at the bottom of the screen) to see the resulting yield and the full report (if you have checked the *Create Report in Simulation Log* checkbox in the *General* tab of the Yield analysis). For this example, only 95% of the amplifiers would pass the EVM test.



```
Yield : Yield1
4/17/2015..12:57 PM
Execution time: 54.833 sec
 100 rounds at about 0.548 sec per round
 100 steps at about 0.548 sec per step

Yield1 (Yield)
Targets: AmpOut_EVM_RMS<2.8
Rounds, RFAmp_1.OP1dB, Error, Criteria
1,31.500000, 0.000000, Pass
2,31.522720, 0.000000, Pass
3,32.560576, 0.000000, Pass
4,30.697388, 0.006058, Fail
5,32.079232, 0.000000, Pass
6,31.590650, 0.000000, Pass
7,31.163448, 0.000000, Pass
8,32.486864, 0.000000, Pass
9,31.490151, 0.000000, Pass
10,31.072210, 0.000000, Pass
11,31.518857, 0.000000, Pass
12,32.093167, 0.000000, Pass
13,32.040936, 0.000000, Pass
14,32.144800, 0.000000, Pass
15,30.876020, 0.000000, Pass
16,31.412745, 0.000000, Pass
17,30.888437, 0.000000, Pass
18,31.572674, 0.000000, Pass
19,32.135794, 0.000000, Pass
20,31.371300, 0.000000, Pass
21,31.039581, 0.000000, Pass
22,32.264902, 0.000000, Pass
23,31.711503, 0.000000, Pass
24,31.346521, 0.000000, Pass
25,32.275155, 0.000000, Pass
26,31.989911, 0.000000, Pass
27,31.590753, 0.000000, Pass
28,31.858544, 0.000000, Pass
29,31.828809, 0.000000, Pass
30,31.360640, 0.000000, Pass
31,31.652424, 0.000000, Pass
32,30.373710, 0.158110, Fail
33,31.834924, 0.000000, Pass
34,31.383728, 0.000000, Pass
35,31.793139, 0.000000, Pass
36,31.575342, 0.000000, Pass
37,31.115163, 0.000000, Pass
38,31.095154, 0.000000, Pass
39,31.682379, 0.000000, Pass
40,31.562012, 0.000000, Pass
41,32.151536, 0.000000, Pass
42,31.790374, 0.000000, Pass
43,31.195509, 0.000000, Pass
44,31.479152, 0.000000, Pass
45,31.365579, 0.000000, Pass
46,31.652766, 0.000000, Pass
47,32.255736, 0.000000, Pass
48,31.691205, 0.000000, Pass
49,31.309366, 0.000000, Pass
50,30.697900, 0.005544, Fail
51,31.230031, 0.000000, Pass
52,31.662503, 0.000000, Pass
```



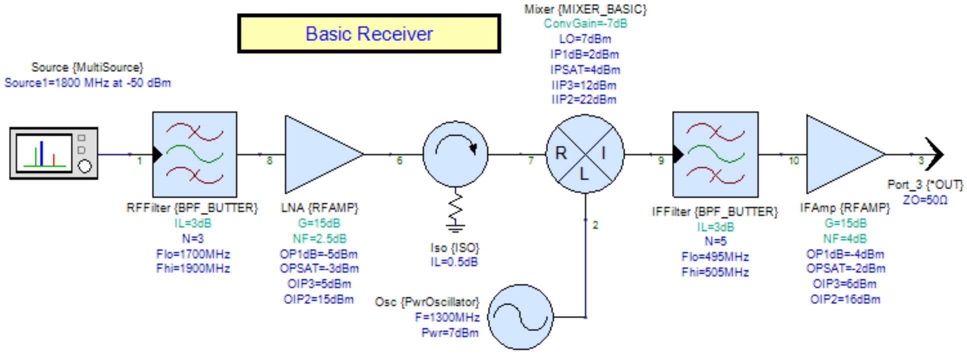
```
47,32.255736, 0.000000, Pass
48,31.691205, 0.000000, Pass
49,31.309366, 0.000000, Pass
50,30.697900, 0.005544, Fail
51,31.230031, 0.000000, Pass
52,31.662503, 0.000000, Pass
53,32.161885, 0.000000, Pass
54,31.602814, 0.000000, Pass
55,31.377112, 0.000000, Pass
56,30.495211, 0.104239, Fail
57,31.546803, 0.000000, Pass
58,31.958459, 0.000000, Pass
59,31.697981, 0.000000, Pass
60,30.770786, 0.000000, Pass
61,31.415487, 0.000000, Pass
62,31.042680, 0.000000, Pass
63,31.326870, 0.000000, Pass
64,30.952271, 0.000000, Pass
65,31.511709, 0.000000, Pass
66,31.203873, 0.000000, Pass
67,32.205824, 0.000000, Pass
68,30.764522, 0.000000, Pass
69,31.573182, 0.000000, Pass
70,31.434446, 0.000000, Pass
71,32.331409, 0.000000, Pass
72,30.820113, 0.000000, Pass
73,31.105134, 0.000000, Pass
74,32.202675, 0.000000, Pass
75,31.750206, 0.000000, Pass
76,32.029602, 0.000000, Pass
77,31.344758, 0.000000, Pass
78,30.480346, 0.111246, Fail
79,31.769391, 0.000000, Pass
80,31.629908, 0.000000, Pass
81,30.781912, 0.000000, Pass
82,31.894928, 0.000000, Pass
83,31.876811, 0.000000, Pass
84,31.040896, 0.000000, Pass
85,31.358505, 0.000000, Pass
86,31.069164, 0.000000, Pass
87,31.231350, 0.000000, Pass
88,31.331337, 0.000000, Pass
89,31.751917, 0.000000, Pass
90,32.387872, 0.000000, Pass
91,31.452993, 0.000000, Pass
92,31.152102, 0.000000, Pass
93,31.065989, 0.000000, Pass
94,32.620802, 0.000000, Pass
95,31.538437, 0.000000, Pass
96,31.045951, 0.000000, Pass
97,32.295480, 0.000000, Pass
98,31.086604, 0.000000, Pass
99,31.273271, 0.000000, Pass
100,30.977430, 0.000000, Pass

Yield: 95.0%, 95 out of 100 passed.
```

### Spectrasys Yield Example

This tutorial example demonstrates how to create and perform a yield evaluation to see how statistical variation of part parameters affects overall system performance of a basic receiver. A yield evaluation performs a Monte Carlo evaluation, but also compares the results of each round (sample) to one or more targets. The results of a Yield show how many rounds passed or failed the yield targets.

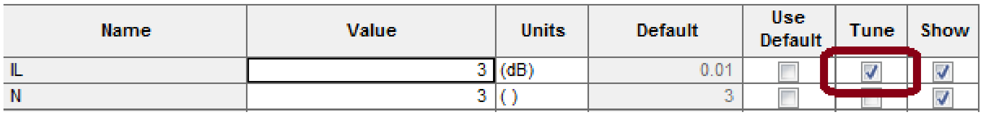
**NOTE** For this tutorial example, click Help / Open Example in SystemVue and open the Tutorials\Simulation_Control_and_Scripting\Receiver Yield.wsv workspace.

1. Create a schematic.



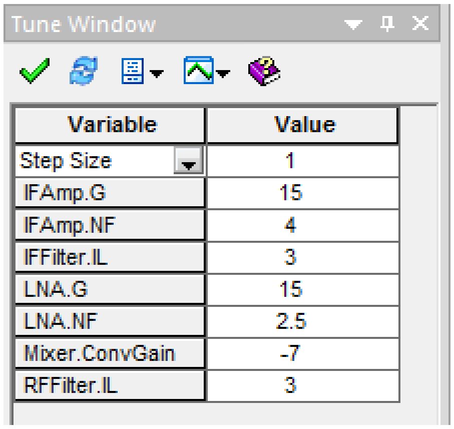
2. Create a System Analysis with a Path
The yield evaluation controls another analysis. In this case, it is a system analysis. Since the path measurements will be used in the Yield evaluation a path must be added also. See Getting Started with Spectrasys for additional information on adding a system analysis and a path.

3. Create variables that the yield evaluation will modify.
Tuned variables are needed for the Yield. A tuned variable can be created by checking the Tune check-box for the given part parameter.



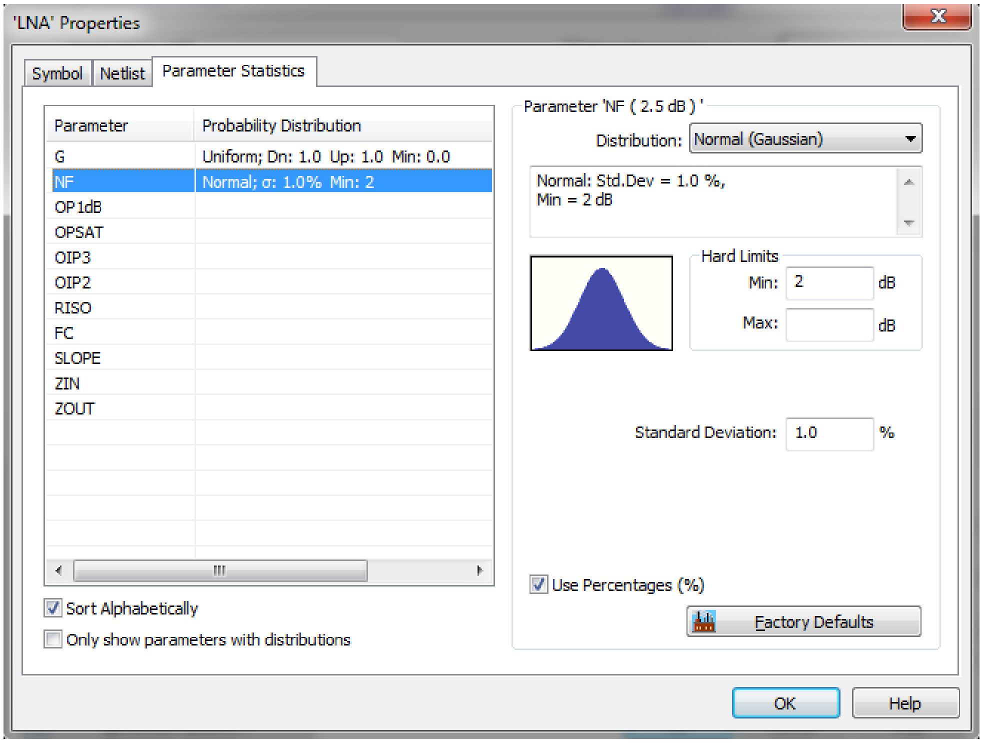
| Name | Value | Units | Default | Use Default | Tune | Show |
|---|---|---|---|---|---|---|
| IL | 3 | (dB) | 0.01 | □ | ☑ | ☑ |
| N | 3 | ( ) | 3 | □ | □ | ☑ |

In the schematic shown above the tuned variables are shown in teal .
A summary of all tuned variables is shown in the tune window.



| Variable | Value |
|---|---|
| Step Size | 1 |
| IFAmp.G | 15 |
| IFAmp.NF | 4 |
| IFFilter.IL | 3 |
| LNA.G | 15 |
| LNA.NF | 2.5 |
| Mixer.ConvGain | -7 |
| RFFilter.IL | 3 |

4. Assign statistical limits and distributions to variables as desired.
Each tuned variable can have a particular statistical distribution assigned to it. See Parameter Statistics for additional information.
The Parameter Statistics Tab can be accessed by clicking the 'Advanced Options...' button ( Advanced Options... ) on the Part Parameters page. The following image depicts the statistical configuration for the LNA.


'LNA' Properties

Symbol | Netlist | Parameter Statistics

| Parameter | Probability Distribution |
|---|---|
| G | Uniform; Dn: 1.0 Up: 1.0 Min: 0.0 |
| NF | Normal; σ: 1.0% Min: 2 |
| OP1dB | |
| OPSAT | |
| OIP3 | |
| OIP2 | |
| RISO | |
| FC | |
| SLOPE | |
| ZIN | |
| ZOUT | |

☑ Sort Alphabetically

□ Only show parameters with distributions

Parameter 'NF ( 2.5 dB ) '

Distribution: Normal (Gaussian)

Normal: Std.Dev = 1.0 %,
Min = 2 dB

Hard Limits

Min: 2 dB

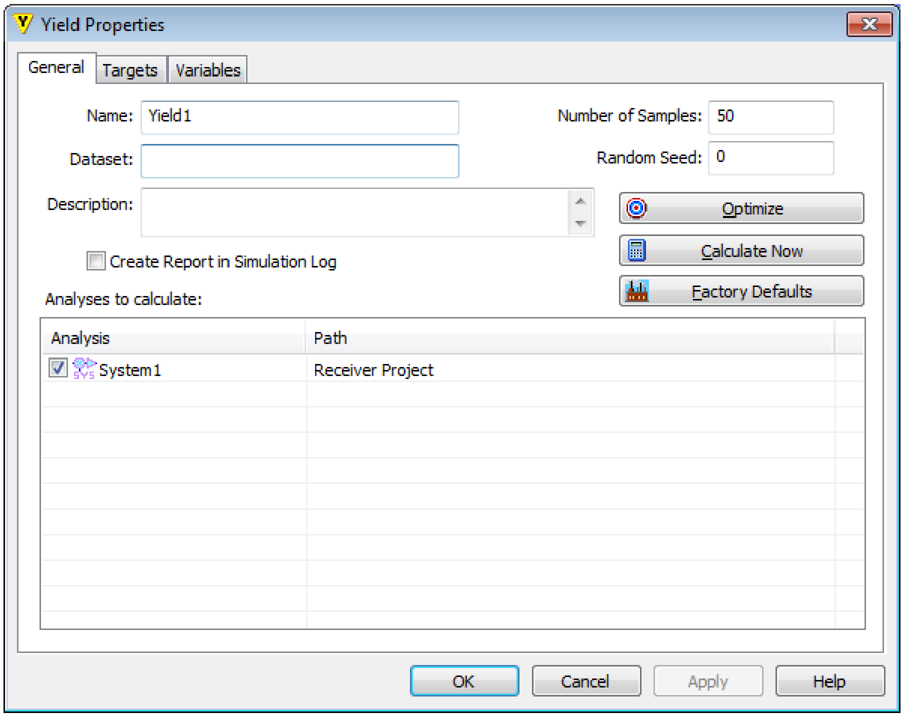
Max: dB

Standard Deviation: 1.0 %

☑ Use Percentages (%)

5. Create a Yield Evaluation.

   a. Set the general information.


Yield Properties

General | Targets | Variables

Name: Yield1

Number of Samples: 50

Dataset:

Random Seed: 0

Description:

□ Create Report in Simulation Log

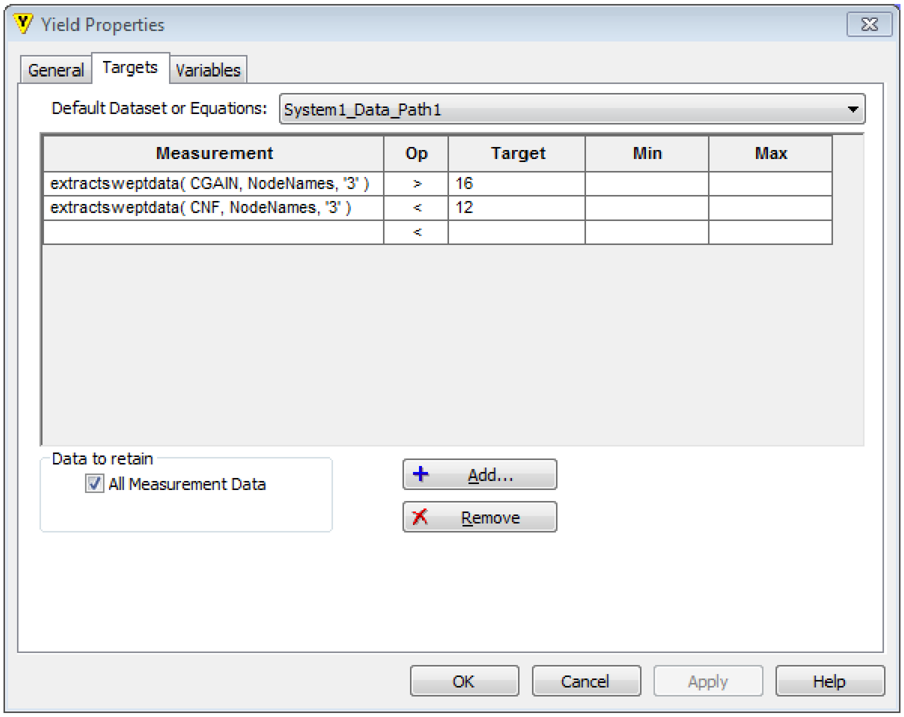
Analyses to calculate:

| Analysis | Path |
|---|---|
| ☑ System1 | Receiver Project |

   b. Add targets.
   When creating a Yield target for a Spectrasys path measurement an additional function is needed to extract a value at a particular node

along the path. The extractsweptdata function is used for that purpose. For this example, we are using the path measurements CGAIN and CNF at the output (node 3) for our yield target measurements.

**NOTE** Port numbers are automatically generated based on the order in which the ports (or sources) are placed in the schematic. In the schematic for this example, the LO is port 2 and the output is port 3. If the order of the parts on your schematic is different than this you will need to select the correct node name in the measurement to display same results.



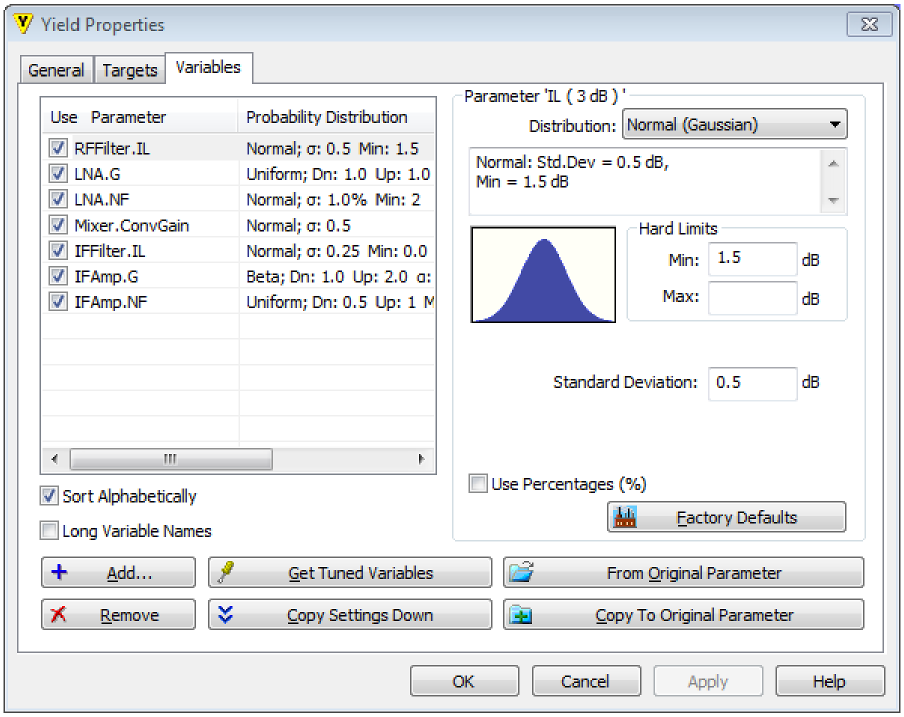
Default Dataset or Equations: System1_Data_Path1

| Measurement | Op | Target | Min | Max |
|---|---|---|---|---|
| extractsweptdata( CGAIN, NodeNames, '3' ) | > | 16 | | |
| extractsweptdata( CNF, NodeNames, '3' ) | < | 12 | | |
| | < | | | |

Data to retain

☑ All Measurement Data

c. Select the variables.



| Use | Parameter | Probability Distribution |
|---|---|---|
| ☑ | RFFilter.IL | Normal; σ: 0.5 Min: 1.5 |
| ☑ | LNA.G | Uniform; Dn: 1.0 Up: 1.0 |
| ☑ | LNA.NF | Normal; σ: 1.0% Min: 2 |
| ☑ | Mixer.ConvGain | Normal; σ: 0.5 |
| ☑ | IFFilter.IL | Normal; σ: 0.25 Min: 0.0 |
| ☑ | IFAmp.G | Beta; Dn: 1.0 Up: 2.0 α: |
| ☑ | IFAmp.NF | Uniform; Dn: 0.5 Up: 1 M |

☑ Sort Alphabetically

□ Long Variable Names

Parameter 'IL ( 3 dB )'

Distribution: Normal (Gaussian)

Normal: Std.Dev = 0.5 dB,
Min = 1.5 dB

Hard Limits

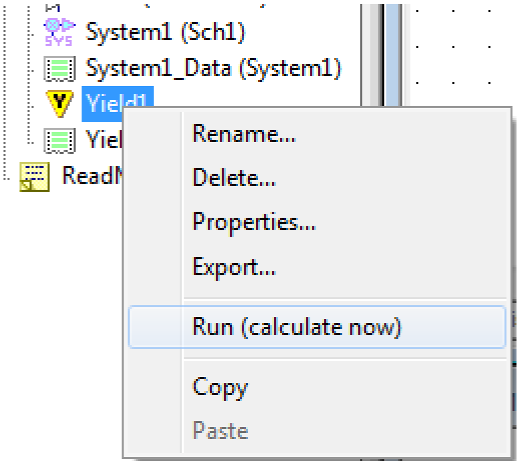
Min: 1.5 dB

Max: dB

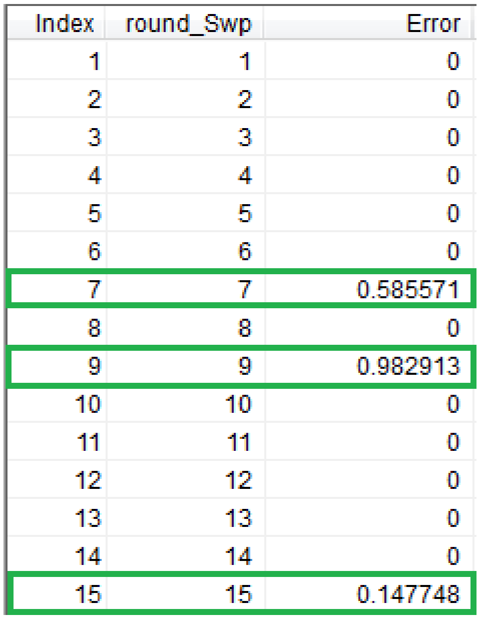
Standard Deviation: 0.5 dB

□ Use Percentages (%)

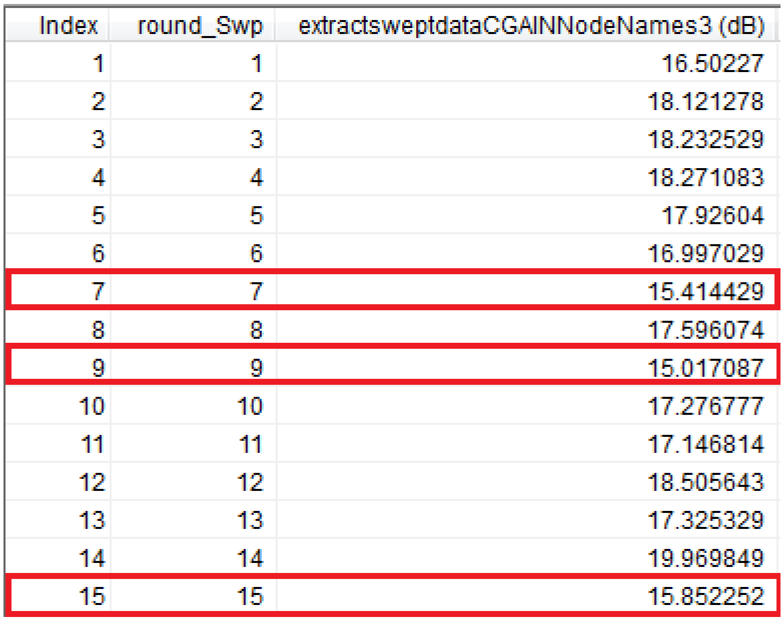
6. Run the Yield evaluation.



7. View Yield output data.
After the yield has finished calculating, the yield output dataset will contain a variable for each target measurement defined. Each variable contains an array of measurement values, one for each round of calculation. The result of the target evaluation for each round is stored in the *Error* variable. An *Error* value of zero represents a passing round. Comparing the data, we can see that during the first 15 rounds there were 3 errors. Recalling that our targets were *CGAIN > 16* and *CNF < 12* we can see that CGAIN contributed to the first three failures.

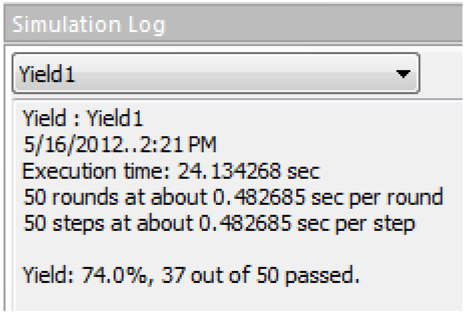


| Index | round_Swp | Error |
|---|---|---|
| 1 | 1 | 0 |
| 2 | 2 | 0 |
| 3 | 3 | 0 |
| 4 | 4 | 0 |
| 5 | 5 | 0 |
| 6 | 6 | 0 |
| 7 | 7 | 0.585571 |
| 8 | 8 | 0 |
| 9 | 9 | 0.982913 |
| 10 | 10 | 0 |
| 11 | 11 | 0 |
| 12 | 12 | 0 |
| 13 | 13 | 0 |
| 14 | 14 | 0 |
| 15 | 15 | 0.147748 |



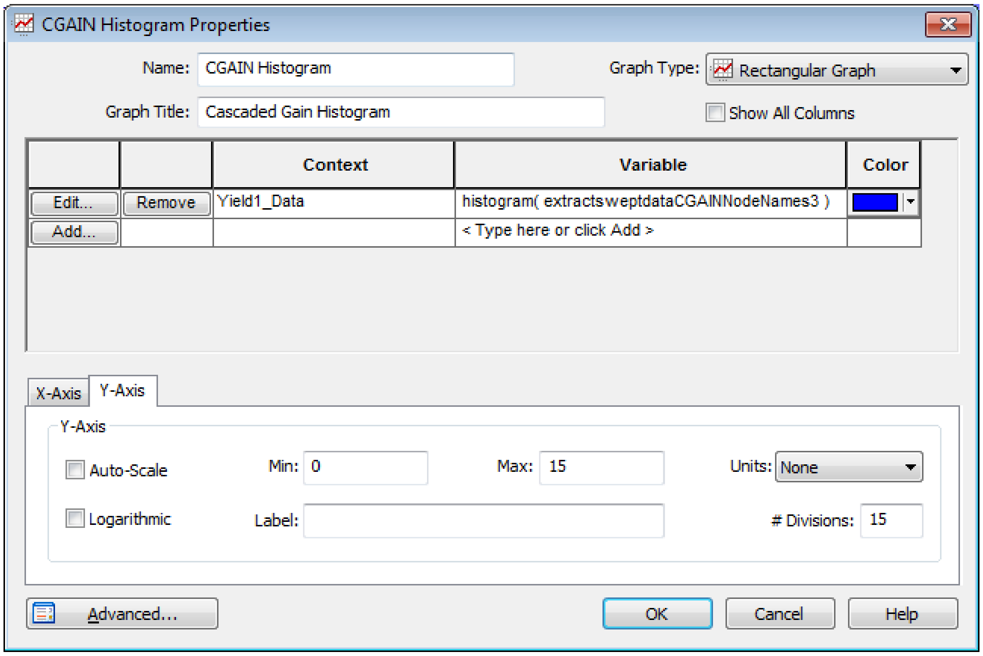
| Index | round_Swp | extractsweptdataCGAINNodeNames3 (dB) |
|---|---|---|
| 1 | 1 | 16.50227 |
| 2 | 2 | 18.121278 |
| 3 | 3 | 18.232529 |
| 4 | 4 | 18.271083 |
| 5 | 5 | 17.92604 |
| 6 | 6 | 16.997029 |
| 7 | 7 | 15.414429 |
| 8 | 8 | 17.596074 |
| 9 | 9 | 15.017087 |
| 10 | 10 | 17.276777 |
| 11 | 11 | 17.146814 |
| 12 | 12 | 18.505643 |
| 13 | 13 | 17.325329 |
| 14 | 14 | 19.969849 |
| 15 | 15 | 15.852252 |

Yield also creates two output variables called *Yield* and *YieldRatio*. The variable *Yield* is the number of rounds that passed, and the variable *YieldRatio* is the percentage of rounds that passed. This information is also displayed in the Simulation log as seen below.



```
Yield : Yield1
5/16/2012..2:21 PM
Execution time: 24.134268 sec
50 rounds at about 0.482685 sec per round
50 steps at about 0.482685 sec per step

Yield: 74.0%, 37 out of 50 passed.
```

8. Plotting the results.
The histogram function is used to plot the results from the yield dataset.



Name: CGAIN Histogram

Graph Type: Rectangular Graph

Graph Title: Cascaded Gain Histogram

□ Show All Columns

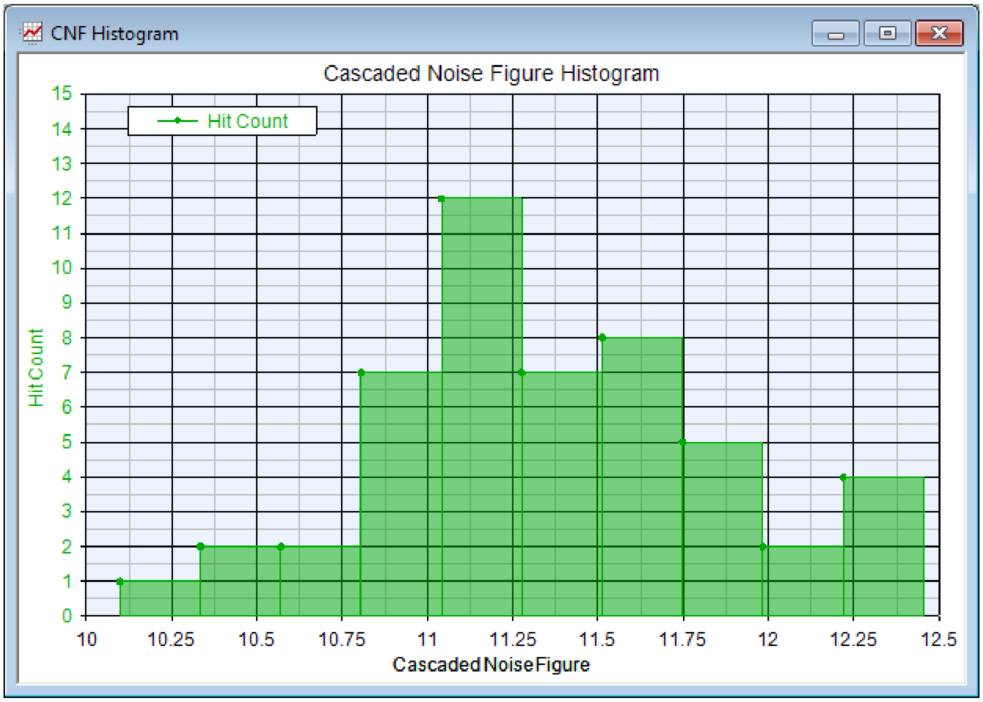
|  |  | Context | Variable | Color |
|---|---|---|---|---|
| Edit... | Remove | Yield1_Data | histogram( extractsweptdataCGAINNodeNames3 ) | 🟦 |
| Add... |  |  | < Type here or click Add > |  |

Y-Axis

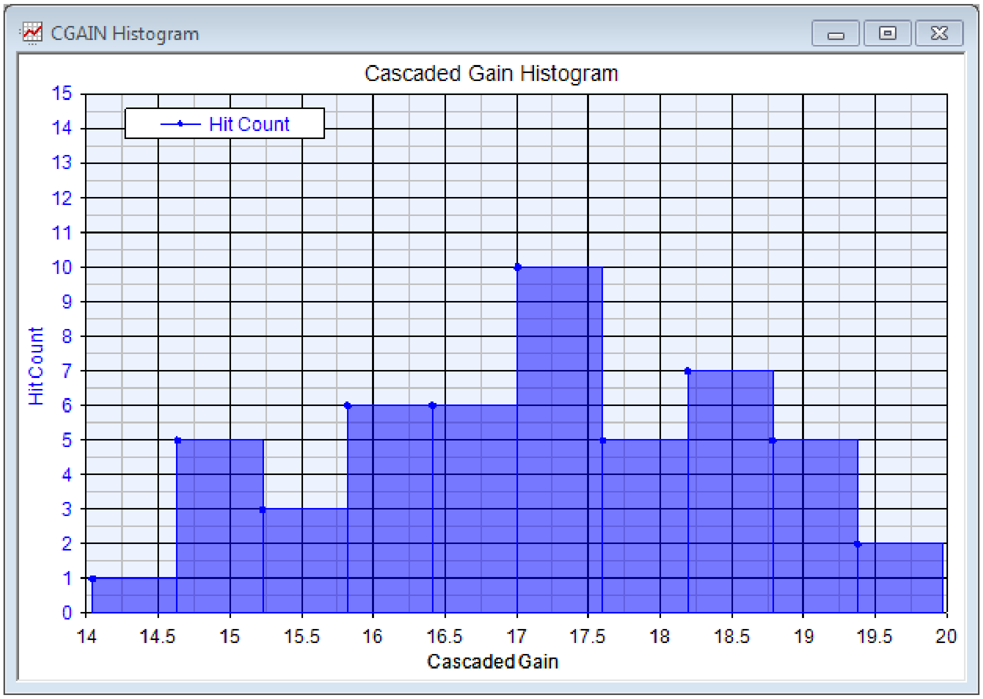
□ Auto-Scale Min: 0 Max: 15 Units: None

□ Logarithmic Label: # Divisions: 15

This is the histogram of the cascaded noise figure from the yield data *extractsweptdataCNFNodeNames3*.



The setup for the cascaded gain graph is identical to that of the cascaded noise figure except the yield data *extractsweptdataCGAINNodeNames3* is used instead.



# Sweeping a Simulation

## Sweeping a Simulation

This section has a series of tutorials that demonstrate how to run a sweep on designs created in Spectrasys or Data Flow.

## Sweep for Spectrasys Designs

Sweep Spectrasys Example

## Sweep for Data Flow Designs

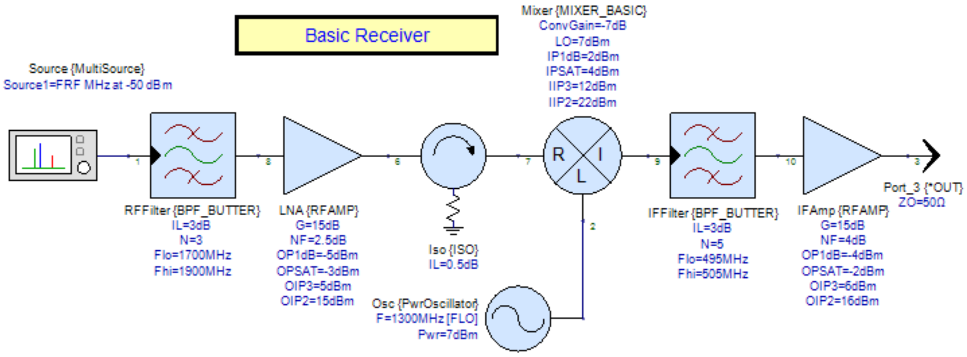
For all the tutorial examples mentioned here please use the MonteCarloForDataFlow.wsv workspace under <SystemVue Installation Directory>\Examples\Tutorials\Simulation_Control_and_Scripting directory.

## Sweep Spectrasys Example (Receiver Sweep.wsv)

This example illustrates how to use a sweep to determine the system performance of a basic receiver. The Cascaded Gain for the receiver will be examined across the receiver input frequency range.

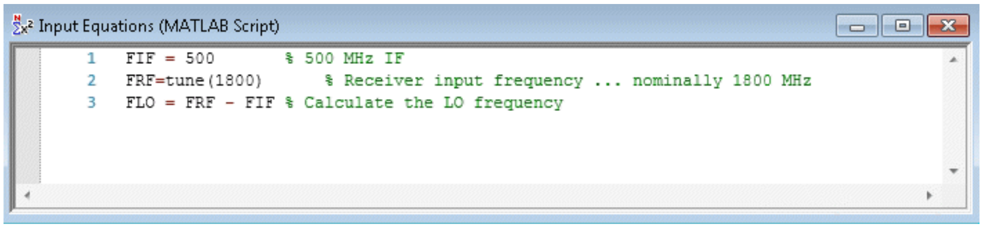
**NOTE** **For this tutorial example, click Help / Open Example in SystemVue and open the Tutorials\Simulation_Control_and_Scripting\Receiver Sweep.wsv workspace.**

1. Create a schematic.



2. Create a System Analysis with a Path
   The sweep controls an analysis. In this case a system analysis. Since path measurements will be plotted a path must be added also. See Getting Started with Spectrasys for additional information on adding a system analysis and a path.

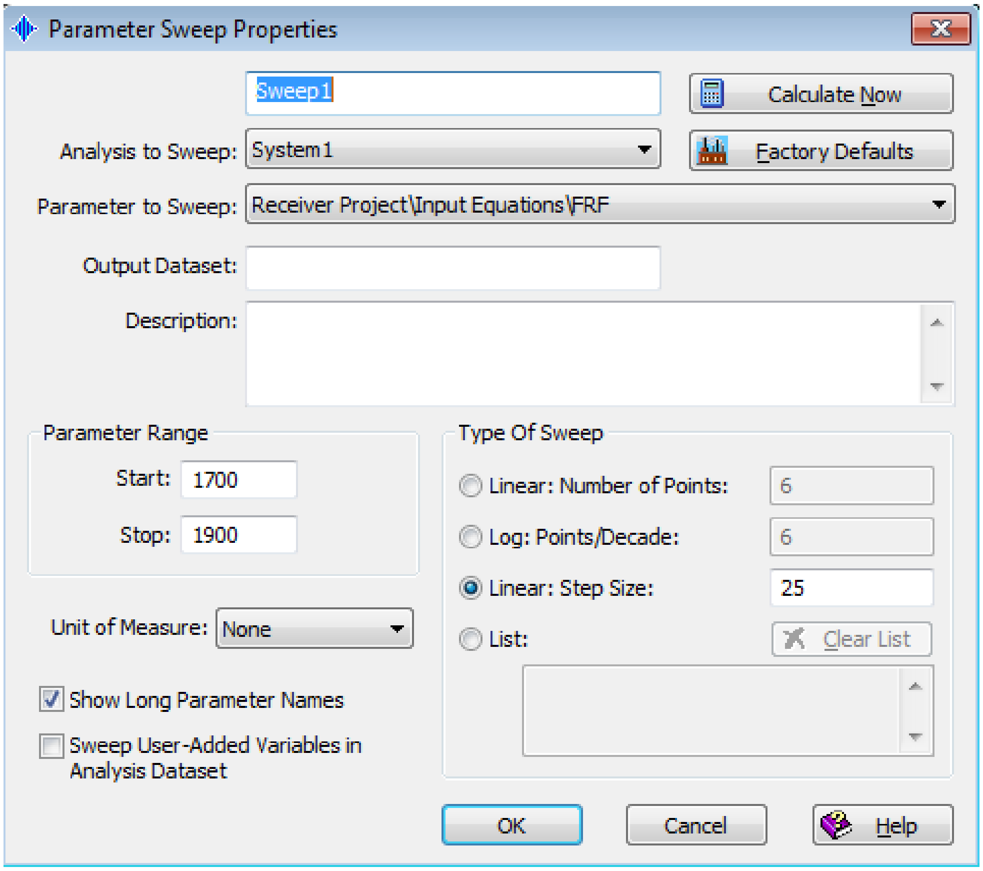
3. Create variables that the sweep will control.
   A tuned variable is needed for the sweep. A tuned variable can be created in the equation block or by checking the Tune checkbox for a given part parameter. In this case, an equation block will be used to tie the RF, LO, and IF frequencies together.



```
FIF = 500         % 500 MHz IF
FRF=tune(1800)         % Receiver input frequency ... nominally 1800 MHz
FLO = FRF - FIF % Calculate the LO frequency
```

**NOTE** The receiver input frequency variable **FRF** must be set as the frequency in the Source and the LO frequency variable **FLO** must be set for the frequency of the oscillator.

4. Create a Sweep evaluation.
   This particular sweep has been configured to sweep the FRF variable from 1700 to 1900 in steps of 25. The units for these variables are specified by the parts using these variables in the schematic and are in MHz.



Parameter Sweep Properties

Sweep1

Analysis to Sweep: System1

Parameter to Sweep: Receiver Project\Input Equations\FRF

Output Dataset:

Description:

Parameter Range

Start: 1700

Stop: 1900

Unit of Measure: None

☑ Show Long Parameter Names

☐ Sweep User-Added Variables in Analysis Dataset

Type Of Sweep

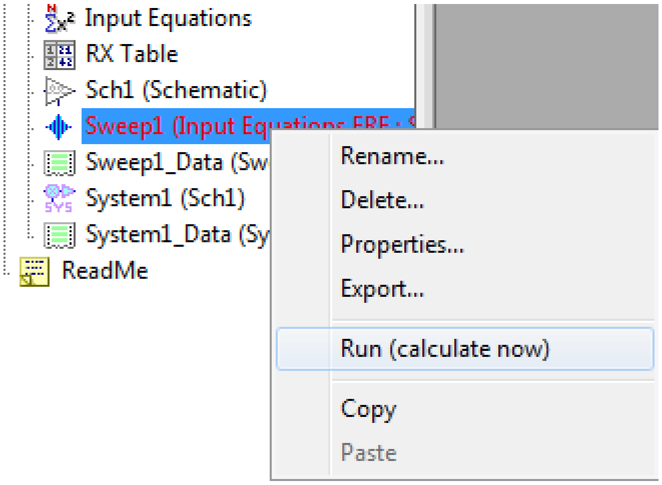
○ Linear: Number of Points: 6

○ Log: Points/Decade: 6

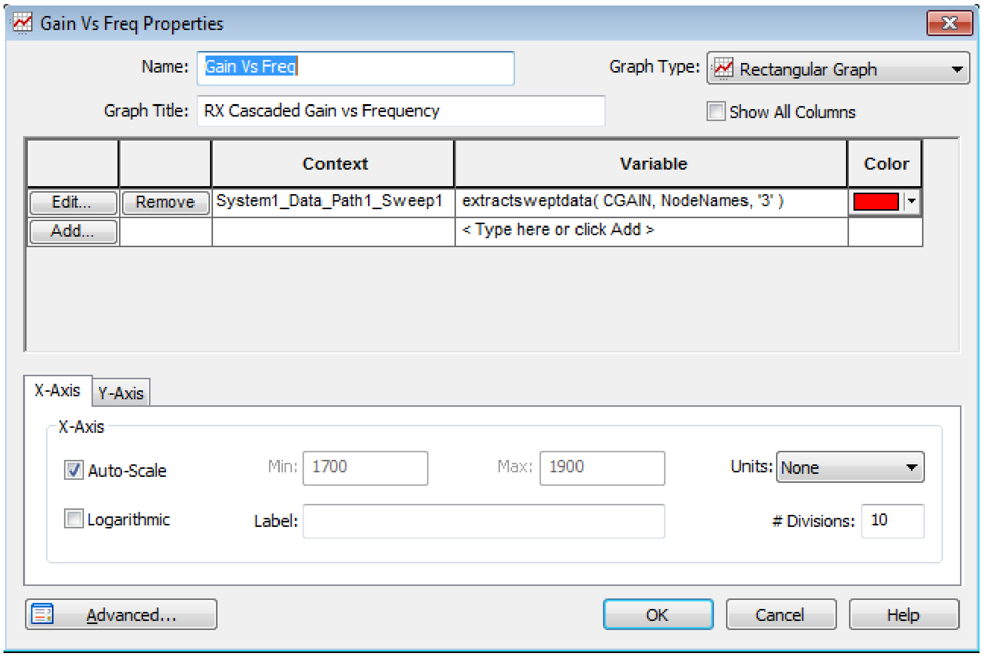
● Linear: Step Size: 25

○ List:

5. Run the Sweep



6. Plotting the results
When plotting the results of a system or Spectrasys path measurement an additional function is needed to extract swept data at a particular node along the path. The extractsweptdata function is used for that purpose. After creating a new graph, the measurement can be entered as follows. This measurement shows that the Cascaded Gain (CGAIN) measurement for the node named '3' will be plotted.



Name: Gain Vs Freq

Graph Type: Rectangular Graph

Graph Title: RX Cascaded Gain vs Frequency

□ Show All Columns

| | | Context | Variable | Color |
|---|---|---|---|---|
| Edit... | Remove | System1_Data_Path1_Sweep1 | extractsweptdata( CGAIN, NodeNames, '3' ) | 🟥 |
| Add... | | | < Type here or click Add > | |

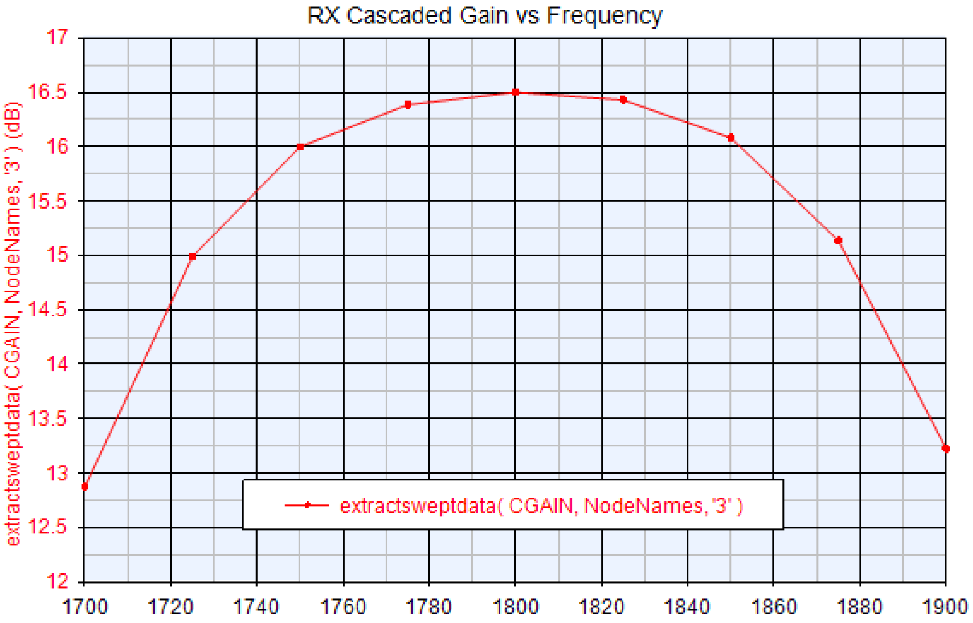
X-Axis

☑ Auto-Scale  Min: 1700  Max: 1900  Units: None

□ Logarithmic  Label:  # Divisions: 10

**NOTE** Port numbers are automatically generated based on the order in which the ports (or sources) are placed in the schematic. In the schematic for this example, the LO is port 2 and the output is port 3. If the order of the parts on your schematic is different from this, you will need to select the correct node name in the measurement to display same results.

This is of a plot of the cascaded gain as a function of the swept receiver input frequency. The response shows the anticipated drop in power at the front end filter corner frequencies of 1700 and 1900 MHz by 3 dB.



## Libraries and Applications

- Getting Started With DPD

## Getting Started With DPD

### Getting Started With DPD

Contents

### Measurement Platforms

#### 1. Hardware Requirements

DPD hardware verification platform needs a Keysight Modular instrument and/or Keysight signal generator (ESG/PSG/MXG), a Keysight signal analyzer (PSA/MXA /PXA) and the power amplifier DUT. Throughout this document, ESG and PXA are chosen as an example.

The connection of the instruments is shown in the following figure. The "10 MHz OUT" of the Keysight signal generator should be connected to the "EXT REF IN" input on the signal analyzers. Also, the "EVENT1" output of the signal generator should be connected with "EXT Trigger" input 1 of the signal analyzer.
Capture PA Input:

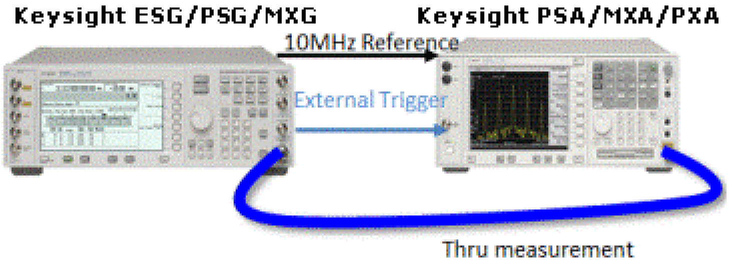


*Figure 1. Connection of the Keysight signal generator, Keysight PXA*

**Capture PA Output**:

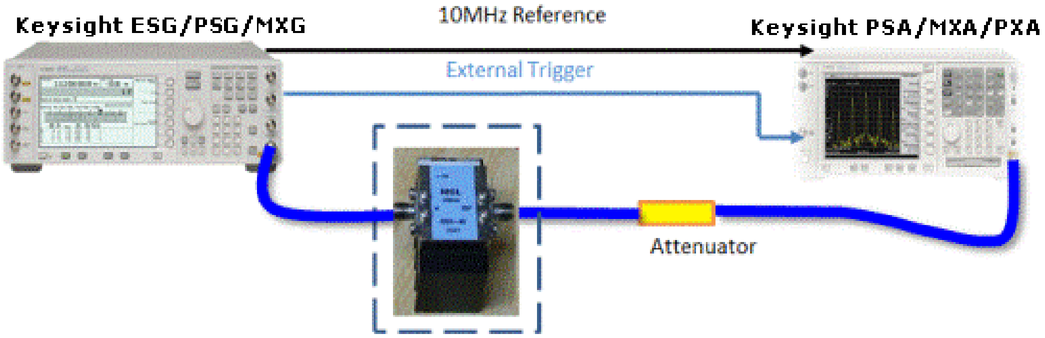
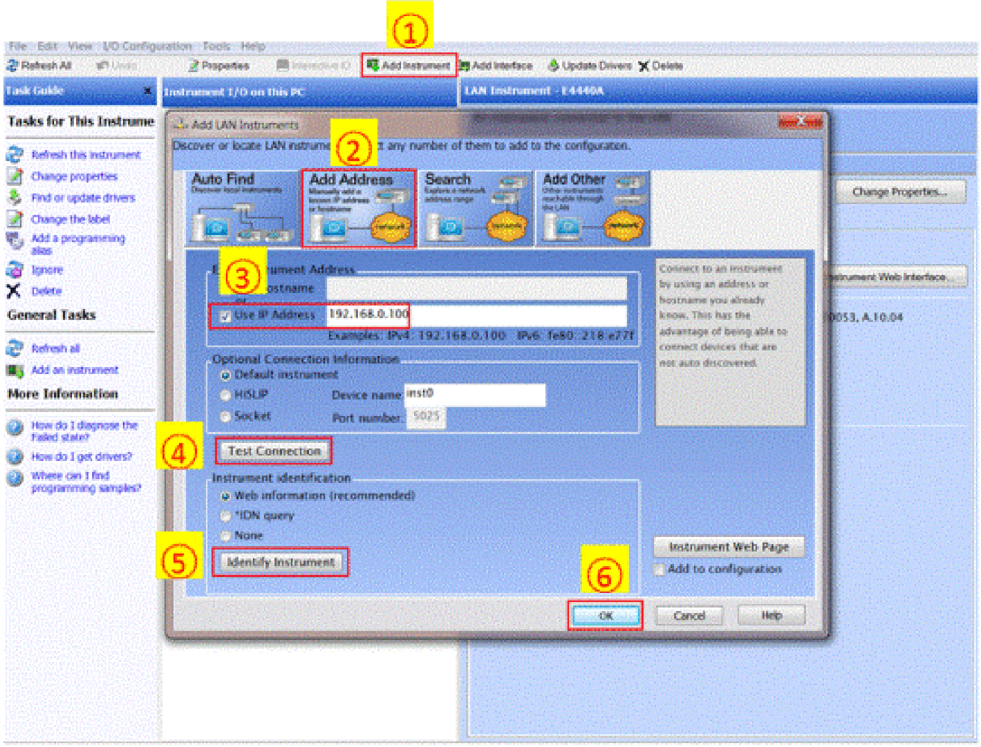


*Figure 2. Connection of the Keysight ESG, Keysight PXA and the amplifier device under test*

## 2. Software Requirements

1. Keysight IO Libraries Suite
   Keysight IO Libraries Suite is needed to capture data from Keysight spectrum analyzers. It is recommended to use the latest version i.e. 16.2.15823.0.
   Open Keysight Connection Expert from folder 'Keysight IO Libraries Suite' in the Start menu. Click 'Add Instrument' in the toolbar to add the signal analyzer. It can be manually added by specifying the IP address or hostname.



Discover or locate LAN instrume t any number of them to add to the configuration.

Instrument Address

Hostname

☑ Use IP Address 192.168.0.100

Examples: IPv4: 192.168.0.100 IPv6: fe80::218:e77f

Optional Connection Information

● Default instrument

○ HiSLIP Device name: inst0

○ Socket Port number: 5025

Instrument identification

● Web information (recommended)

○ *IDN query

○ None

□ Add to configuration

Connect to an instrument by using an address or hostname you already know. This has the advantage of being able to connect devices that are not auto discovered.

2. Data Capture tools
   Two methods are supported to capture PA response data from vector signal analyzer (MXA/PXA, or PXI modular): Keysight 89600 VSA software and Keysight Command Expert.

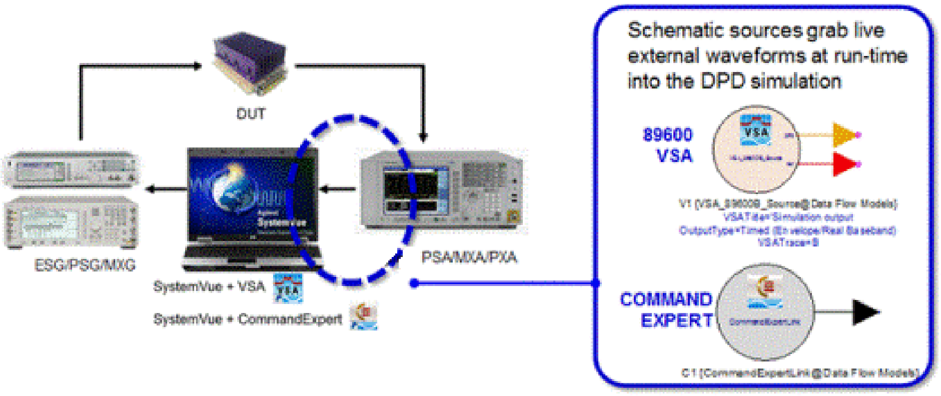

DUT
ESG/PSG/MXG
PSA/MXA/PXA
SystemVue + VSA
SystemVue + CommandExpert
Schematic sources grab live external waveforms at run-time into the DPD simulation
89600 VSA
VSA
V1 {VSA_89600B_Source@Data Flow Models}
VSATitle=Simulation output
OutputType=Timed (Envelope/Real Baseband)
VSATrace=B
COMMAND EXPERT
CommandExpertLink
C1 {CommandExpertLink@Data Flow Models}

*Figure 3. Two ways to transfer data from Instruments to SystemVue*

   a. 89600 VSA software. It is very convenient, but the license is required.

      Firstly please make 89600 VSA software gets the remote control of the signal analyzer. Open 89600 VSA software and click Utilities->Hardware->Configurations. In the popup, click the '+' button to add new configurations. In the instrument list, select the target instrument. You may want to rename this configuration before clicking OK. Don't forget to select the new configuration as the current Analyzer configuration after clicking OK.

b. Command Expert. It is for free but it requires extra knowledge about SCPI commands to do some customization according to the instrument type.

  - Version: Command Expert 1.1 or higher version
  - Download location: http://www.keysight.com/find/commandexpert
  - CommandSet: X-Series Signal Analyzers LTE FDD Commands / A.08.03
  - Support Models: N9030A PXA, N9020A MXA

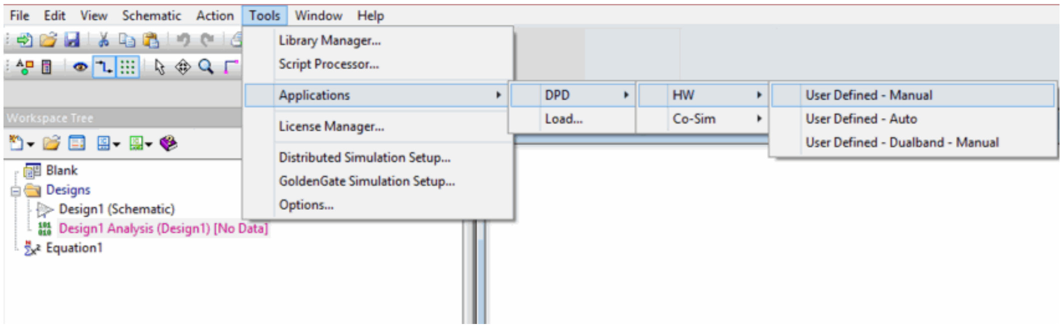
### 3. Platform Calibration

There would be power gain or loss in the path between signal generator and the power amplifier DUT due to preamplifier (if any) or cable. In other words, the power at the input port of the DUT would in most cases not equal to the RF power set in the signal generator. This difference would influence the performance of DPD. Hence, the path gain/loss should be measured and taken into consideration in the DPD solution. The calibration process will be discussed in Measure Path Gain/Loss.

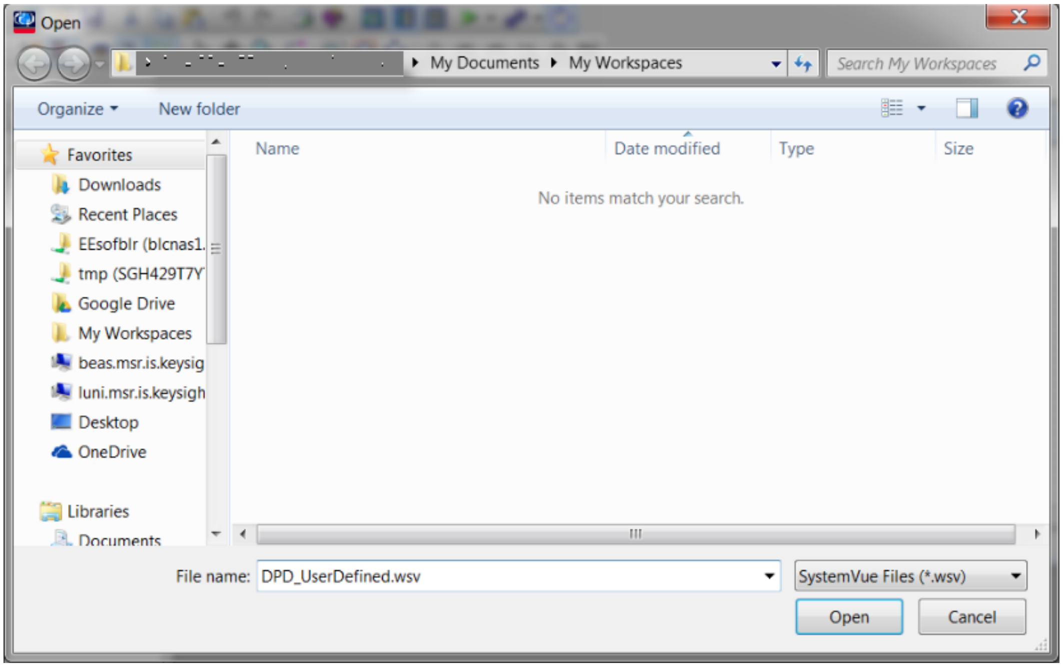
## Hardware DPD Measurement - Manual

### 1. Open DPD UI for Hardware DPD Measurement - Manual

To open DPD UI for manual flow, click Tools->Applications->DPD->HW->User Defined – Manual.



The following window will pop up.

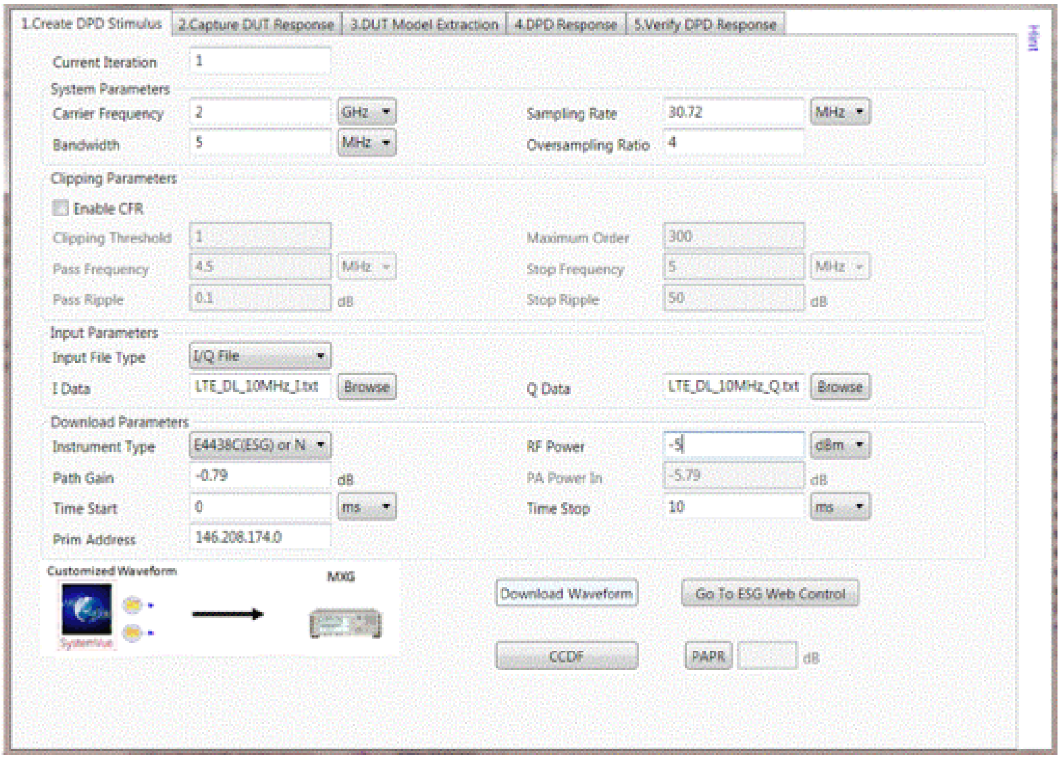


No items match your search.

File name: DPD_UserDefined.wsv

SystemVue Files (*.wsv)

To use the example workspace, it is strongly recommended to copy the whole example folder **DPD UserDefined HardwareVerfication** from **$SystemVue\Examples\DPD** to a writable folder, where **$SystemVue** stands for the SystemVue installation folder, typically **C:\Program Files\SystemVue<Version_Number>** . If you have done so, navigate to the new destination in above dialog, select **DPD_UserDefined.wsv** and click open. The following GUI will be shown. We'll discuss the details in this GUI later.



Current Iteration: 1

System Parameters

Carrier Frequency: 2 GHz

Bandwidth: 5 MHz

Sampling Rate: 30.72 MHz

Oversampling Ratio: 4

Clipping Parameters

□ Enable CFR

Clipping Threshold: 1

Pass Frequency: 4.5 MHz

Pass Ripple: 0.1 dB

Maximum Order: 300

Stop Frequency: 5 MHz

Stop Ripple: 50 dB

Input Parameters

Input File Type: I/Q File

I Data: LTE_DL_10MHz_I.txt

Q Data: LTE_DL_10MHz_Q.txt

Download Parameters

Instrument Type: E4438C(ESG) or N

Path Gain: -0.79 dB

Time Start: 0 ms

Prim Address: 146.208.174.0

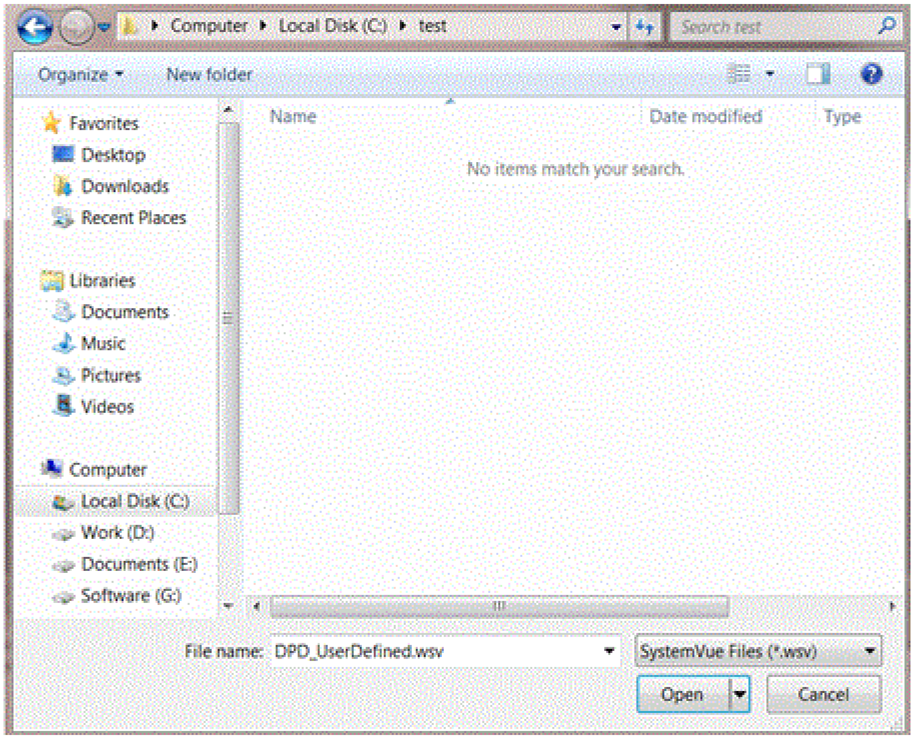
RF Power: -5 dBm

PA Power In: -5.79 dB

Time Stop: 10 ms

PAPR: dB

Otherwise, if you haven't copied the files, the GUI can copy the workspace for you. Please navigate to the folder in which you want to save the files, e.g. **C:\test** and click open.

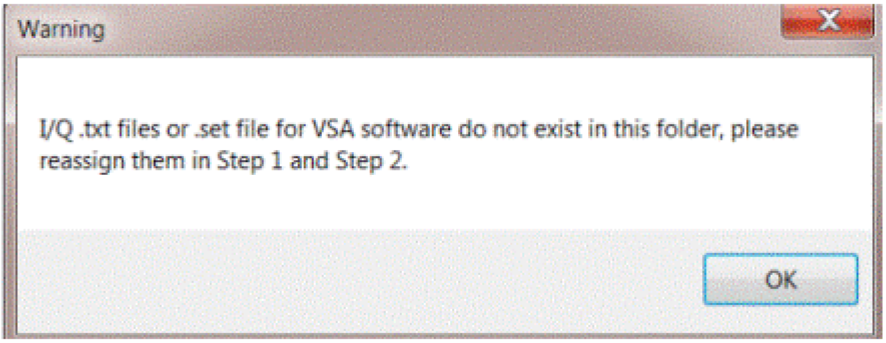


No items match your search.

File name: DPD_UserDefined.wsv

SystemVue Files (*.wsv)

The GUI will copy **DPD_UserDefined.wsv** to the new folder. However, other files such as I/Q data files and the .setx file will not be copied automatically. You still need to copy these files manually. The following warning message will be popped up in this case.



I/Q .txt files or .set file for VSA software do not exist in this folder, please reassign them in Step 1 and Step 2.

If you want to open an existing workspace other than the example, please directly open the workspace in 'open' dialog shown above.

It should be noted that there are strong dependencies between the GUI and the workspace. The GUI may not work properly if you modify the corresponding workspace.

## 2. Step 1: Create DPD Stimulus

Step 1 (Create DPD Stimulus) is to download waveform into a signal generator.

1. Set Current Iteration
   Iterative DPD is supported in SystemVue. This parameter specifies the current iteration. Please refer to Current Iteration for more details.

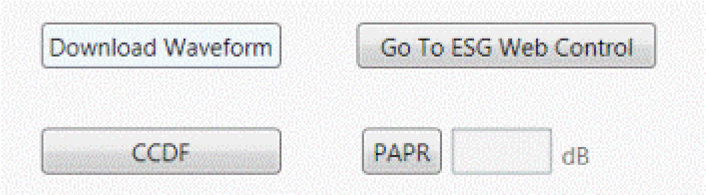
2. Set System Parameters
   This group of parameters specifies the system configurations. Please refer to System Parameters for more details.
   It should be noted that Bandwidth and Oversampling Ratio here are just for display. They are not actually used in the workspace. The real sampling rate of the signal is explicitly set by Sampling Rate. In other words, the setting of Bandwidth and Oversampling Ratio would not affect the signal. These two textboxes are just for you to note down the signal characteristics.

3. Set Clipping Parameters
   Multi-carrier signals typically have high PAPR which would drive PA to its saturation region very quickly and lower the PAE. A classic clipping and filtering method is provided in SystemVue for Crest Factor Reduction. Please refer to Clipping Parameters for more details.

4. Set Input Parameters
   The stimulus signal can be read from I/Q data files (.txt), Signal Studio waveforms (.wfm) and 89600 VSA software recordings (.sdf), or generated by wireless libraries in SystemVue. Please refer to Input Parameters for more details.

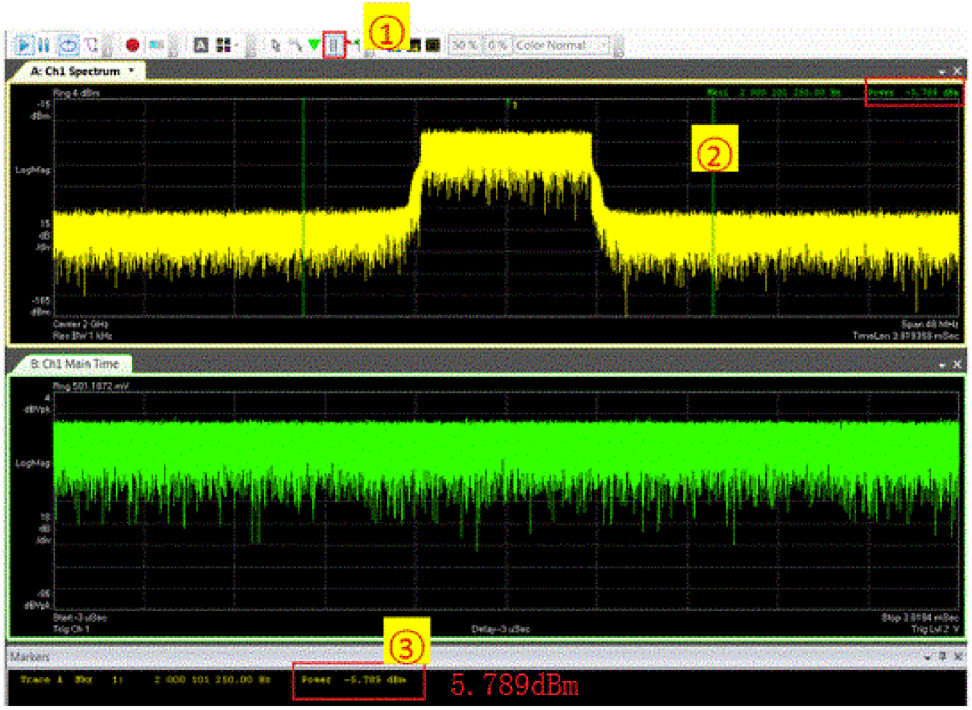
5. Set Download Parameters
   After configuring the input signal, now we set the download parameters. Please refer to Download Parameters for more details.

6. Download Waveforms
   After setting all parameters, now we can click Download Waveform button to download the signals into the signal generator. Please refer to Create DPD Stimulus for more details.
   You may notice that there are obvious differences between the background of ‘Download Waveform’ button and the other buttons e.g. ‘Go To ESG Web Control’. Those buttons with light color are ‘MUST DO’ buttons, which means these actions are mandatory in DPD manual flow. Clicking those buttons will run a corresponding analysis in the workspace. Those buttons with dark color correspond to optional actions, clicking which will either show intermediate results or bring certain graphs in front.



7. Show Results
   Check the results in this step. Please refer to DPD Stimulus Results for more details.

8. Measure Path Gain/Loss
   After downloading signal to signal generator, open VSA software and choose the right configuration so that it connects to the analyzer. Make sure the signal generator and analyzer are connected directly. Click the “band power” button in the toolbar to measure the signal power as shown in the following picture. Read the result in the bottom or the upper right.
   In the following example, the RF Power in the signal generator is set to -5 dBm, while the measured power from VSA is about -5.79dBm, hence the path gain is -0.79dB.



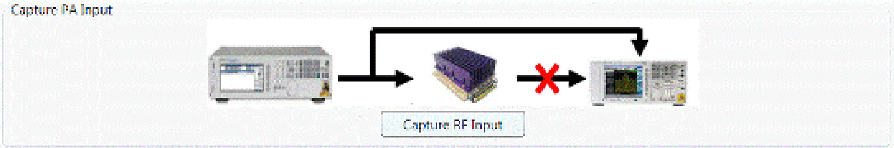
   You can also measure the power in the signal analyzer and calculate the Path Gain.

## 3. Step 2: Capture DUT Response

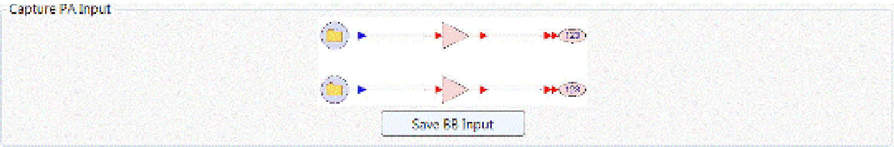
Step 2 (Capture DUT Response) is to capture the input and output signals of power amplifier DUT from PSA/MXA/PXA.
Through Step 2 to Step 5, parameter Current Iteration is read-only. Upon completion of each iteration, please change Current Iteration in Step 1. It will be passed to the followed steps automatically. Please refer to Current Iteration for more details.

1. Set Capture Parameters
   Before data capture, please set the capture parameters. Please refer to Capture Parameters for more details.
2. Capture PA Input
   Two approaches are supported for DPD measurement. The first method is to measure both PA input and output signals, the second one is to calculate PA input and measure PA Output.

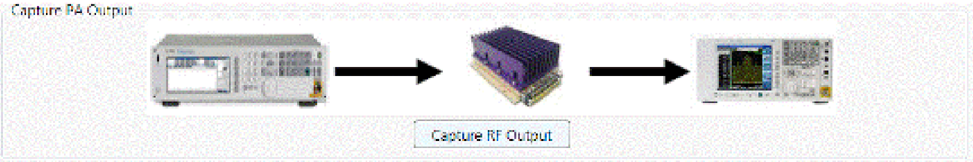
a. Method 1: Measure both PA Input and Output signals
For method 1, please select ‘PA Input Type’ as ‘->’. Connect signal analyzer directly with a signal generator. Then click ‘Capture RF Input’ button to capture PA input signal.



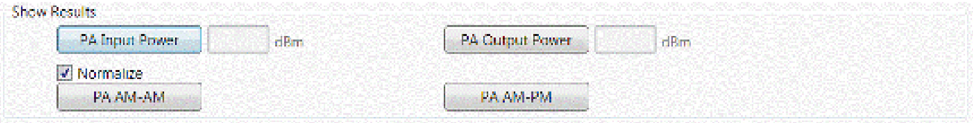
b. Method 2: Calculate PA input, Measure PA output
For method 2, please select ‘PA Input Type’ as ‘Baseband Signal’. The PA input signal is calculated from the baseband design in SystemVue. Click ‘Save BB Input’ to store the input signal.



3. Capture PA Output
After PA input is successfully captured, please connect the PA DUT between the signal analyzer and signal generator. Then click ‘Capture RF Output’ button to capture PA output signal.



4. Show Results
You can click these buttons to check the PA input power, output power, AM-AM characteristics and AM-PM characteristics. Please refer to DUT Response Results for more details.

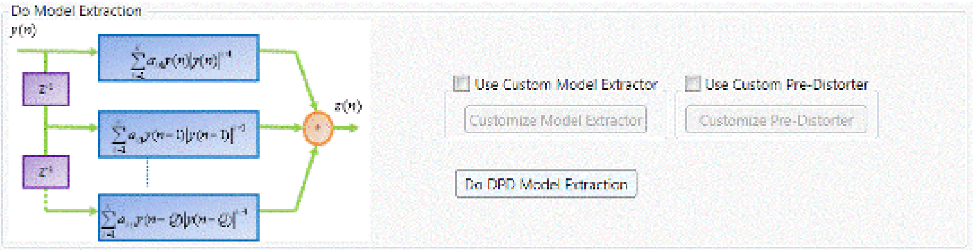


## 4. Step 3: DPD Model Extraction

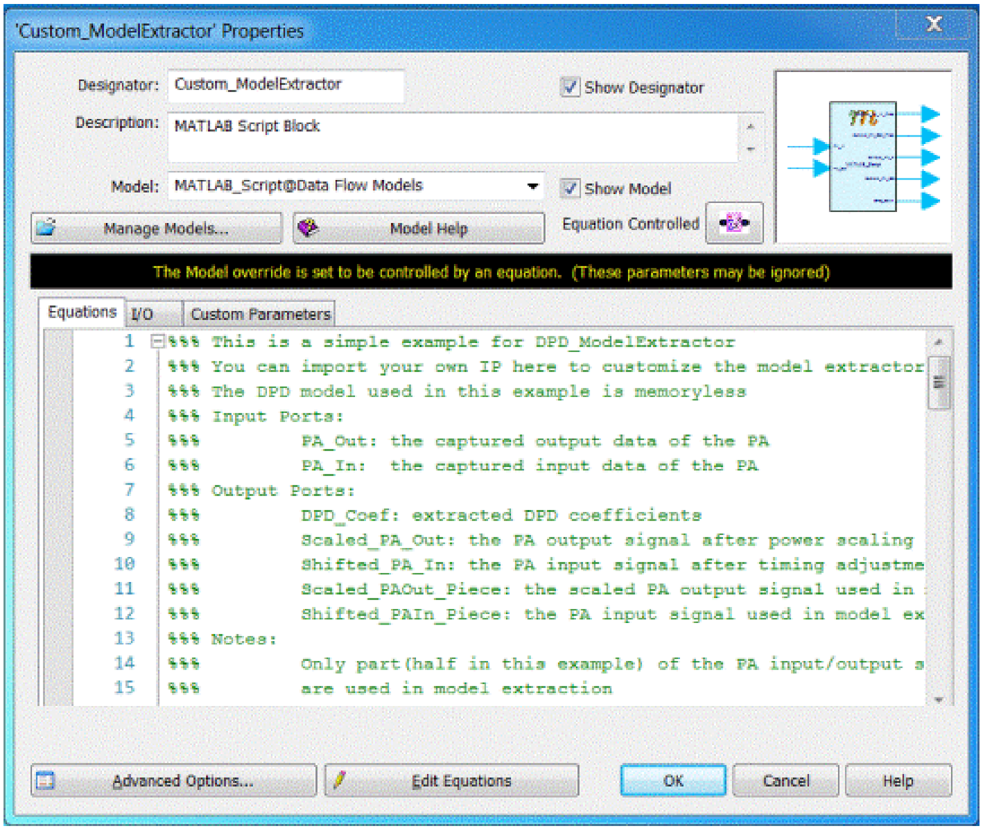
This step is to extract the DPD model using the PA input and output signals captured in the previous step.

1. Set Model Extraction Parameters
Please refer to DPD Setting for more details.

2. Do Model Extraction
   Use Custom Model Extractor: select this checkbox if you want to use your own IP of model extraction. Otherwise, the model extractor provided in SystemVue DPD library will be used.


Do Model Extraction
y(n)
z⁻¹
z⁻¹
z(n)
☐ Use Custom Model Extractor
Customize Model Extractor
☐ Use Custom Pre-Distorter
Customize Pre-Distorter
Do DPD Model Extraction

Once you select to use the custom model extractor, the ‘Customize Model Extractor’ button would be activated. Click this button to open the MATLAB_Script model of the custom model extractor. A simple algorithm is provided in this model. The definition of each input/output port is described in the comments. Please refer to these comments, follow the definition and input your implementation in this model. Then it will be used in the simulation.



Designator: Custom_ModelExtractor

☑ Show Designator

Description: MATLAB Script Block

Model: MATLAB_Script@Data Flow Models

☑ Show Model

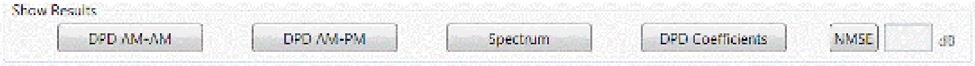
The Model override is set to be controlled by an equation. (These parameters may be ignored)

```
%%% This is a simple example for DPD_ModelExtractor
%%% You can import your own IP here to customize the model extractor
%%% The DPD model used in this example is memoryless
%%% Input Ports:
%%%         PA_Out: the captured output data of the PA
%%%         PA_In:  the captured input data of the PA
%%% Output Ports:
%%%         DPD_Coef: extracted DPD coefficients
%%%         Scaled_PA_Out: the PA output signal after power scaling
%%%         Shifted_PA_In: the PA input signal after timing adjustme
%%%         Scaled_PAOut_Piece: the scaled PA output signal used in
%%%         Shifted_PAIn_Piece: the PA input signal used in model ex
%%% Notes:
%%%         Only part(half in this example) of the PA input/output s
%%%         are used in model extraction
```

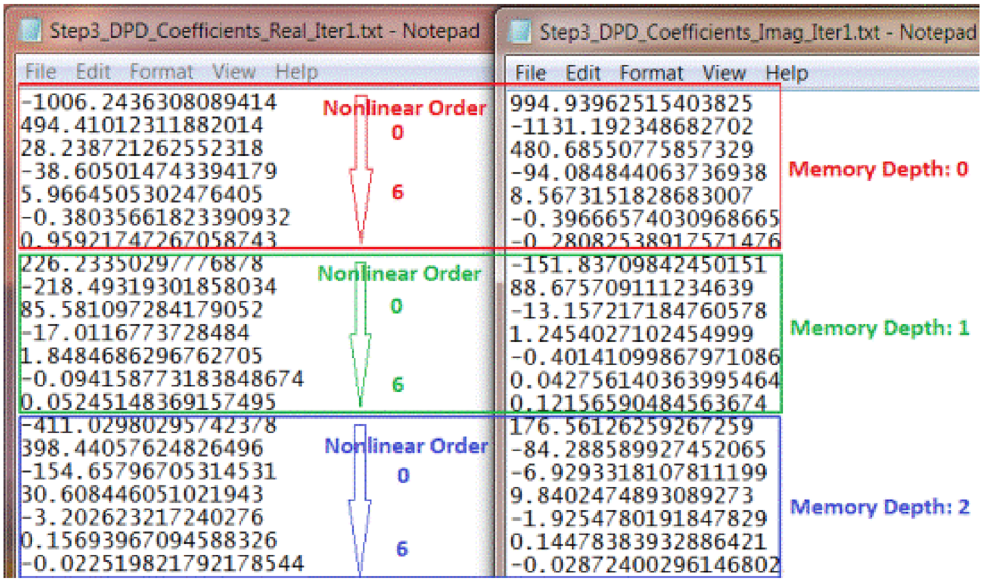
Use Custom Pre-Distorter: similar with ‘Use Custom Model Extractor’, you can import your own IP for pre-distorter.
It should be noted that typically you should simultaneously enable/disable

these two customized models.
Click ‘Do Model Extraction’ to run the analysis. The DPD model coefficients will be saved in two separate text files for the real part and image part respectively.

3. Show Results
   Click these buttons to show DPD AM-AM characteristics, DPD AM-PM characteristics, spectrum, DPD model coefficients as well as NMSE.



In this example, memory order and nonlinear order are set to 2 and 7 respectively. The extracted DPD model coefficients are shown below.


Step3_DPD_Coefficients_Real_Iter1.txt - Notepad
File Edit Format View Help
Nonlinear Order
0
6
Nonlinear Order
0
6
Nonlinear Order
0
6
Step3_DPD_Coefficients_Imag_Iter1.txt - Notepad
File Edit Format View Help
Memory Depth: 0
Memory Depth: 1
Memory Depth: 2

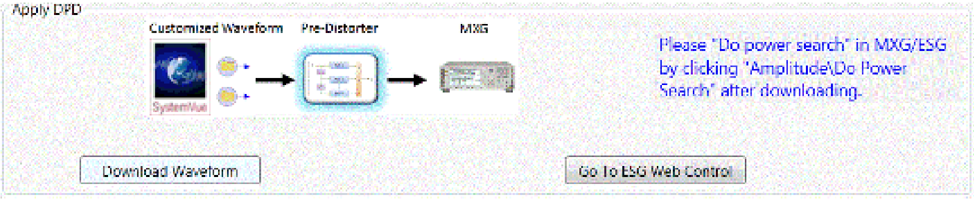
```
-1006.2436308089414
494.41012311882014
28.23872126255231 8
-38.605014743394179
5.9664505302476405
-0.38035661823390932
0.95921747267058743
226.23350297776878
-218.49319301858034
85.581097284179052
-17.0116773728484
1.848468629676 2705
-0.094158773183848674
0.05245148369157495
-411.02980295742378
398.44057624826496
-154.65796705314531
30.608446051021943
-3.202623217240276
0.15693967094588326
-0.022519821792178544
```

```
994.93962515403825
-1131.192348682702
480.68550775857329
-94.084844063736938
8.5673151828683007
-0.39666574030968665
-0.28082538917571476
-151.83709842450151
88.675709111234639
-13.157217184760578
1.2454027102454999
-0.40141099867971086
0.042756140363995464
0.12156590484563674
176.5612625926 7259
-84.288589927452065
-6.9293318107811199
9.8402474893089273
-1.9254780191847829
0.14478383932886421
-0.028724002961468 02
```

## 5. Step 4: Apply DPD

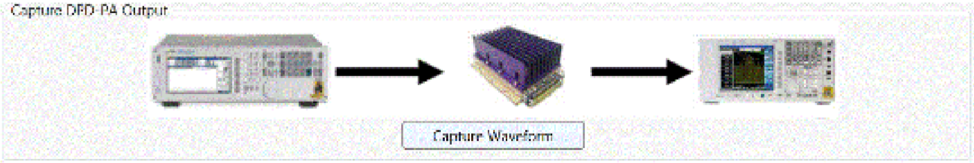
1. Set Download Parameters
   RF Power here indicates the output power of the signal generator, which defines the mean power of the signal w/ DPD. Click Use Default will set RF power to a suggested value at which the extracted DPD model is supposed to give good performance on the PA DUT. This default value is calculated from the lower level design and takes Path Gain into consideration. We may tune this parameter to find the best power at which the DPD model shows the largest improvements.
   The other parameters are read-only and kept the same as in Step 1. Please refer to Download Parameters for more details on the other parameters.

2. Apply DPD
   Click Download Waveform button to download the pre-distorted signal into the signal generator.
   If ALC is OFF in the signal generator, please Do Power Search by clicking *Amplitude->Do Power Search* after signal downloading.

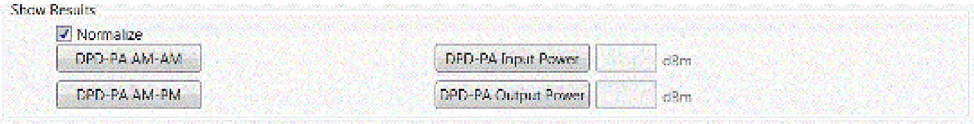


Please "Do power search" in MXG/ESG by clicking "Amplitude\Do Power Search" after downloading.

3. Capture DPD-PA Output
   Click Capture Waveform to capture the DPD-PA response.



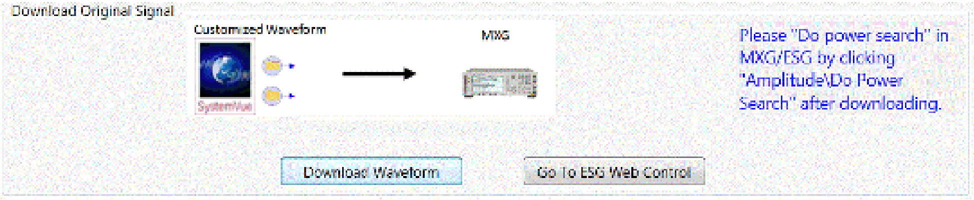
4. Check DPD-PA Response
   Click the following buttons to check the DPD-PA response, including AM-AM characteristics, AM-PM characteristics, input power and output power of the combination of DPD+PA. Please refer to DPD Response Results for more details.



## 6. Step 5: Verify DPD Response

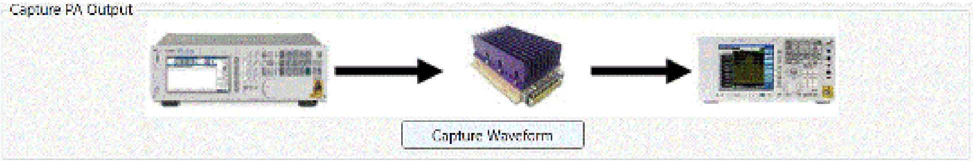
This step is to verify DPD response and compare the results in terms of spectrum, ACP and/or EVM.

1. Check Download Parameters
   The download parameters are kept the same as in Step 4, so that we can compare the results of w/ DPD and w/o DPD under the same conditions.
2. Download Signal
   Click Download Waveform button to download the original signal (w/o DPD) into the signal generator.

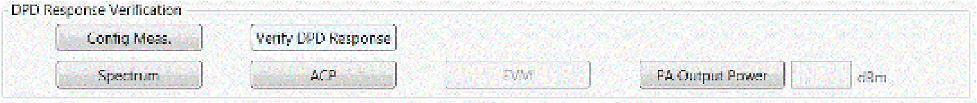


Please "Do power search" in MXG/ESG by clicking "Amplitude\Do Power Search" after downloading.

3. Capture Waveform
   Click Capture Waveform button to capture the PA output signal.



4. Verify DPD Response
   Click Config Meas. button to configure the spectrum, ACP and EVM measurement. Then click Verify DPD Response button to run the analysis. Then click Spectrum, EVM (if active), ACP and PA Output Power button to check the results comparison.



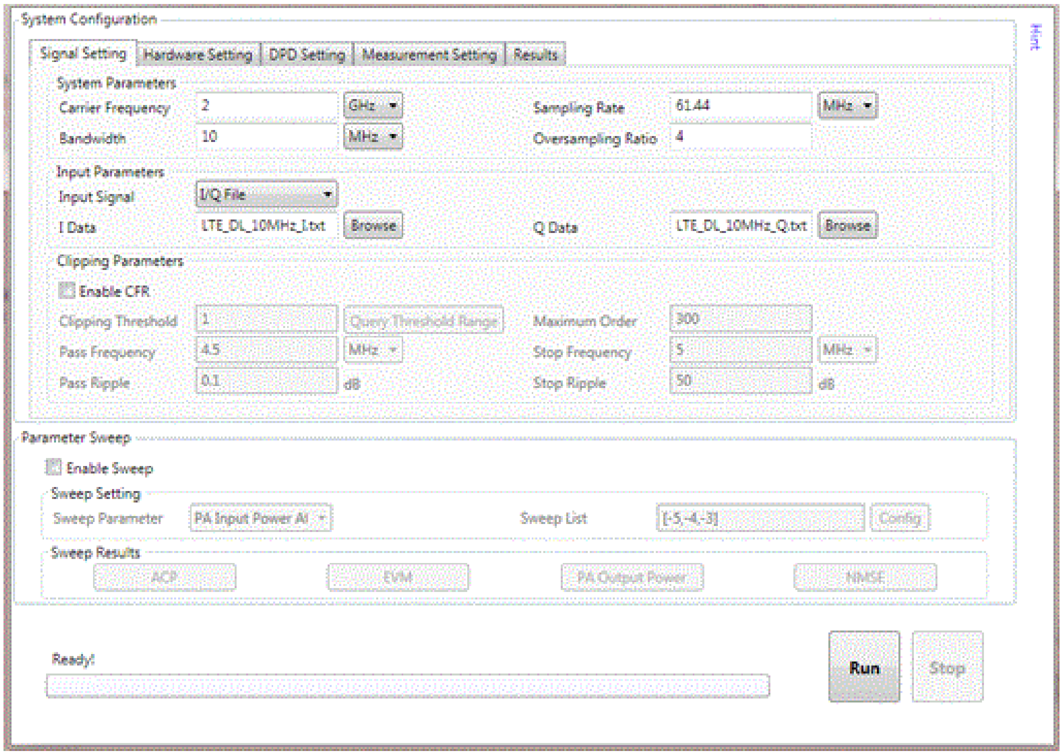
   Please refer to DPD Response Verification Results for more details.

## Hardware DPD Measurement - Auto

In SystemVue, DPD measurement automation solution is provided, which is very convenient to use.

### 1. Open DPD UI for Hardware DPD Measurement – Auto

To open DPD UI for auto flow, please click Tools->Applications->DPD->HW->User Defined – Auto. The autoflow uses the same example workspace as the manual flow, i.e. DPD_UserDefined.wsv in $SystemVue\Examples\DPD\DPD UserDefined HardwareVerfication, where $SystemVue stands for the SystemVue installation folder, typically C:\Program Files\SystemVue<Version_Number>. For more details, see Open DPD UI for Hardware DPD Measurement - Manual.
The following UI is displayed if it is successfully opened.



System Configuration

System Parameters

Carrier Frequency 2 GHz

Bandwidth 10 MHz

Sampling Rate 61.44 MHz

Oversampling Ratio 4

Input Parameters

Input Signal I/Q File

I Data LTE_DL_10MHz_I.txt

Q Data LTE_DL_10MHz_Q.txt

Clipping Parameters

□ Enable CFR

Clipping Threshold 1

Pass Frequency 4.5 MHz

Pass Ripple 0.1 dB

Maximum Order 300

Stop Frequency 5 MHz

Stop Ripple 50 dB

Parameter Sweep

□ Enable Sweep

Sweep Setting

Sweep Parameter PA Input Power Al

Sweep List [-5,-4,-3]

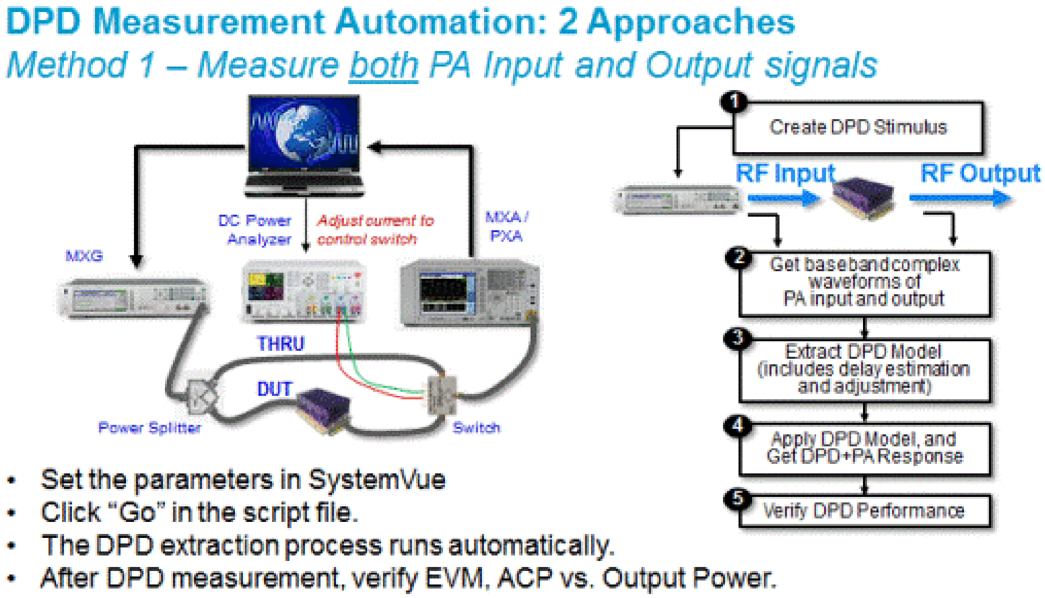
Sweep Results

Ready!

## 2. Platform Setup

Two approaches are supported for DPD measurement automation. The first method is to measure both PA input and output signals, the second one is to calculate PA input and measure PA Output.

The first method is depicted in the following figure. As can be seen, a power splitter, a switch and a DC power analyzer are needed. The DC power analyzer is controlled by SystemVue so that it adjusts current to control switch. To capture PA input signal, the upper line is turned on so that the signal is directly transmitted from MXG to MXA. To capture PA output signal, the lower line is turned on so that the signal is transmitted through PA DUT. The whole flow is automated in SystemVue. Users can just click the Run button in SystemVue to start the measurement. This method is most accurate, but requires additional signal routing as depicted.



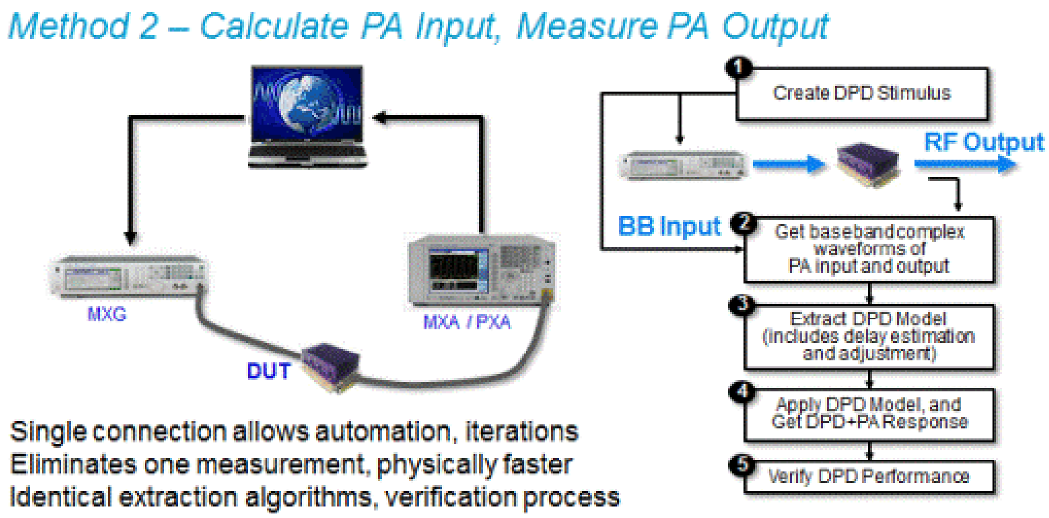
## DPD Measurement Automation: 2 Approaches

*Method 1 – Measure both PA Input and Output signals*

1. Create DPD Stimulus
2. Get baseband complex waveforms of PA input and output
3. Extract DPD Model (includes delay estimation and adjustment)
4. Apply DPD Model, and Get DPD+PA Response
5. Verify DPD Performance

- Set the parameters in SystemVue
- Click "Go" in the script file.
- The DPD extraction process runs automatically.
- After DPD measurement, verify EVM, ACP vs. Output Power.

The second method is depicted in the following figure, which uses only 1 real RF measurement as the PA input signal is estimated by simulation, while the PA output signal is captured from the measurement. It requires only a single connection which allows automation and modeling iterations. The PA DUT is connected between MXG and MXA to measure PA Output. Obviously, this method is physically faster by eliminating one measurement compared to the first method. It is typical of industry practice today. It linearizes the entire system, not just the PA , and provides acceptable accuracy for quick Evaluation and MFG Test applications.



*Method 2 – Calculate PA Input, Measure PA Output*

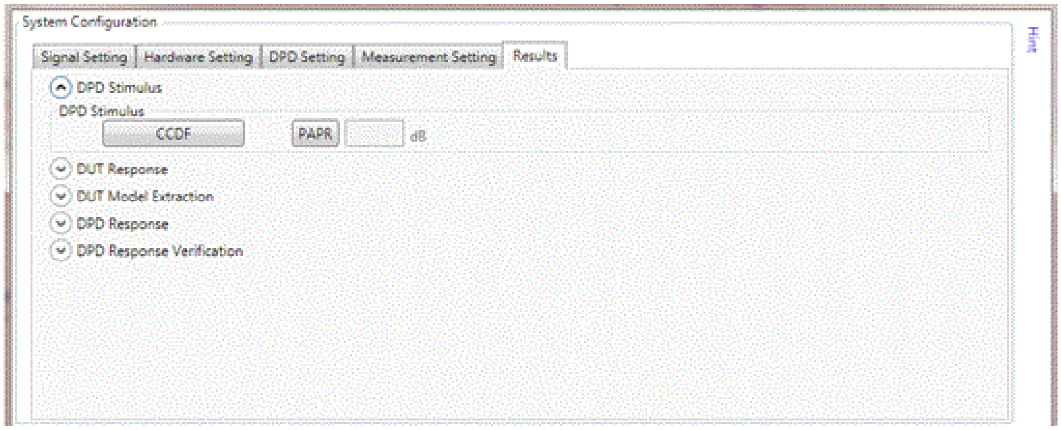
1. Create DPD Stimulus
2. Get baseband complex waveforms of PA input and output
3. Extract DPD Model (includes delay estimation and adjustment)
4. Apply DPD Model, and Get DPD+PA Response
5. Verify DPD Performance

Single connection allows automation, iterations
Eliminates one measurement, physically faster
Identical extraction algorithms, verification process

### 3. Linearize PA DUT

After setting up the hardware platform, please configure the parameters in DPD UI. In Signal Setting tab, configure System Parameters, Input Parameters and Clipping Parameters. In Hardware Setting tab, configure Download Parameters and Capture Parameters. In DPD Setting tab, configure DPD Setting. And in Measurement

Setting tab, configure Spectrum Measurement, EVM Measurement and ACP Measurement. Please click the corresponding links for more details.
After setting the parameters, please click Run button to start the measurement. The progress messages are displayed in the Progress Bar. Once finished, check the results in the Results tab, including DPD Stimulus Results, DUT Response Results, DUT Model Extraction Results, DPD Response Results and DPD Response Verification Results.



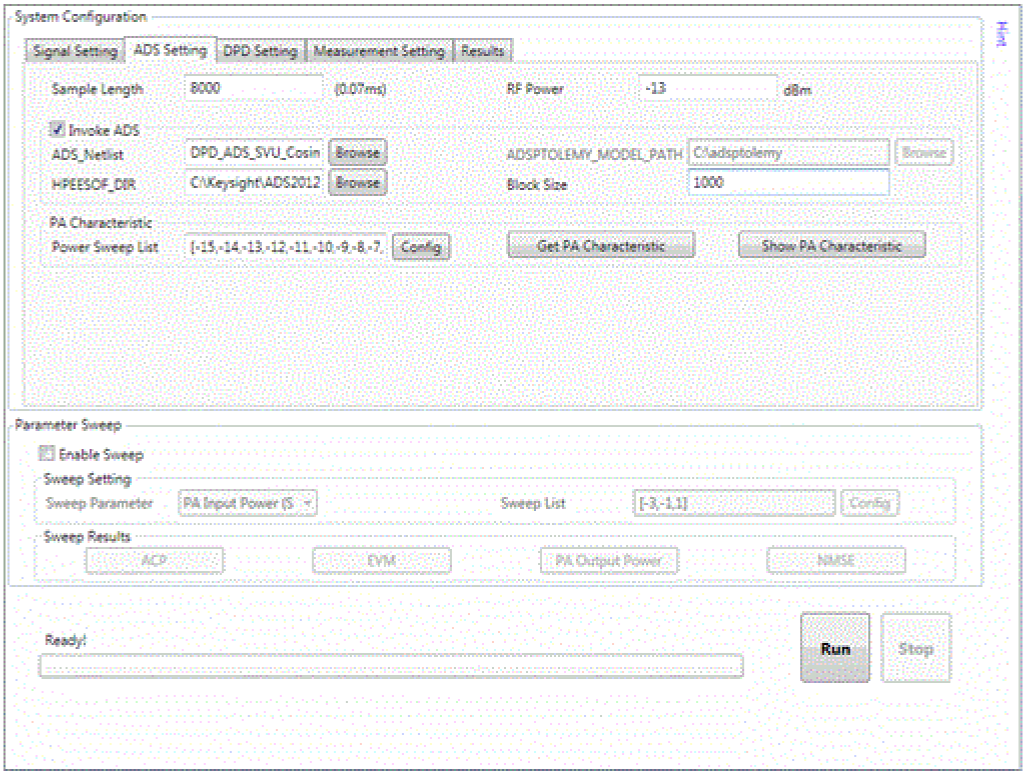
You can also Enable Sweep to sweep certain key parameters in DPD flow. After configuring the Parameter Sweep, you can click Run button to start the sweep. Once finished, you can check Sweep Results including spectrum, ACP, EVM and NMSE.

## DPD Cosimulation

Besides Hardware DPD Measurement, SystemVue also provides DPD Cosimulation with ADS, GoldenGate and X-Parameters. These three cosimulations use the same example workspace DPD_Cosimulation.wsv which can be found in $SystemVue\Examples\DPD\DPD Cosimulation, where $SystemVue stands for the SystemVue installation folder, typically C:\Program Files\SystemVue <Version_Number>.
The cosimulation flow is similar with Hardware DPD Measurement - Auto. However, it need not any hardware support.

### 1. ADS Cosimulation

To open DPD ADS Cosimulation UI, click Tools->Applications->DPD->Co-Sim->ADS Cosim. See Open DPD UI for Hardware DPD Measurement - Manual for more details. The ADS Cosim UI is shown below.



Sample Length 8000 (0.07ms)
RF Power -13 dBm

☑ Invoke ADS
ADS_Netlist DPD_ADS_SVU_Cosin
HPEESOF_DIR C:\Keysight\ADS2012
ADSPTOLEMY_MODEL_PATH C:\adsptolemy
Block Size 1000

PA Characteristic
Power Sweep List [-15,-14,-13,-12,-11,-10,-9,-8,-7,

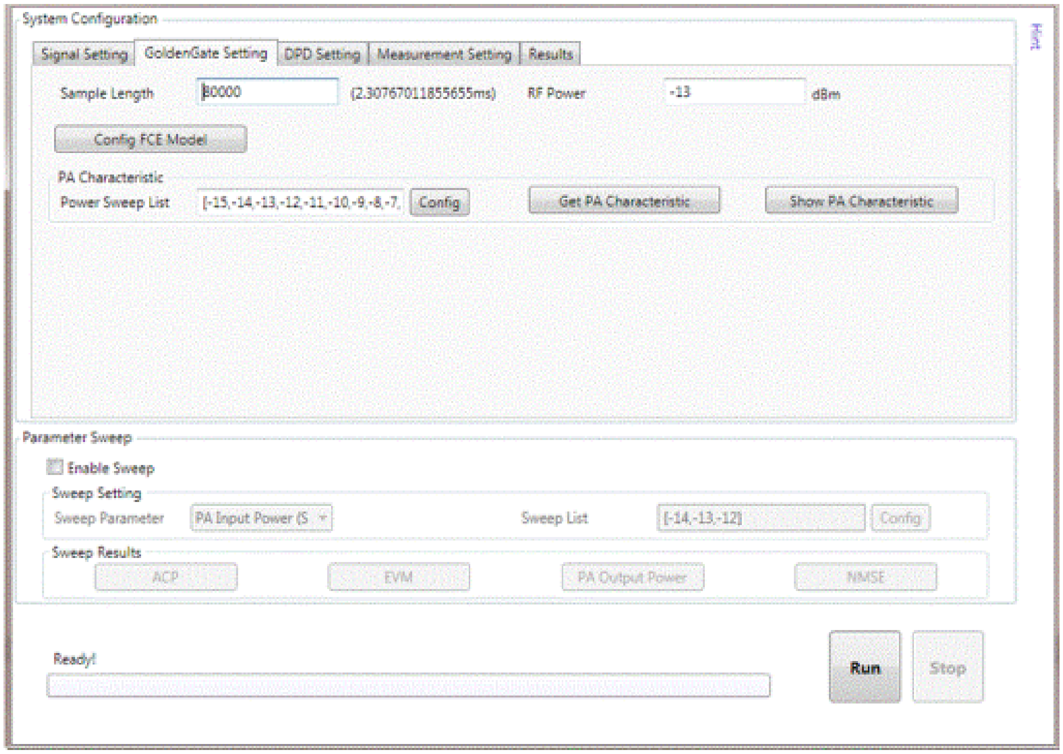
□ Enable Sweep
Sweep Parameter PA Input Power (S
Sweep List [-3,-1,1]

Ready!

Configure ADS Setting and the other parameters.
Get PA characteristics and determine the RF power to extract DPD model. Refer to PA Characteristic for more details.
Click Run button to start the simulation.

## 2. GoldenGate Cosimulation

To open DPD GoldenGate Cosimulation UI, click Tools->Applications->DPD->Co-Sim->GoldenGate Cosim. Refer to Open DPD UI for Hardware DPD Measurement - Manual for more details. The GoldenGate Cosim UI is shown below.



Sample Length 80000 (2.30767011855655ms) RF Power -13 dBm

PA Characteristic
Power Sweep List [-15,-14,-13,-12,-11,-10,-9,-8,-7,

☐ Enable Sweep

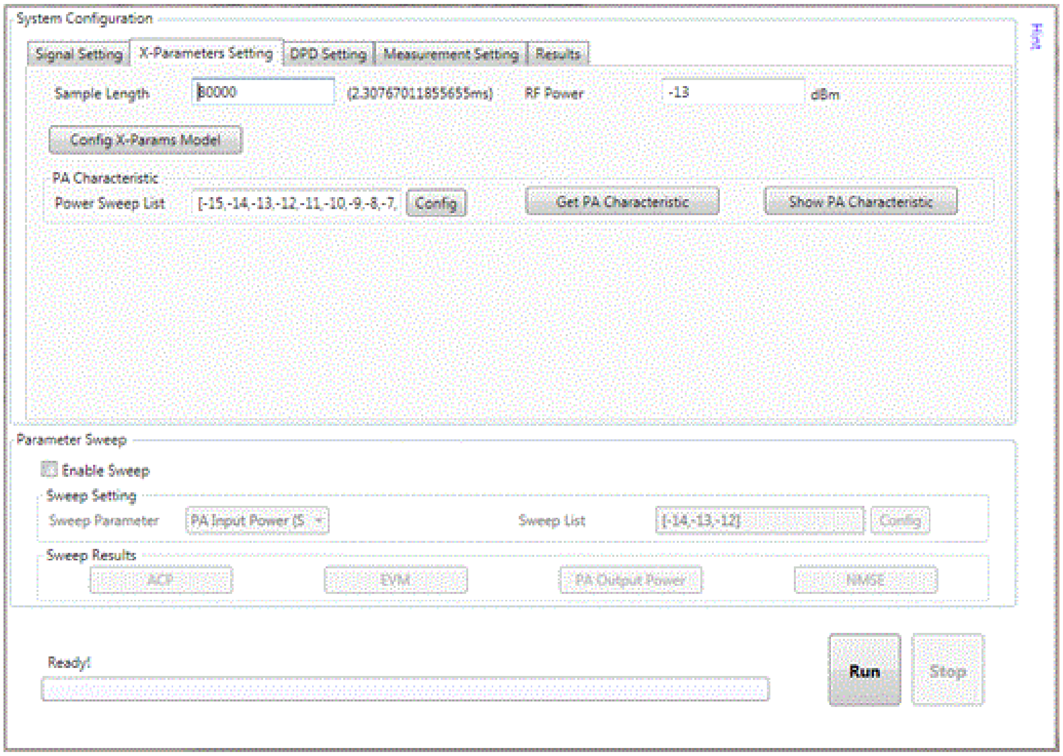
Sweep Setting
Sweep Parameter PA Input Power (S
Sweep List [-14,-13,-12]

Ready!

Configure GoldenGate Setting and the other parameters.
Get PA characteristics and determine the RF power to extract DPD model. Refer to PA Characteristic for more details.
Click Run button to start the simulation.

## 3. X-Parameters Cosimulation

To open DPD ADS Cosimulation UI, click Tools->Applications->DPD->Co-Sim->X-Parameter Cosim. Refer to Open DPD UI for Hardware DPD Measurement - Manual for more details. The X-Parameter Cosim UI is shown below.



Sample Length 80000 (2.30767011855655ms) RF Power -13 dBm

Power Sweep List [-15,-14,-13,-12,-11,-10,-9,-8,-7,

□ Enable Sweep

Sweep Parameter PA Input Power (S

Sweep List [-14,-13,-12]

Ready!

Configure X-Parameters Setting and the other parameters.

Get PA characteristics and determine the RF power to extract DPD model. Refer to PA Characteristic for more details.

Click Run button to start the simulation.

**NOTE** You can also Enable Sweep to sweep certain key parameters in DPD cosimulation flow. After configuring the Parameter Sweep, you can click Run button to start the sweep. Once finished, you can check Sweep Results including spectrum, ACP, EVM and NMSE.

# RF Design

- Getting Started with Spectrasys
- Embedding Spectrasys in Data Flow using RF_Link

## Getting Started with Spectrasys

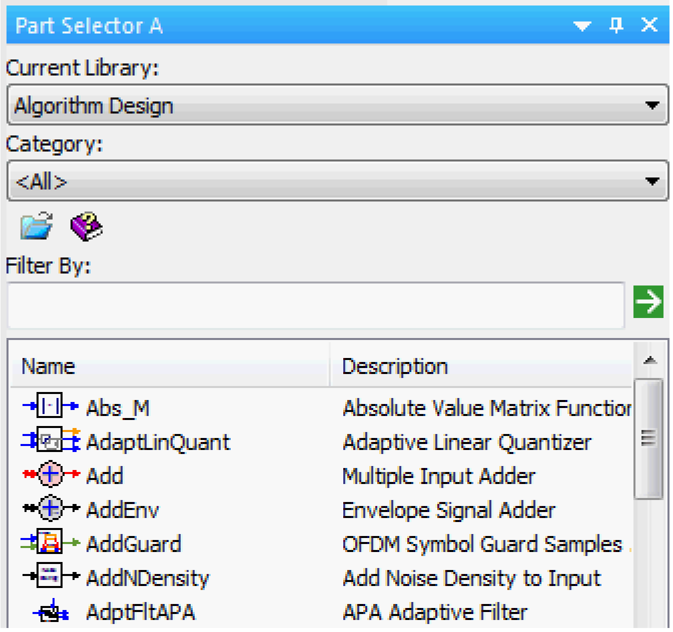
Spectrasys uses a new simulation technique called SPARCA that brings RF architecture design to a whole new level. This walkthrough helps you design a simple RF chain and measure the architecture's noise and gain performance.

The basic steps for analyzing an RF system are:

1. Create a System Schematic
2. Adding a System Analysis
3. Run the Simulation
4. Add a Graph or Table

### Create a System Schematic

Spectrasys supports all linear models and behavioral non-linear models. The behavioral models can be found on the system toolbar or in the part selector.



Part Selector A

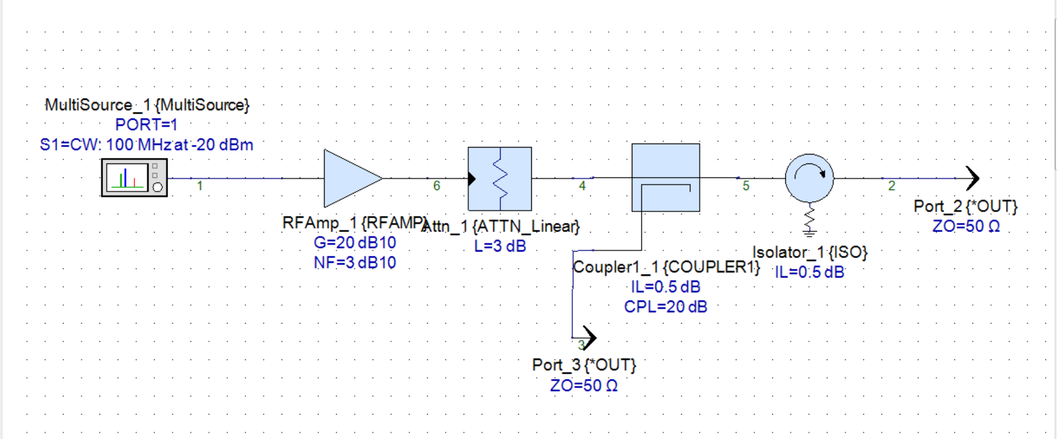
Current Library: Algorithm Design

Category: <All>

Filter By:

| Name | Description |
|---|---|
| Abs_M | Absolute Value Matrix Functior |
| AdaptLinQuant | Adaptive Linear Quantizer |
| Add | Multiple Input Adder |
| AddEnv | Envelope Signal Adder |
| AddGuard | OFDM Symbol Guard Samples |
| AddNDensity | Add Noise Density to Input |
| AdptFltAPA | APA Adaptive Filter |

Create the following system schematic (default parameters for all models will be used). For additional help, creating a schematic click here.



1. Select the 'Amp (2nd & 3rd Order)' from the system toolbar or part selector.
2. Move the cursor and click inside the schematic window to place the part.
3. Use the prior steps to place a fixed Attenuator, Coupler (Single Dir), and Isolator.
4. Place a Source (Multi) at the input. Now add a carrier by double-clicking the source and clicking the Add button. A source user interface will appear. Change the power level to -20 dBm.
5. Place an Output Port on the output of the isolator and the coupler.

   NOTE Press the " ***O*** key on the keyboard to place an output port.

6. Make sure each part output is wired to the subsequent part input.

   NOTE Use the '**F4**' key when a part is highlighted to repeatedly move the part text to default locations around the part.

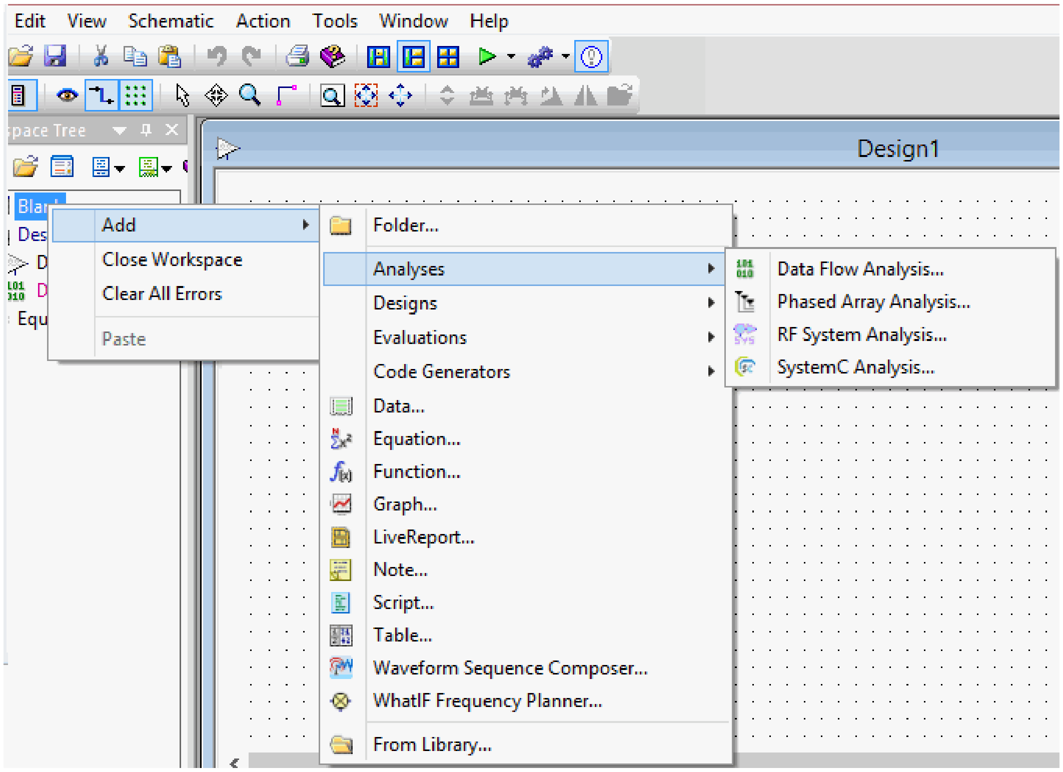
7. Select the desired options and click OK.

## Adding a System Analysis

After creating a schematic a system analysis must be created. There a several ways to accomplish this. Only one way will be shown here. For additional information on adding analyses click here.

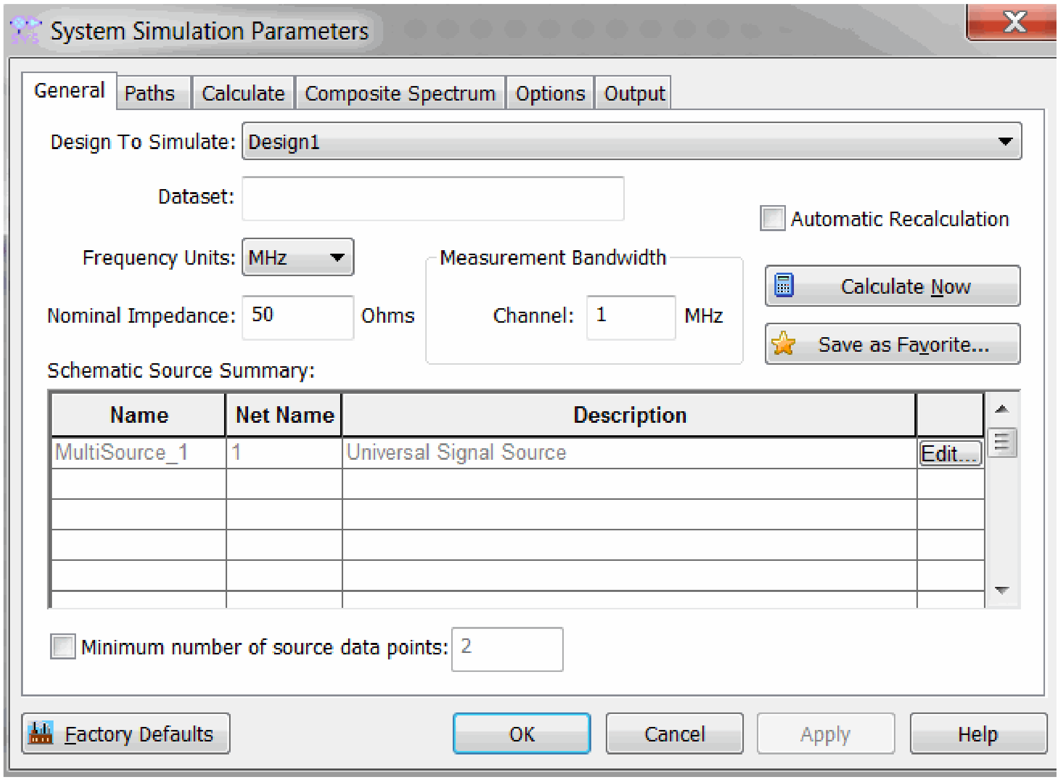
To add a system analysis:

Right-click a folder in the workspace tree where you want the analysis located.



Select RF System Analysis... from the selected sub menus as shown above.

The following 'System Analysis Parameter' dialog box will appear.



System Simulation Parameters

General | Paths | Calculate | Composite Spectrum | Options | Output

Design To Simulate: Design1

Dataset:

□ Automatic Recalculation

Frequency Units: MHz

Measurement Bandwidth

Channel: 1 MHz

Nominal Impedance: 50 Ohms

Calculate Now

Save as Favorite...

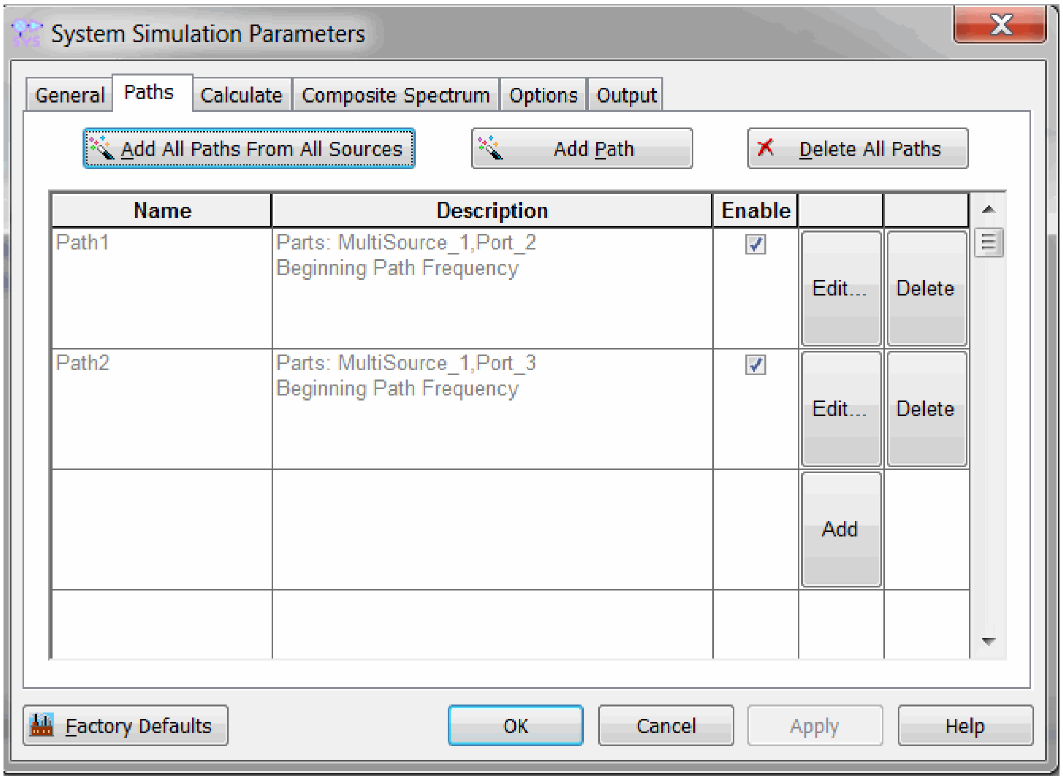
Schematic Source Summary:

| Name | Net Name | Description | |
|---|---|---|---|
| MultiSource_1 | 1 | Universal Signal Source | Edit... |

□ Minimum number of source data points: 2

Factory Defaults | OK | Cancel | Apply | Help

If path measurements are desired \(i.e. cascaded gain or cascaded noise figure\) click on the Paths tab.



| Name | Description | Enable |
|---|---|---|
| Path1 | Parts: MultiSource_1,Port_2<br>Beginning Path Frequency | ☑ |
| Path2 | Parts: MultiSource_1,Port_3<br>Beginning Path Frequency | ☑ |

Click the Add All Paths From All Sources button.

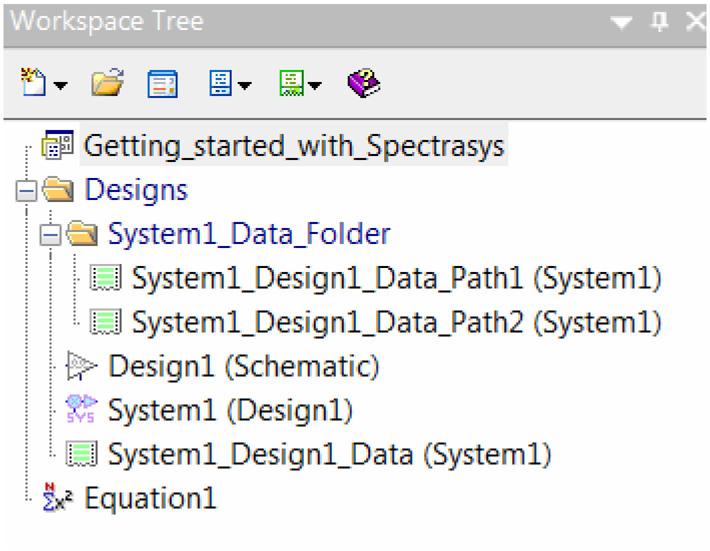
NOTE Node numbers may be different than shown above depending on the node numbers in your schematic. For more information, see specifying paths.

Click the dialog OK button.

## Run the Simulation

Analysis data must be created before it can be plotted or displayed in tables. The analysis can be enabled to 'Automatically Recalculate' or may need to be manually calculated. If the analysis has been set to 'Automatically Recalculate' datasets will appear in the workspace tree after the analysis. If the manual calculation is needed the calculate button will appear red and so will other items in the workspace tree. Click the calculate button to update the system analysis and create the necessary datasets.

After calculation the workspace tree should look like:



Getting_started_with_Spectrasys
Designs
System1_Data_Folder
System1_Design1_Data_Path1 (System1)
System1_Design1_Data_Path2 (System1)
Design1 (Schematic)
System1 (Design1)
System1_Design1_Data (System1)
Equation1

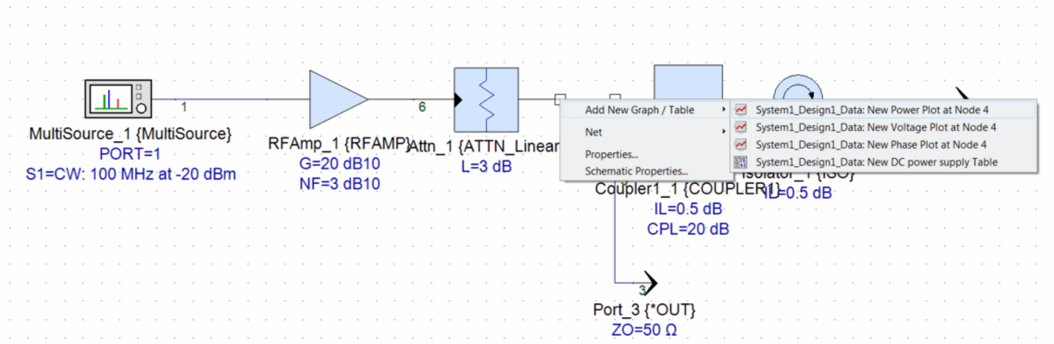
For more information, see datasets.

## Add a Graph or Table

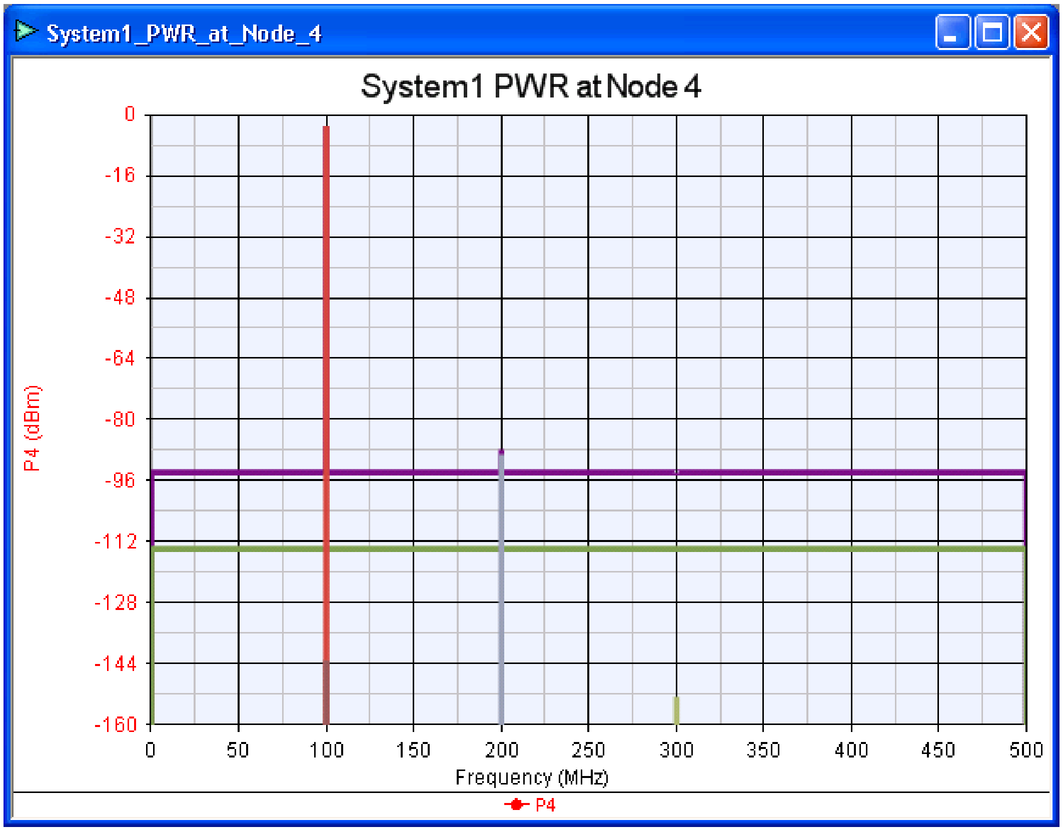
There are several ways to display data in
SystemVue

. Only one way will be demonstrated here. For additional information on graphs, click here.

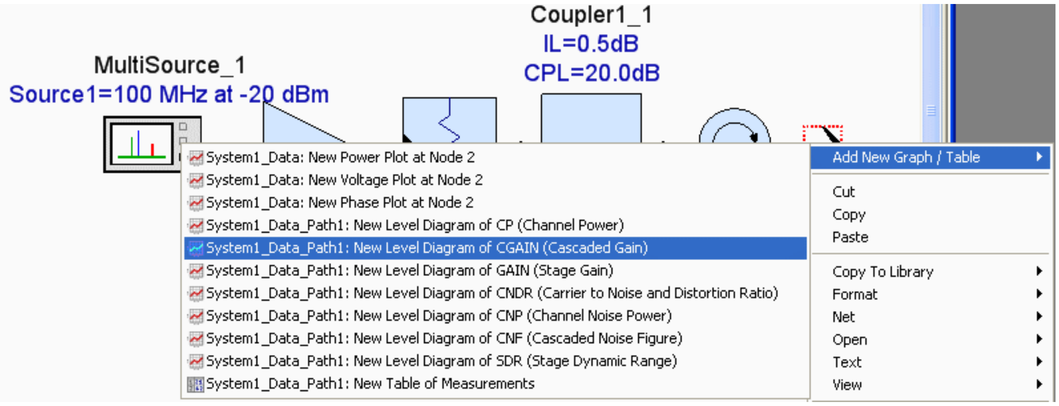
The easiest way to add a spectral power, phase, or voltage plot in Spectrasys is by right clicking the node to be viewed and selecting 'System1_Data: New Power Plot at Node x' from the submenu 'Add New Graph/Table'. (The output of the attenuator was selected in the following figure)


MultiSource_1 {MultiSource}
PORT=1
S1=CW: 100 MHz at -20 dBm
1
RFAmp_1 {RFAMP}
G=20 dB10
NF=3 dB10
6
Attn_1 {ATTN_Linear
L=3 dB
Coupler1_1 {COUPLER1}
IL=0.5 dB
CPL=20 dB
Isolator_1 {ISO}
IL=0.5 dB
3
Port_3 {*OUT}
ZO=50 Ω
Add New Graph / Table
Net
Properties...
Schematic Properties...
System1_Design1_Data: New Power Plot at Node 4
System1_Design1_Data: New Voltage Plot at Node 4
System1_Design1_Data: New Phase Plot at Node 4
System1_Design1_Data: New DC power supply Table

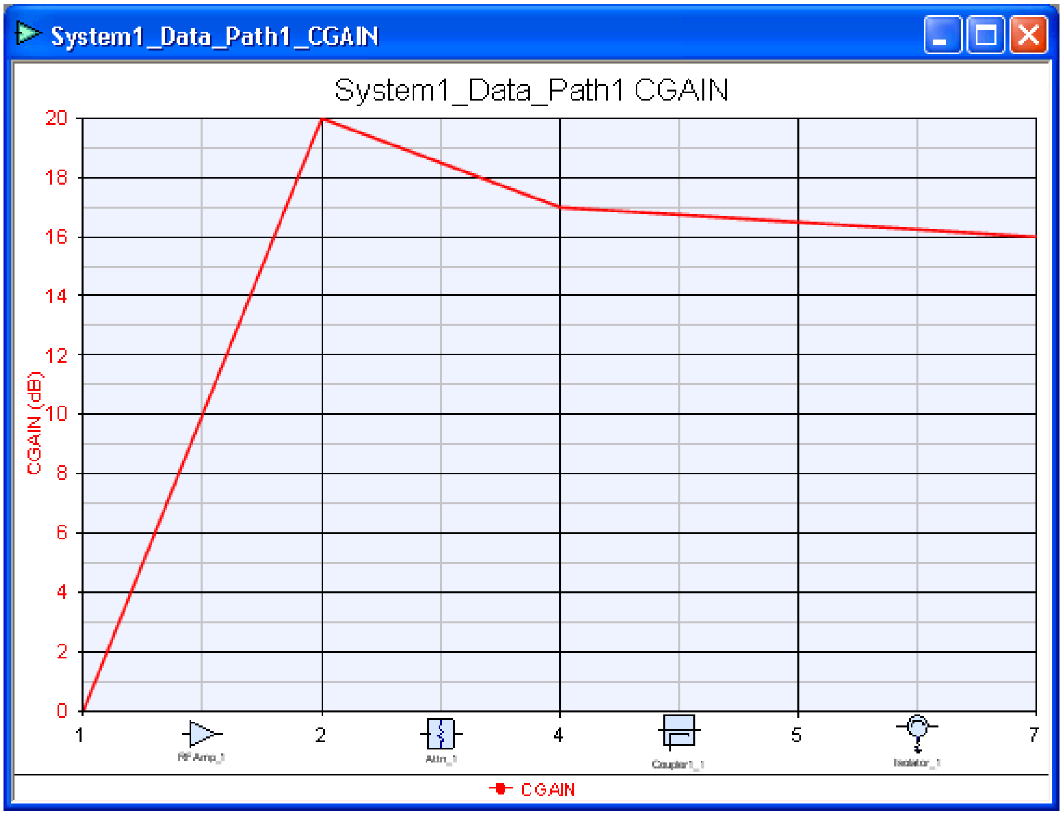
The following graph will appear:



To add a level diagram (a path number be defined first) right click on the ending node of the path and selecting 'System1_Data_Path1: New Level Diagram of CGAIN (Cascaded Gain)' from the 'Add New Graph / Table' submenu.

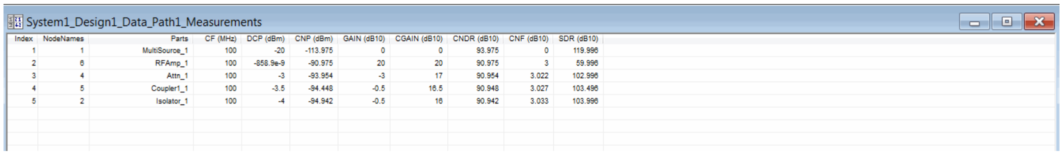

Coupler1_1
IL=0.5dB
CPL=20.0dB
MultiSource_1
Source1=100 MHz at -20 dBm
System1_Data: New Power Plot at Node 2
System1_Data: New Voltage Plot at Node 2
System1_Data: New Phase Plot at Node 2
System1_Data_Path1: New Level Diagram of CP (Channel Power)
System1_Data_Path1: New Level Diagram of CGAIN (Cascaded Gain)
System1_Data_Path1: New Level Diagram of GAIN (Stage Gain)
System1_Data_Path1: New Level Diagram of CNDR (Carrier to Noise and Distortion Ratio)
System1_Data_Path1: New Level Diagram of CNP (Channel Noise Power)
System1_Data_Path1: New Level Diagram of CNF (Cascaded Noise Figure)
System1_Data_Path1: New Level Diagram of SDR (Stage Dynamic Range)
System1_Data_Path1: New Table of Measurements
Add New Graph / Table
Cut
Copy
Paste
Copy To Library
Format
Net
Open
Text
View

The following level diagram will appear:



Follow the same process as adding a level diagram to add a predefined table of common measurements except select 'System1_Data_Path1: New Table of Measurements' from the 'Add New Graph / Table' submenu. For additional path measurement information click here.

The default table will look like:


System1_Design1_Data_Path1_Measurements

| Index | NodeNames | Parts | CF (MHz) | DCP (dBm) | CNP (dBm) | GAIN (dB10) | CGAIN (dB10) | CNDR (dB10) | CNF (dB10) | SDR (dB10) |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | MultiSource_1 | 100 | -20 | -113.975 | 0 | 0 | 93.975 | 0 | 119.996 |
| 2 | 6 | RFAmp_1 | 100 | -858.9e-9 | -90.975 | 20 | 20 | 90.975 | 3 | 59.996 |
| 3 | 4 | Attn_1 | 100 | -3 | -93.954 | -3 | 17 | 90.954 | 3.022 | 102.996 |
| 4 | 5 | Coupler1_1 | 100 | -3.5 | -94.448 | -0.5 | 16.5 | 90.948 | 3.027 | 103.496 |
| 5 | 2 | Isolator_1 | 100 | -4 | -94.942 | -0.5 | 16 | 90.942 | 3.033 | 103.996 |

**NOTE** Right-click on the table data to see additional table options.

# Embedding Spectrasys in Data Flow using RF_Link

## Embedding Spectrasys in Data Flow using RF_Link

The examples in this tutorial section show how to use RF designs created in Spectrasys in a Data Flow simulation. Before such a design is used in a Data Flow simulation it should simulate without errors with Spectrasys. Embedding a Spectrasys RF design in a Data Flow simulation is done through the use of RF_Link. For more information about the theory of operation, limitations, etc. see RF/Data Flow Cosimulation. For more examples look in the directory <SystemVue Installation Directory>\Examples\RF Architecture Design.

1. Simple TX RX

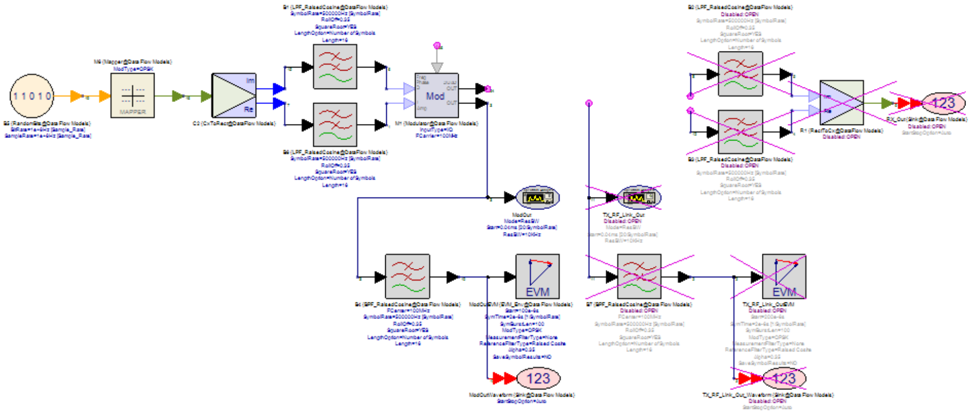
## Simple TX RX

This tutorial example shows how to use a simple Spectrasys transmitter and receiver in a Data Flow simulation.

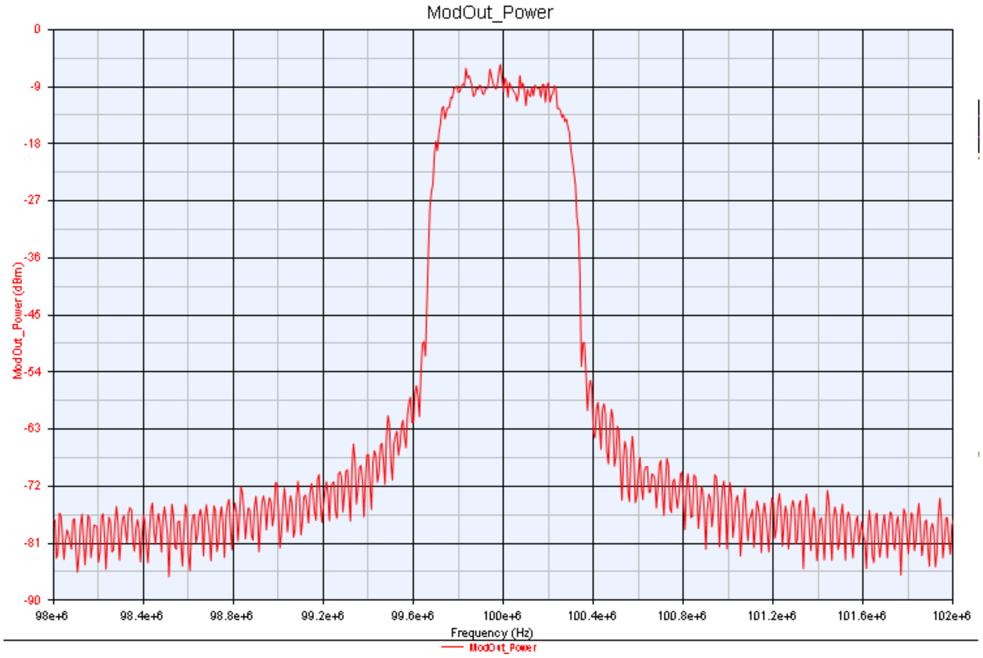
**CAUTION** Basic familiarity with Spectrasys is assumed.

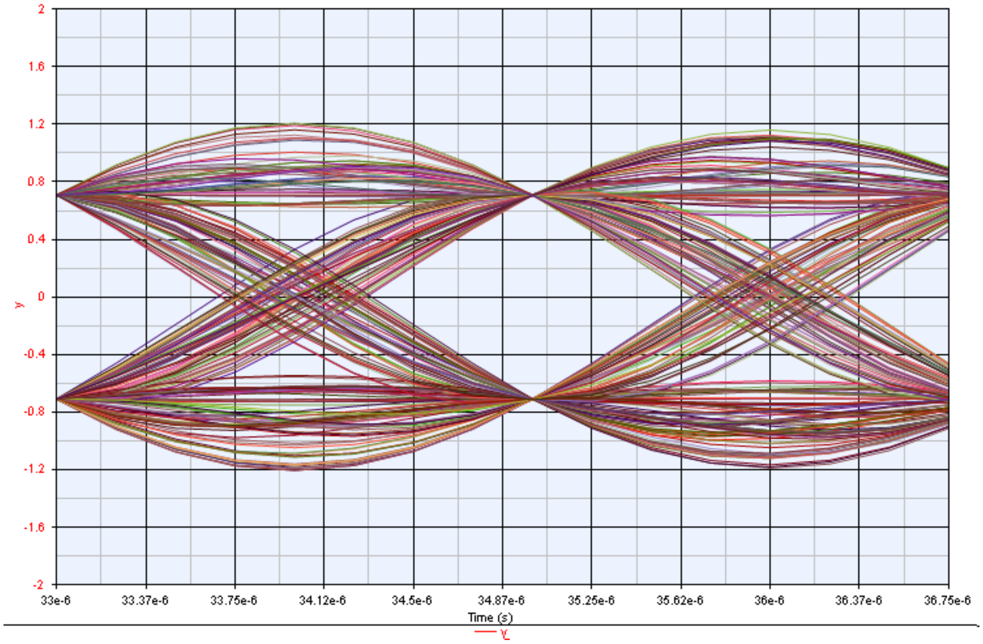
**CAUTION** During this tutorial example, you may see a lot of errors coming from graphs. This is because the graphs are pre-configured to show data that will be created later on in the tutorial.

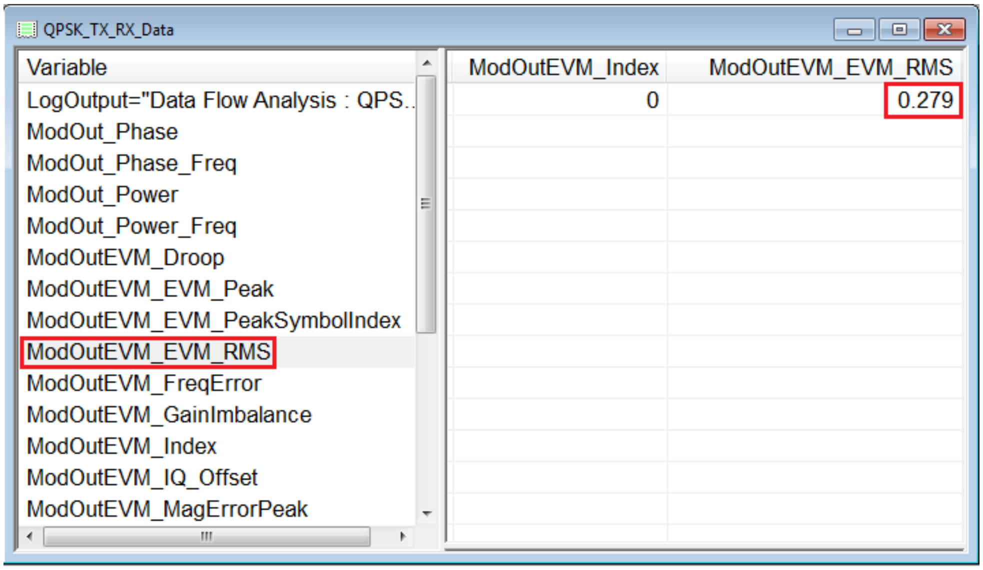
1. Open the example <SystemVue Installation Directory>\Examples\Tutorials\RF_Design\Simple_TX_RX.wsv.
2. Open the design *QPSK_TX_RX* in the *Designs* folder. This design generates a QPSK signal using Data Flow parts. The symbol rate is 0.5 MHz and the modulator output frequency is 100 MHz. Our goal is to create an RF transmitter that will amplify the signal and upconvert it to 2.1 GHz and a receiver that will demodulate the signal.



3. Run *QPSK_TX_RX Analysis* and observe the spectrum (*QPSK_TX_RX_ModOut* graph), eye diagram (*QPSK_TX_RX_Eye* graph), and EVM (*QPSK_TX_RX_Data* dataset) at the output of the modulator.


ModOut_Power
0
-9
-18
-27
-36
-45
-54
-63
-72
-81
-90
ModOut_Power (dBm)
98e+6
98.4e+6
98.8e+6
99.2e+6
99.6e+6
100e+6
100.4e+6
100.8e+6
101.2e+6
101.6e+6
102e+6
Frequency (Hz)
ModOut_Power


2
1.6
1.2
0.8
0.4
0
-0.4
-0.8
-1.2
-1.6
-2
y
33e-6
33.37e-6
33.75e-6
34.12e-6
34.5e-6
34.87e-6
35.25e-6
35.62e-6
36e-6
36.37e-6
36.75e-6
Time (s)
y

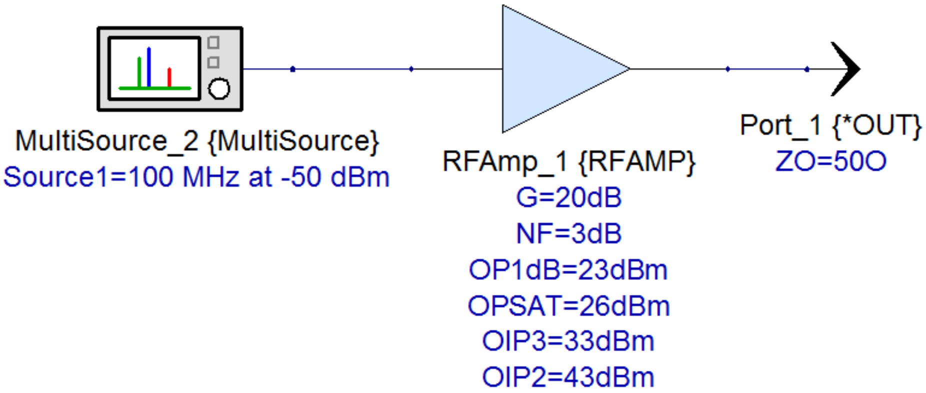


QPSK_TX_RX_Data

Variable
LogOutput="Data Flow Analysis : QPS..
ModOut_Phase
ModOut_Phase_Freq
ModOut_Power
ModOut_Power_Freq
ModOutEVM_Droop
ModOutEVM_EVM_Peak
ModOutEVM_EVM_PeakSymbolIndex
ModOutEVM_EVM_RMS
ModOutEVM_FreqError
ModOutEVM_GainImbalance
ModOutEVM_Index
ModOutEVM_IQ_Offset
ModOutEVM_MagErrorPeak

| ModOutEVM_Index | ModOutEVM_EVM_RMS |
|---|---|
| 0 | 0.279 |

4. Create a new design (right click on the *Designs* folder and select *Add* > *Designs* > *Add Sub-Network Model...*) and call it *TX_Design*. Place (from the RF_Design part library) a MultiSource, an RFAMP, and an output port. Set the Multisource to generate a CW signal at 100 MHz (same as our modulator output frequency) and -50 dBm. Set the amplifier parameters as shown in the schematic below.



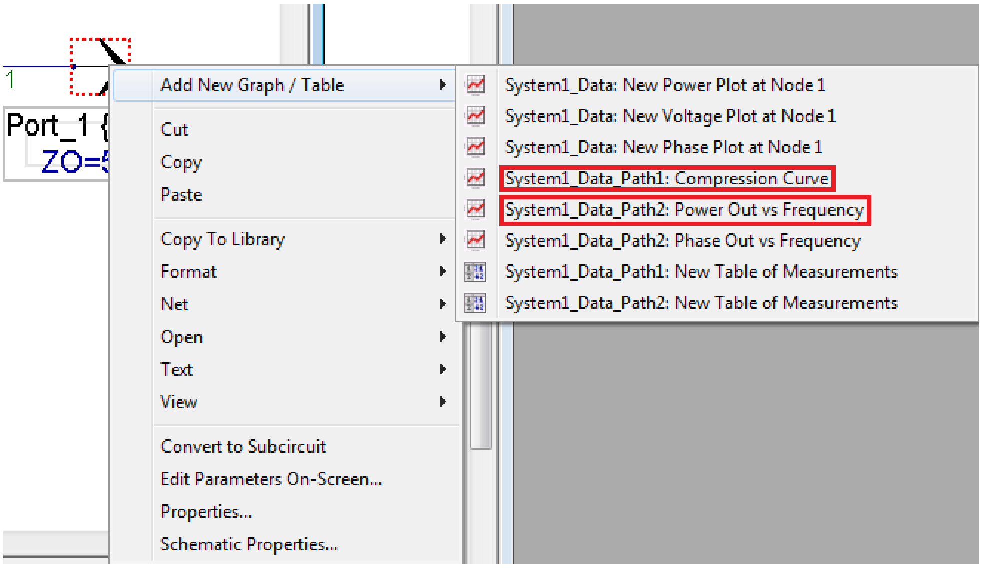
MultiSource_2 {MultiSource}
Source1=100 MHz at -50 dBm

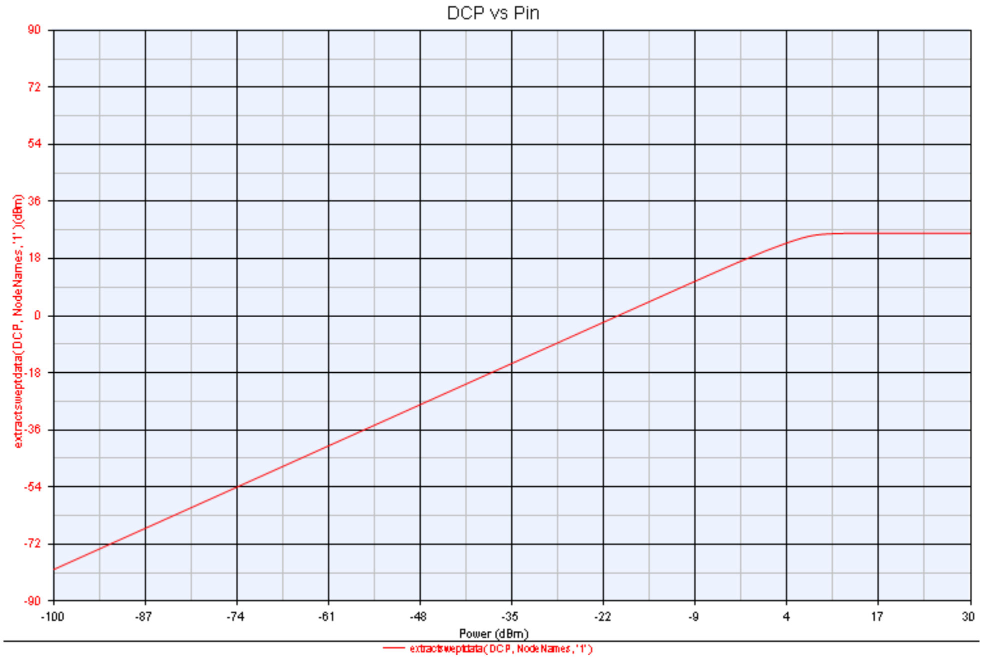
RFAmp_1 {RFAMP}
G=20dB
NF=3dB
OP1dB=23dBm
OPSAT=26dBm
OIP3=33dBm
OIP2=43dBm

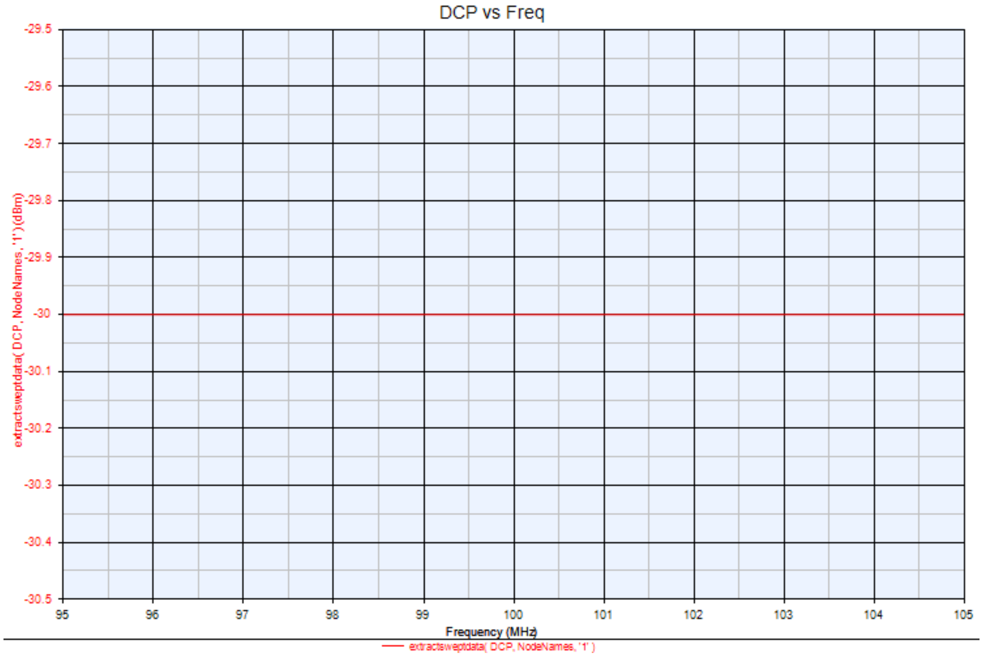
Port_1 {*OUT}
ZO=50O

5. Add a System analysis (right click on the folder *Analyses* and select *Add* > *Analyses* > *Add RF System Analysis...*). In the *General* tab set Design To Simulate to *TX_Design*, and in the *Paths* tab create two *Quick Sweep* paths (one for *Compression Curve* and one for *Frequency Response*).

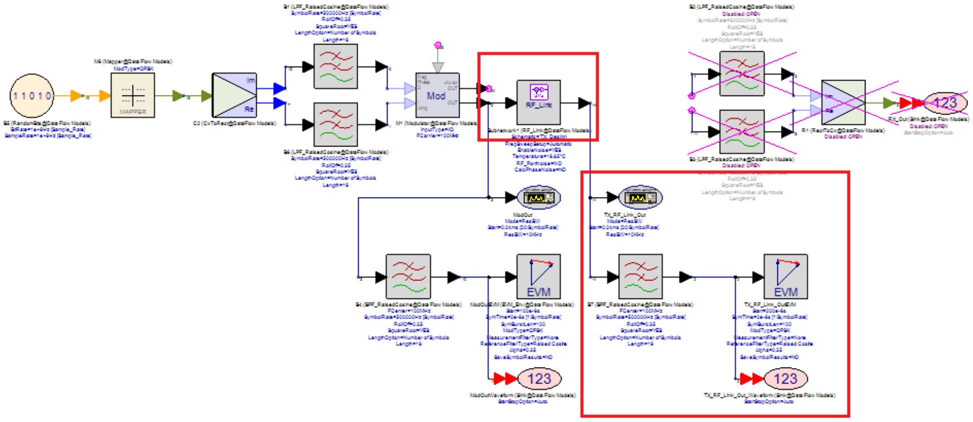
6. Run *System1* analysis. Right click on the output port and select the *Compression Curve* and *Power Out vs Frequency* to create compression curve and output power vs frequency graphs.


1
Port_1 {
ZO=5
Add New Graph / Table
Cut
Copy
Paste
Copy To Library
Format
Net
Open
Text
View
Convert to Subcircuit
Edit Parameters On-Screen...
Properties...
Schematic Properties...
System1_Data: New Power Plot at Node 1
System1_Data: New Voltage Plot at Node 1
System1_Data: New Phase Plot at Node 1
System1_Data_Path1: Compression Curve
System1_Data_Path2: Power Out vs Frequency
System1_Data_Path2: Phase Out vs Frequency
System1_Data_Path1: New Table of Measurements
System1_Data_Path2: New Table of Measurements

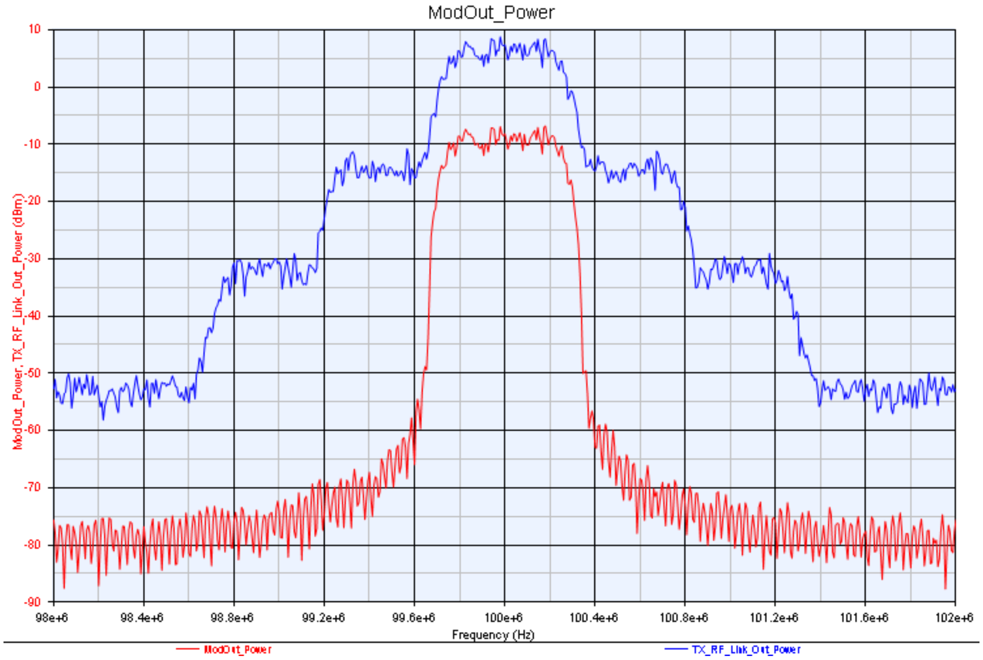

DCP vs Pin
90
72
54
36
18
0
-18
-36
-54
-72
-90
extractsweptdata(DCP, NodeNames,'1')(dBm)
-100
-87
-74
-61
-48
-35
-22
-9
4
17
30
Power (dBm)
extractsweptdata(DCP, NodeNames,'1')

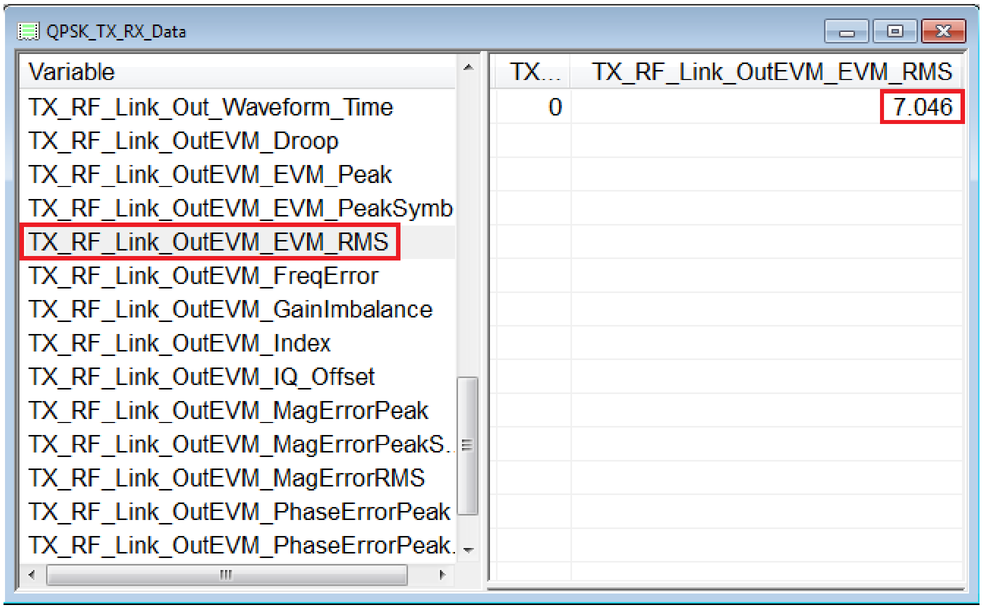


7. Now drag *TX_Design* from the workspace tree on the *QPSK_TX_RX* design. An RF_Link part is automatically instantiated. Connect it at the output of the modulator. Double-click on the RF_Link part and check the Calculate Thermal Noise check-box and uncheck the Add source thermal noise to input checkbox. Activate the parts connected to the output of RF_Link.



8. Run *QPSK_TX_RX Analysis* again and observe the spectrum ( *QPSK_TX_RX_ModOut* graph), eye diagram (*QPSK_TX_RX_Eye2* graph), and EVM (*QPSK_TX_RX_Data* dataset) at the output of the modulator. Make sure you unhide the second series in the *QPSK_TX_RX_ModOut* graph. You can clearly see the spectral regrowth caused by the non-linear amplifier. EVM also increased to 7%.

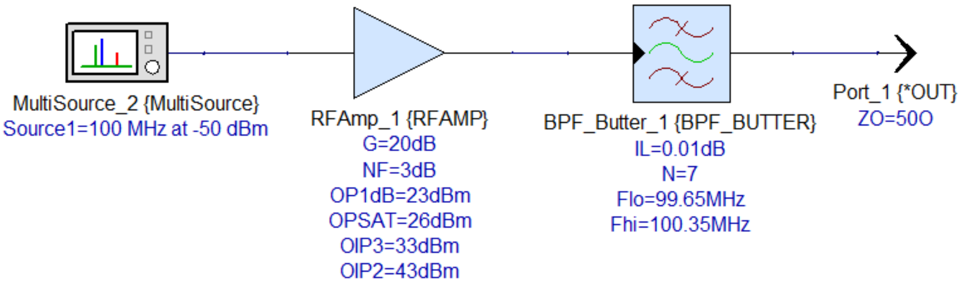




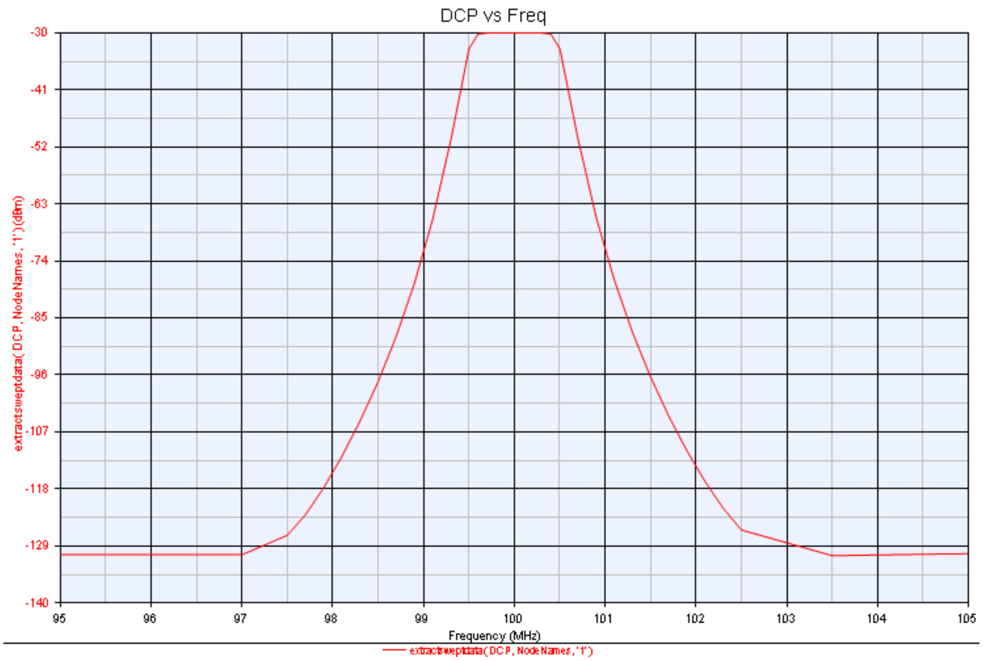
Variable
TX_RF_Link_Out_Waveform_Time
TX_RF_Link_OutEVM_Droop
TX_RF_Link_OutEVM_EVM_Peak
TX_RF_Link_OutEVM_EVM_PeakSymb
TX_RF_Link_OutEVM_EVM_RMS
TX_RF_Link_OutEVM_FreqError
TX_RF_Link_OutEVM_GainImbalance
TX_RF_Link_OutEVM_Index
TX_RF_Link_OutEVM_IQ_Offset
TX_RF_Link_OutEVM_MagErrorPeak
TX_RF_Link_OutEVM_MagErrorPeakS.
TX_RF_Link_OutEVM_MagErrorRMS
TX_RF_Link_OutEVM_PhaseErrorPeak
TX_RF_Link_OutEVM_PhaseErrorPeak.

| TX... | TX_RF_Link_OutEVM_EVM_RMS |
|---|---|
| 0 | 7.046 |

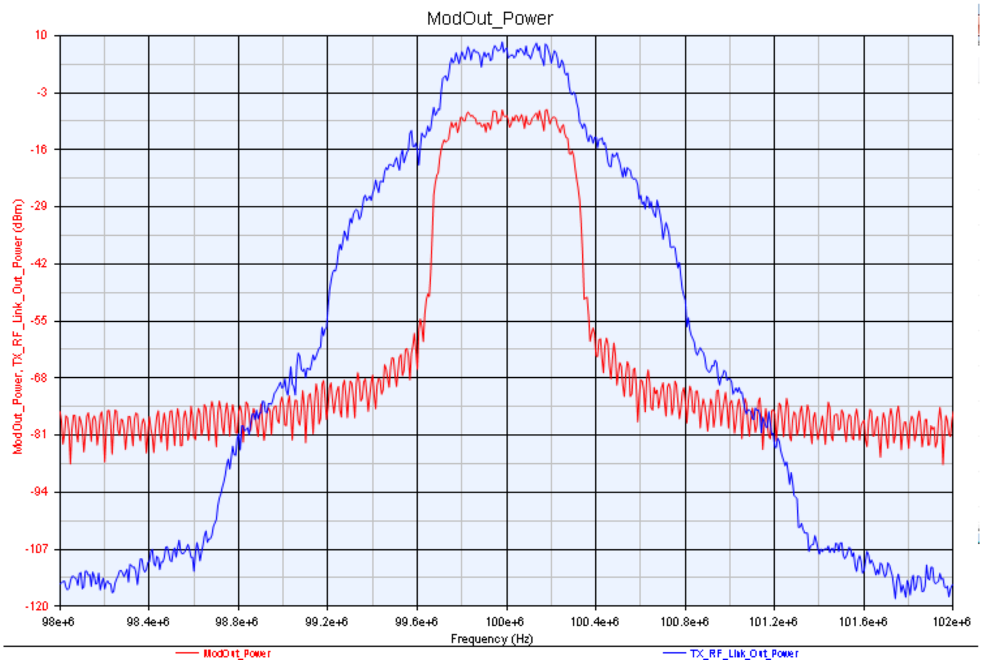
9. Open *TX_Design* and add a filter at the output of the amplifier to reduce the spectral regrowth.

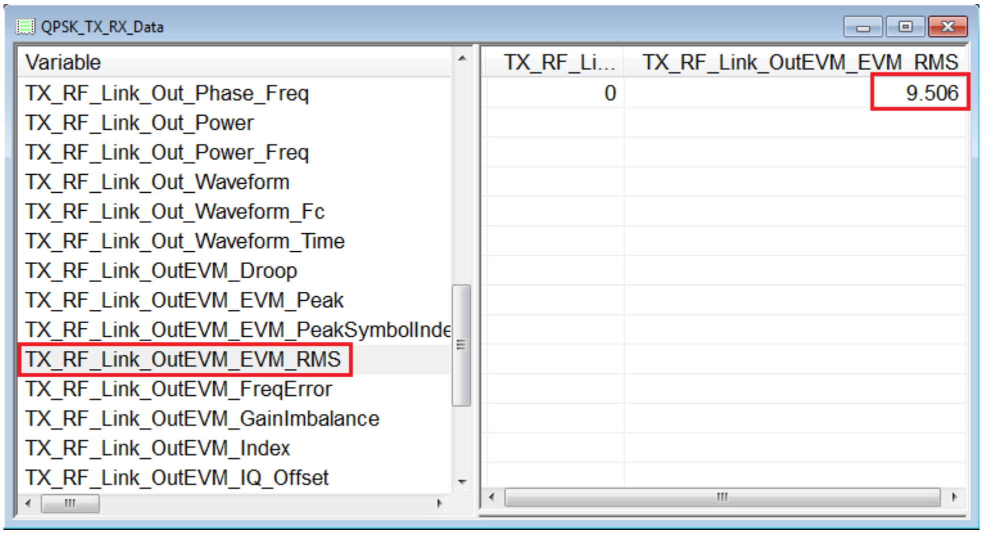


10. Run *System1* analysis and observe the compression curve and output power vs frequency graphs. The power vs frequency is no longer flat.



11. Run *QPSK_TX_RX Analysis* again and observe the spectrum ( *QPSK_TX_RX_ModOut* graph), eye diagram (*QPSK_TX_RX_Eye2* graph), and EVM (*QPSK_TX_RX_Data* dataset) at the output of the modulator. The spectral regrowth caused by the non-linear amplifier has been reduced but the EVM increased to 9.5%.



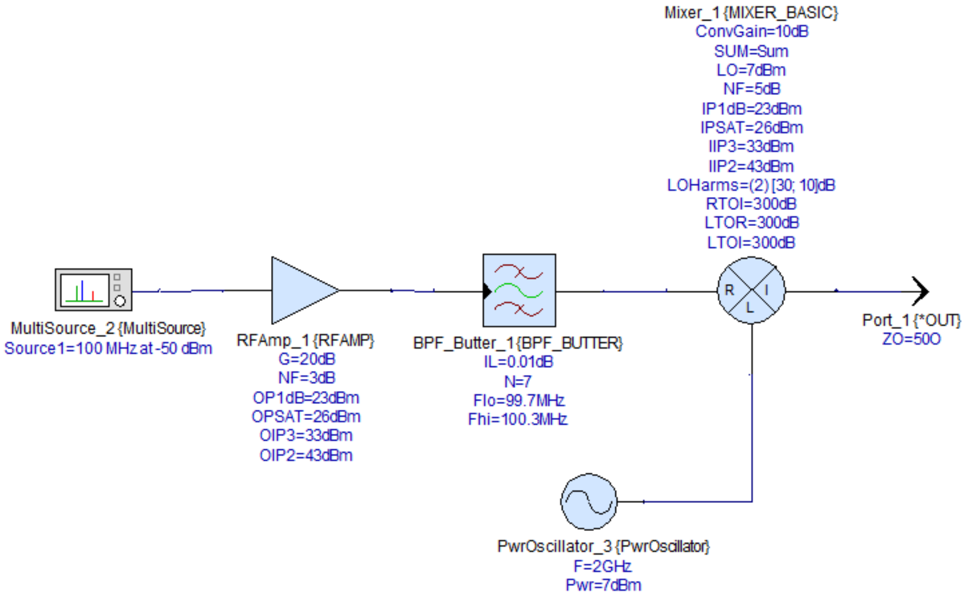


QPSK_TX_RX_Data

| Variable |
|---|
| TX_RF_Link_Out_Phase_Freq |
| TX_RF_Link_Out_Power |
| TX_RF_Link_Out_Power_Freq |
| TX_RF_Link_Out_Waveform |
| TX_RF_Link_Out_Waveform_Fc |
| TX_RF_Link_Out_Waveform_Time |
| TX_RF_Link_OutEVM_Droop |
| TX_RF_Link_OutEVM_EVM_Peak |
| TX_RF_Link_OutEVM_EVM_PeakSymbolInde |
| TX_RF_Link_OutEVM_EVM_RMS |
| TX_RF_Link_OutEVM_FreqError |
| TX_RF_Link_OutEVM_GainImbalance |
| TX_RF_Link_OutEVM_Index |
| TX_RF_Link_OutEVM_IQ_Offset |

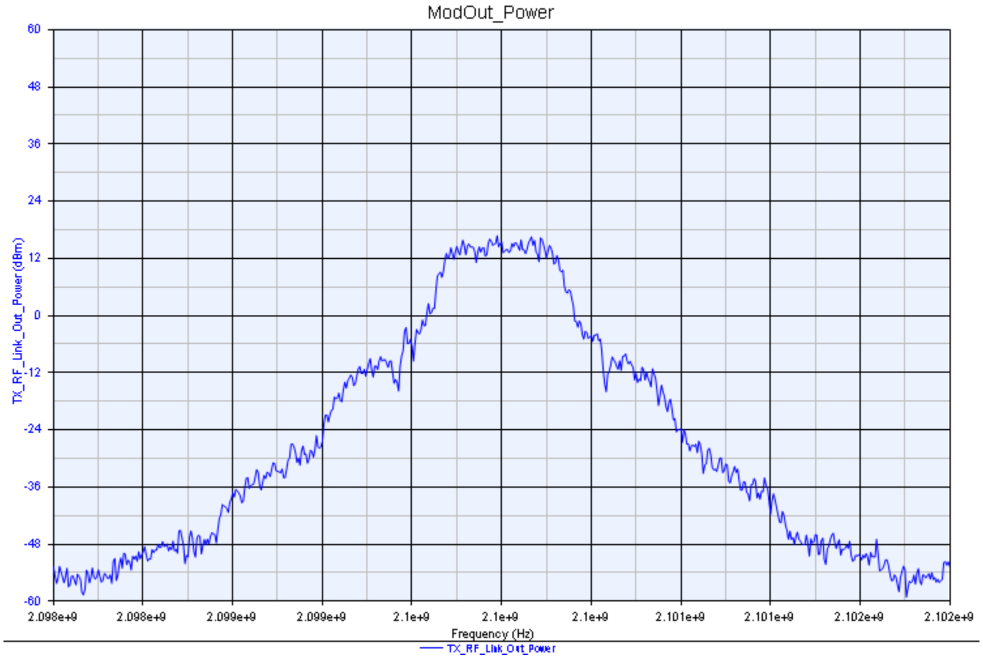
| TX_RF_Li... | TX_RF_Link_OutEVM_EVM_RMS |
|---|---|
| 0 | 9.506 |

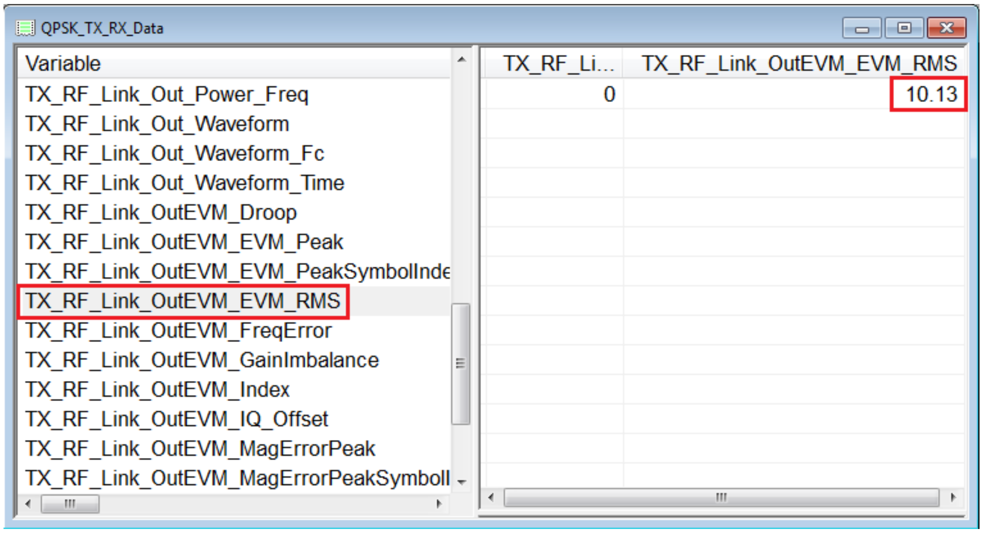
12. Finally, let's add a mixer to the *TX_Design* to upconvert the signal to 2.1 GHz.



13. Run *System1* analysis and observe the compression curve and output power vs frequency graphs.

14. In *QPSK_TX_RX* change *FCenter* of the filter *B7* (at the output of RF_Link) to 2100 MHZ and run *QPSK_TX_RX Analysis* again. Observe the spectrum ( *QPSK_TX_RX_ModOut* graph), eye diagram (*QPSK_TX_RX_Eye2* graph), and EVM (*QPSK_TX_RX_Data* dataset) at the output of the modulator. Since the two spectra plotted in the *QPSK_TX_RX_ModOut* graph are centered at different frequencies, hide the *ModOut_Power* series to be able to see the spectrum at the output of RF_Link. EVM is now over 10%.

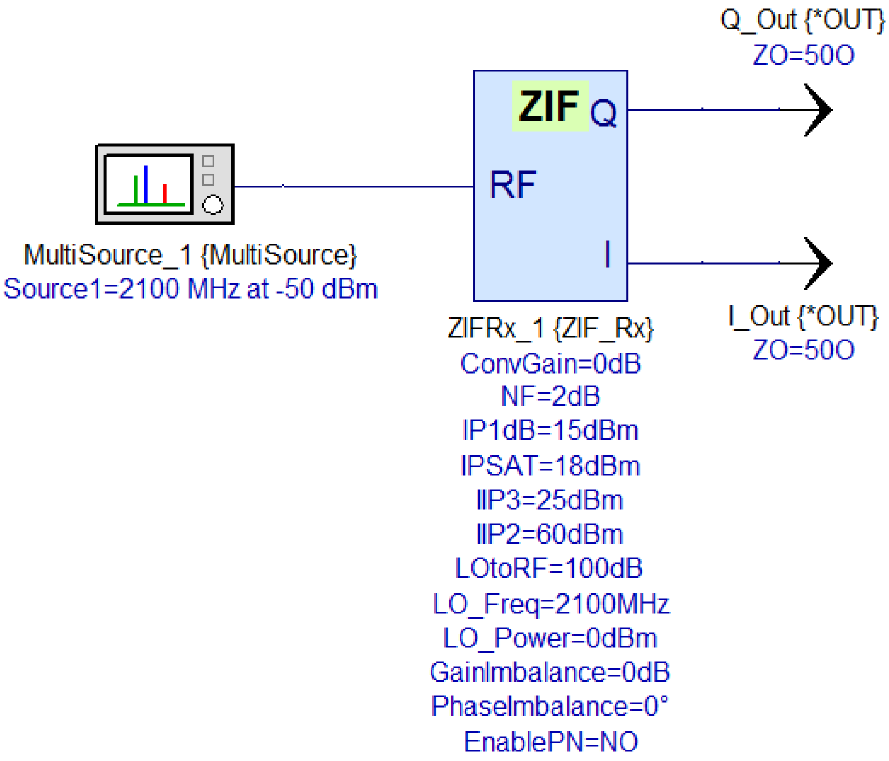




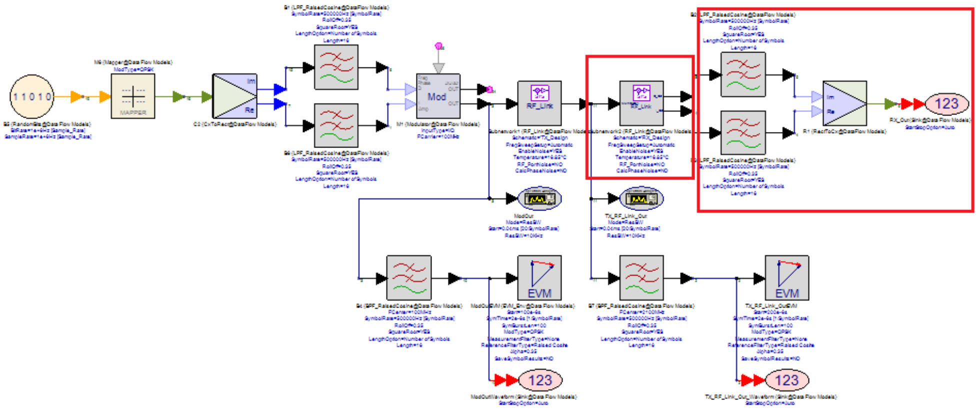
Variable
TX_RF_Link_Out_Power_Freq
TX_RF_Link_Out_Waveform
TX_RF_Link_Out_Waveform_Fc
TX_RF_Link_Out_Waveform_Time
TX_RF_Link_OutEVM_Droop
TX_RF_Link_OutEVM_EVM_Peak
TX_RF_Link_OutEVM_EVM_PeakSymbolInde
TX_RF_Link_OutEVM_EVM_RMS
TX_RF_Link_OutEVM_FreqError
TX_RF_Link_OutEVM_GainImbalance
TX_RF_Link_OutEVM_Index
TX_RF_Link_OutEVM_IQ_Offset
TX_RF_Link_OutEVM_MagErrorPeak
TX_RF_Link_OutEVM_MagErrorPeakSymbolI

| TX_RF_Li... | TX_RF_Link_OutEVM_EVM_RMS |
|---|---|
| 0 | 10.13 |

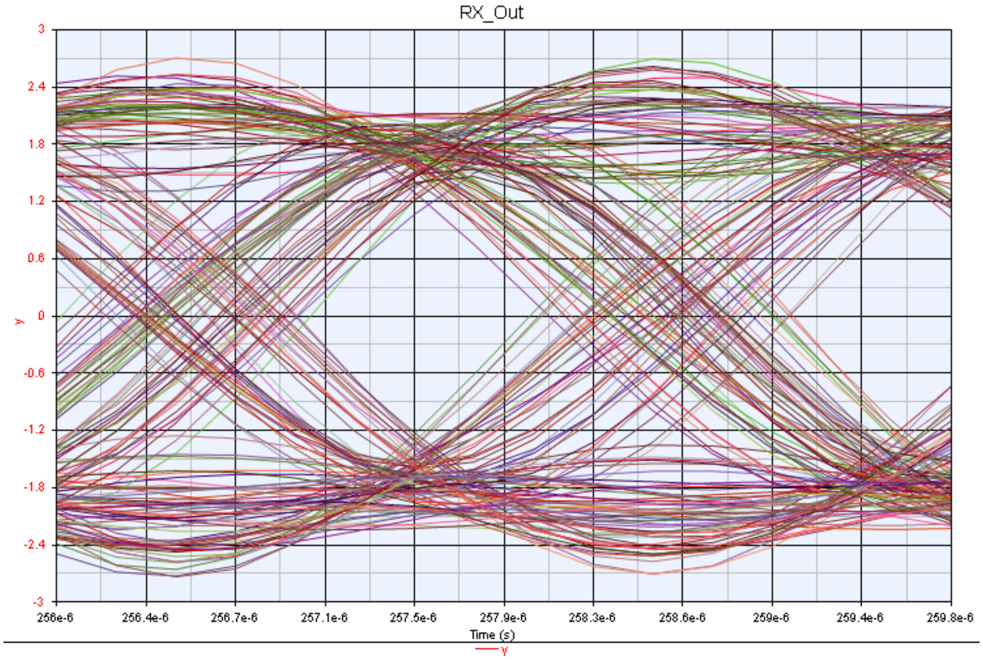
15. Now let's create a receiver. Add a new design and call it *RX_Design*. Place (from the RF_Design part library) a MultiSource and a Zero IF Receiver (ZIF_Rx). Set the Multisource to generate a CW signal at 2100 MHz (same as our RF_Link output frequency) and -50 dBm. Set the ZIF_Rx parameters as shown in the schematic below.



16. Now drag *RX_Design* from the workspace tree on the *QPSK_TX_RX* design. An RF_Link part is automatically instantiated. Connect it at the output of the first RF_Link. Double-click on it and check the Calculate Thermal Noise check-box and uncheck the Add source thermal noise to input checkbox. Activate the parts connected to its output.



17. Observe the eye diagram (*QPSK_TX_RX_Eye3* graph) at the output of the receiver.


RX_Out
3
2.4
1.8
1.2
0.6
0
-0.6
-1.2
-1.8
-2.4
-3
y
256e-6
256.4e-6
256.7e-6
257.1e-6
257.5e-6
257.9e-6
258.3e-6
258.6e-6
259e-6
259.4e-6
259.8e-6
Time (s)
y

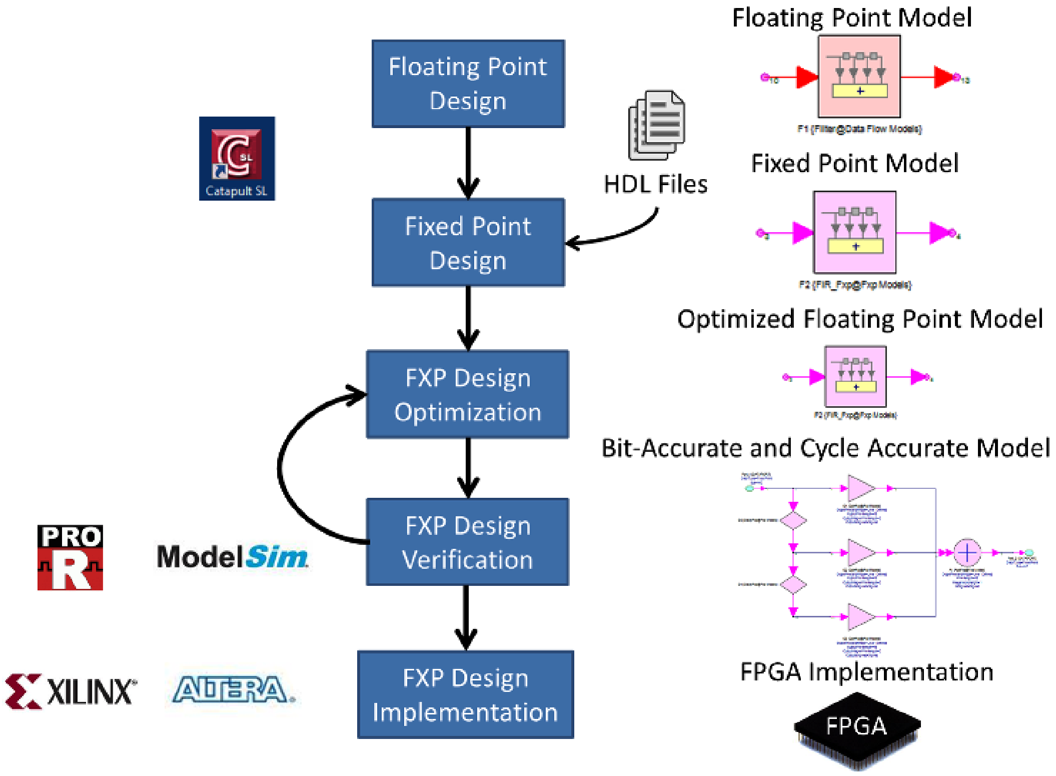
# Hardware Design

## Contents

## Getting Started with Hardware Design

### Introduction

This tutorial introduces the tools and features available in SystemVue to develop, verify and implement signal processing algorithms in Field Programmable Gate Arrays (FPGAs). The Figure below illustrates the general design flow and the associated tools and features in each stage of that flow



This tutorial consists of the following sections:

### Fixed Point Representation

This section introduces the Hardware Design Library and Fixed Point Representation.

### HDL Code Generation (Hardware Design)

This section describes how to generate HDL code (Verilog and VHDL) from a SystemVue design developed using the Hardware Design Library.

### C++ to HDL using Catapult design

This section illustrates how to migrate C++ based design to HDL design using SystemVue flow in Catapult.

### Fixed Point Optimization

This section explains how to use the floating point analysis table.

### HDL Cosimulation (Hardware Design)

This section explains in detail how to conduct HDL cosimulation using ModelSim /Questa and Riviera Pro HDL simulators.

### FPGA Implementation

This section describes the flow of implementing a generated HDL code on Xilinx FPGA prototyping board and verify the results using JTAG and ChipScope.

## Using Xilinx IP Cores

This section describes the steps required to cosimulate Xilinx IP cores inside SystemVue and generate HDL codes for hardware designs that contain Xilinx IP cores.

## Terminologies

- FPGA: Field Programmable Gate Arrays. For more information, you can refer to here.
- Cosimulation: is the process of simulating HDL-based designs inside SystemVue environment. For more information refer to HDL Cosimulation With ModelSim and Questa and HDL Cosimulation with Riviera-PRO,
- Code Generation: is the process of generating HDL code that describes the Hardware Design in the associated sub-network.
- Hardware Design: is a design developed in SystemVue using the parts of Hardware Design library.
- HDL: hardware description language
- VHSIC: Very High-Speed Integrated Circuit

## Brief Notes on Hardware Description Languages

The hardware description languages are used to describe the operation and the schematics of a digital design. They were originally created to assist in design simulation rather than schematic descriptions. This is a more portable and convenient alternative to creating a circuit from elements. As a result, HDL language constructs can be *synthesizable* i.e., suitable for synthesis and simulation, and *non-synthesizable* i.e., suitable only for simulation.
VHDL and Verilog are two widespread HDL languages.

VHDL stands for VHSIC Hardware Description Language. It is a standard HDL language that is used to describe digital hardware devices, systems and components. A VHDL-based design consists of at least one entity and one or more architectures. The entity section is used to declare the I/O ports of the circuit, while the architecture section is used to list the description code. The Verilog-based design consists of at least one module where the functions and procedures are defined. Standardized design libraries are included prior to the entity declaration in VHDL. Example: "library ieee;" and "use ieee.std_logic_1164.all;". STD_LOGIC and STD_LOGIC_VECTOR are basic data types used in VHDL, defined in IEEE standard 1164. STD_LOGIC can represent the following values: 1, 0, U (undefined), X (unknown), Z (high impedance), W (weak), H (weak 1), L (weak 0), - (don't care). STD_LOGIC_VECTOR is an array of STD_LOGIC. On the other hand, there is no concept of packages in Verilog.

In the following comparison table, we summarize the main elements of HDL codes as implemented using Verilog and VHDL.

| VHDL | Verilog |
|---|---|

| | | |
|---|---|---|
| **Language type** | Strong typed | Weak typed |
| **Identifiers case-sensitivity** | case-insensitive | case-sensitive |
| **Ports declaration** | entity counter is<br>Port (<br>CLK : in STD_LOGIC;<br>CLR : in STD_LOGIC;<br>DOUT : out STD_LOGIC_VECTOR (7 downto 0)\\);<br>end counter; | module counter(clk, clr, dout);<br>input clk;<br>input clr;<br>output [7:0] dout; |
| **Signals declaration** | signal value: std_logic_vector(7 downto 0); | wire [7:0] combinational_value;<br>reg [7:0] register_value; |
| **Concurrent assignments** | c <= a and b; | assign c = a & b; |
| **Conditional assignments** | c <= a when sw='1' else b; | assign c = sw ? a : b; |
| **Processes** | process (CLK,CLR) is<br>begin<br>if CLR='1' then<br>val<="00000000";<br>elsif rising_edge(CLK) then<br>val<=val+1;<br>end if;<br>end process; | always @ (posedge CLK or posedge CLR) begin<br>if (CLR) val<=0;<br>else val<=val+1;<br>end |
| **Nonblocking Sequential assignments** | process (CLK) is<br>begin<br>if rising_edge(CLK) then<br>val<=val+1;<br>val2<=val;<br>end if;<br>end process; | always @ (posedge CLK ) begin<br>val<=val+1;<br>val2<=val;<br>end |
| **Structural description** | counter_inst: component counter<br>port map( CLK => CLK, CLR => CLR, DOUT => counter_value ); | counter counter_inst(.CLK(CLK),.CLR(CLR),.counter_value(DOUT)); |
| **Component Declaration** | component counter is<br>Port ( CLK : in STD_LOGIC; CLR : in STD_LOGIC;<br>DOUT : out STD_LOGIC_VECTOR (7 downto 0));<br>end component counter; | No need |

|  |  |  |
|---|---|---|
| **Operators** | +<br>-<br>*<br>mod<br>not | +<br>_<br>*<br>%<br>~ |

### VHDL Code Example

This example shows a portion of the VHDL code generated using SystemVue for the fixed point counter (CounterFxp) from the Hardware Design library.

```
--------------------------------------------------------
------------------------
--
-- CounterFxp - VHDL source
--
--------------------------------------------------------
------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.p_fxp.all;

entity CounterFxp is
    generic(
        dataOutWL  : natural;
        dataOutIWL : integer;
        dataOutSGN : natural;

        Overflow       : natural;
        Quantization   : natural;
        SaturationBits : integer;

        FxpInitValue : integer := 0;
        StepFxp      : integer := 0;
        Direction    : integer := 0
    );
    port(
        -- Port: Inputs
        rst    : in std_logic;
        clk    : in std_logic;
        ce     : in std_logic;
        ENABLE : in std_logic;
        RESET  : in std_logic;

        -- Port: Outputs
        RDY     : out std_logic;
        dataOut : out std_logic_vector(dataOutWL-1
downto 0)
    );

end CounterFxp;

architecture CounterFxp_Arch of CounterFxp is

    signal countExt      : std_logic_vector (dataOutWL
downto 0);
    signal count         : std_logic_vector (dataOutWL-1
 downto 0);
    constant internalSGN : boolean := (dataOutSGN = 0
and Direction = 1) or dataOutSGN = 1;

begin

    count <= FxpConvert(countExt,
                        (dataOutWL+1, dataOutIWL+1,
internalSGN),
                        (dataOutWL, dataOutIWL,
to_boolean(dataOutSGN)),
                        ToFxpQznModeT(Quantization),
ToFxpOvfModeT(Overflow), SaturationBits);

    process (CLK)
    begin
        if CLK'event and clk = '1' then
            if rst = '1' then
                if dataOutSGN = 1 then
                    countExt <= std_logic_vector
(to_signed(FxpInitValue, (dataOutWL+1)));
                else
                    countExt <= std_logic_vector
(to_unsigned(FxpInitValue, (dataOutWL+1)));
                end if;
            else
                if CE = '1' then
                    if RESET = '1'then
                        if dataOutSGN = 1 then
                            countExt <= std_logic_vector
(to_signed(FxpInitValue, (dataOutWL+1)));
                        else
                            countExt <= std_logic_vector
(to_unsigned(FxpInitValue, (dataOutWL+1)));
                        end if;
                    elsif ENABLE = '1' then
                        if Direction = 0 then
                            if dataOutSGN = 1 then
                                countExt <=
std_logic_vector(signed(count) + to_signed(StepFxp,
dataOutWL+1));
                            else
```

```
                                countExt <=
std_logic_vector(unsigned(count) + to_unsigned(StepFxp,
dataOutWL+1));
                            end if;
                        else
                            if dataOutSGN = 1 then
                                countExt <=
std_logic_vector(signed(count) - to_signed(StepFxp,
dataOutWL+1));
                            else
                                countExt <=
std_logic_vector(unsigned(count) - to_unsigned(StepFxp,
dataOutWL+1));
                            end if;
                        end if;
                    end if;
                end if;
            end if;
        end if;
    end process;

    dataOut <= count;
    rdy     <= CE;

end CounterFxp_Arch;
```

### Verilog Code Example

This example shows a portion of the Verilog code generated using SystemVue for the fixed point counter (CounterFxp) from the Hardware Design library.

```
//------------------------------------------------------
------------------------
//
// CounterFxp - Verilog source
//
//------------------------------------------------------
------------------------

module CounterFxp(dataOut, rdy, enable, reset, rst,
clk, ce);

    parameter dataOutWL = 0, dataOutIWL = 0, dataOutSGN
= 0;
    parameter Quantization = 0, Overflow = 0,
SaturationBits = 0;
    parameter [dataOutWL-1:0] FxpInitValue = 0;
    parameter [dataOutWL-1:0] StepFxp = 0;
    parameter Direction = 0;
    parameter internalSGN = (!dataOutSGN & Direction) |
dataOutSGN;
```

```
    input clk, rst, enable, reset, ce;
    output rdy;
    output [dataOutWL-1:0] dataOut;

    wire [dataOutWL-1:0] count;
    reg [dataOutWL:0] countExt;

    assign rdy = ce;
    assign dataOut = (dataOutSGN) ? $signed(count) :
count;

    fxpconvert #(.xWL(dataOutWL+1), .xIWL(dataOutIWL+1),
.xSGN(internalSGN),
                .yWL(dataOutWL), .yIWL(dataOutIWL), .
ySGN(dataOutSGN),
                .quantMode(Quantization),
                .ovfMode(Overflow),
                .satBits(SaturationBits)
                ) convertCount(count, countExt);

    always@(posedge clk) begin
        if (rst) begin
            if (dataOutSGN) begin
                countExt = $signed(FxpInitValue); end
            else begin
                countExt = FxpInitValue;
            end end
        else begin
            if (ce) begin
                if (reset) begin
                    if (dataOutSGN) begin
                        countExt = $signed
(FxpInitValue); end
                    else begin
                        countExt = FxpInitValue;
                    end end
                else if (enable) begin
                    if (Direction) begin
                        if (dataOutSGN) begin
                            countExt = $signed(count) -
$signed(StepFxp); end
                        else begin
                            countExt  = count - StepFxp;
                        end end
                    else begin
                        if (dataOutSGN) begin
                            countExt = $signed(count) +
$signed(StepFxp); end
                        else begin
                            countExt  = count + StepFxp;
                        end
                    end
                end
            end
        end
    end

endmodule
```

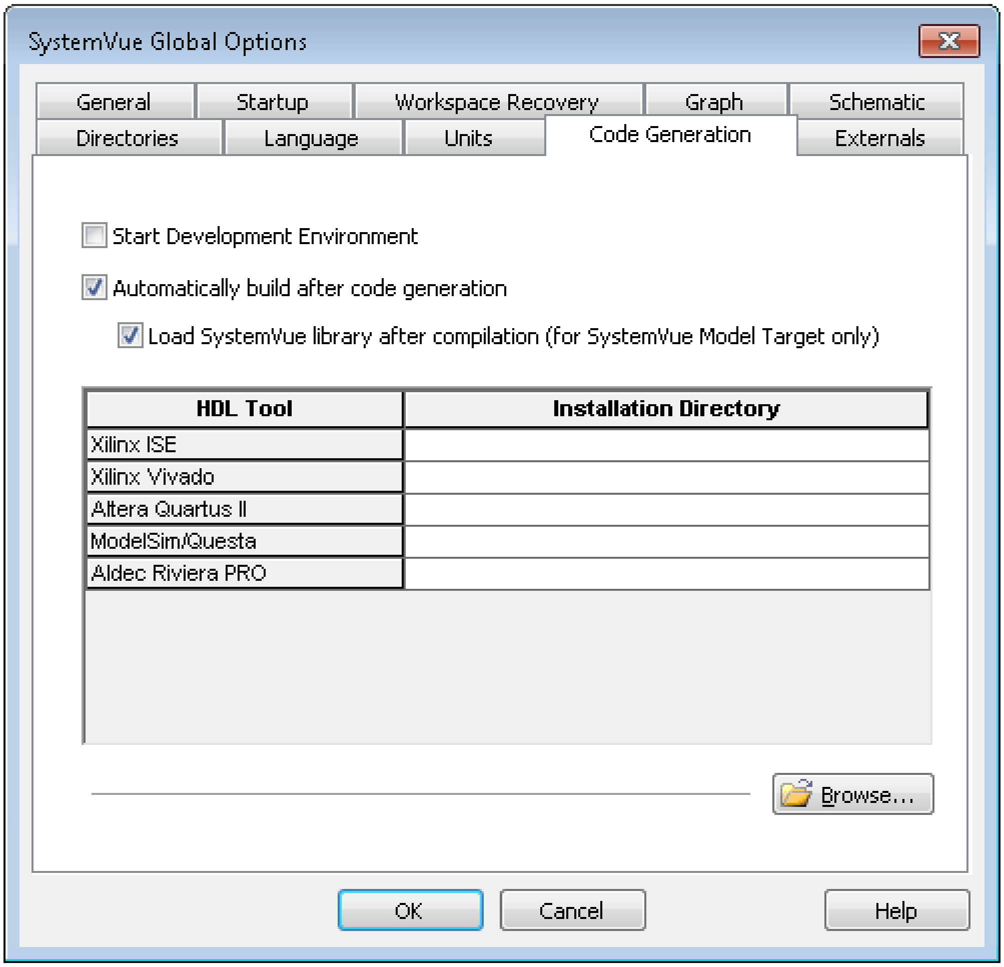
### Before Starting

Depending on the tutorials that you are going to follow, you may need to install the following software:

- Riviera Pro from Aldec: For HDL Cosimulation Tutorial
- ModelSim/Questa from Mentor Graphics: For HDL Cosimulation Tutorial
- Catapult from Mentor Graphics: For C Plus Plus to HDL Using Catapult Design
- ISE from Xilinx: For HDL Code Generation Tutorial, FPGA Implementation Tutorial, and Using Xilinx IP Cores Tutorial.
- Quartus from Altera: For HDL Code Generation Tutorial and FPGA Implementation.

For software version compatibility, refer to System Requirements

After installing the required software, configure the installation paths in the Code Generation tab of Global Options which is shown in the figure below:



□ Start Development Environment

☑ Automatically build after code generation

☑ Load SystemVue library after compilation (for SystemVue Model Target only)

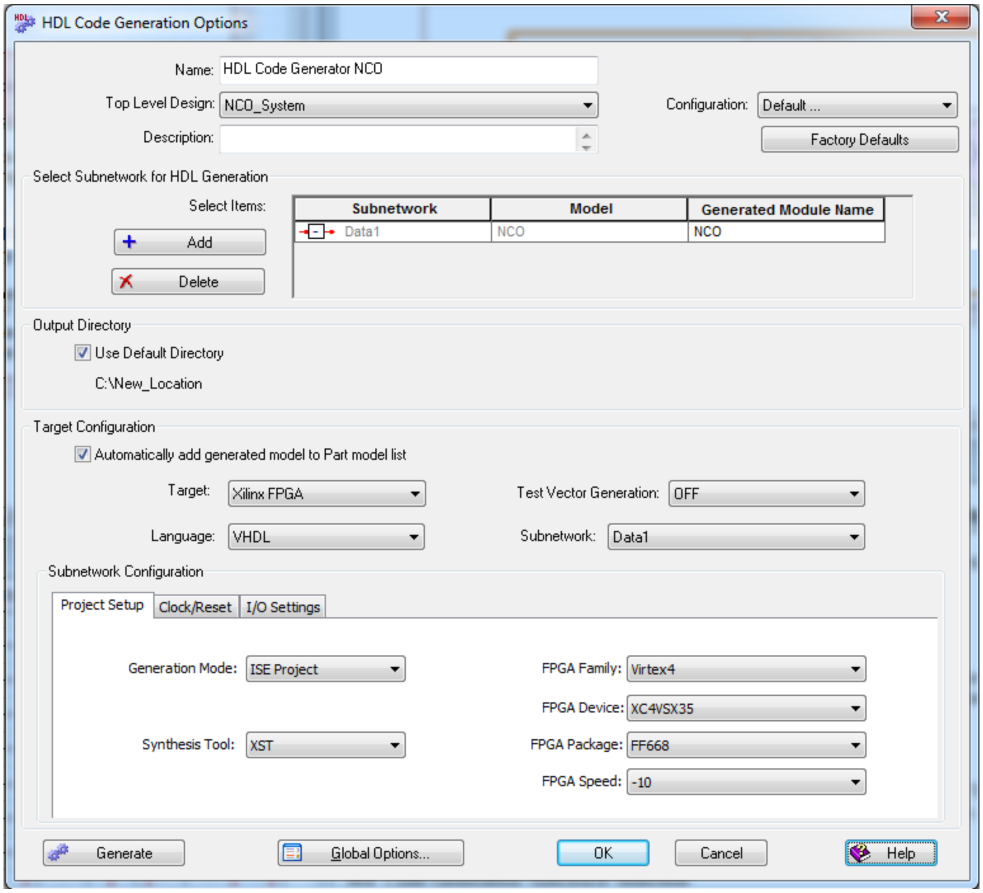
| HDL Tool | Installation Directory |
|---|---|
| Xilinx ISE | |
| Xilinx Vivado | |
| Altera Quartus II | |
| ModelSim/Questa | |
| Aldec Riviera PRO | |

## Software Setup Verification For Training

Before going through the steps below, make sure that you are able to open each of the following software without any licensing problem:

- Riviera Pro from Aldec
- ModelSim/Questa from Mentor Graphics
- ISE from Xilinx
- Quartus from Altera

1. Open SystemVue
2. Go to Help -> Open Example...
3. Go to Hardware Design -> HDLCodeGeneration --> NCO
4. Open NCO.wsv
5. Save the workspace in a new location <Example_Location>
6. Double-Click on HDL Code Generation NCO
7. Configure it as shown in the figure below and then click Generate



HDL Code Generation Options

Name: HDL Code Generator NCO

Top Level Design: NCO_System

Configuration: Default ...

Description:

Select Subnetwork for HDL Generation

Select Items:

| Subnetwork | Model | Generated Module Name |
|---|---|---|
| Data1 | NCO | NCO |

Output Directory

☑ Use Default Directory

C:\New_Location

Target Configuration

☑ Automatically add generated model to Part model list

Target: Xilinx FPGA

Test Vector Generation: OFF

Language: VHDL

Subnetwork: Data1

Subnetwork Configuration

Generation Mode: ISE Project

FPGA Family: Virtex4

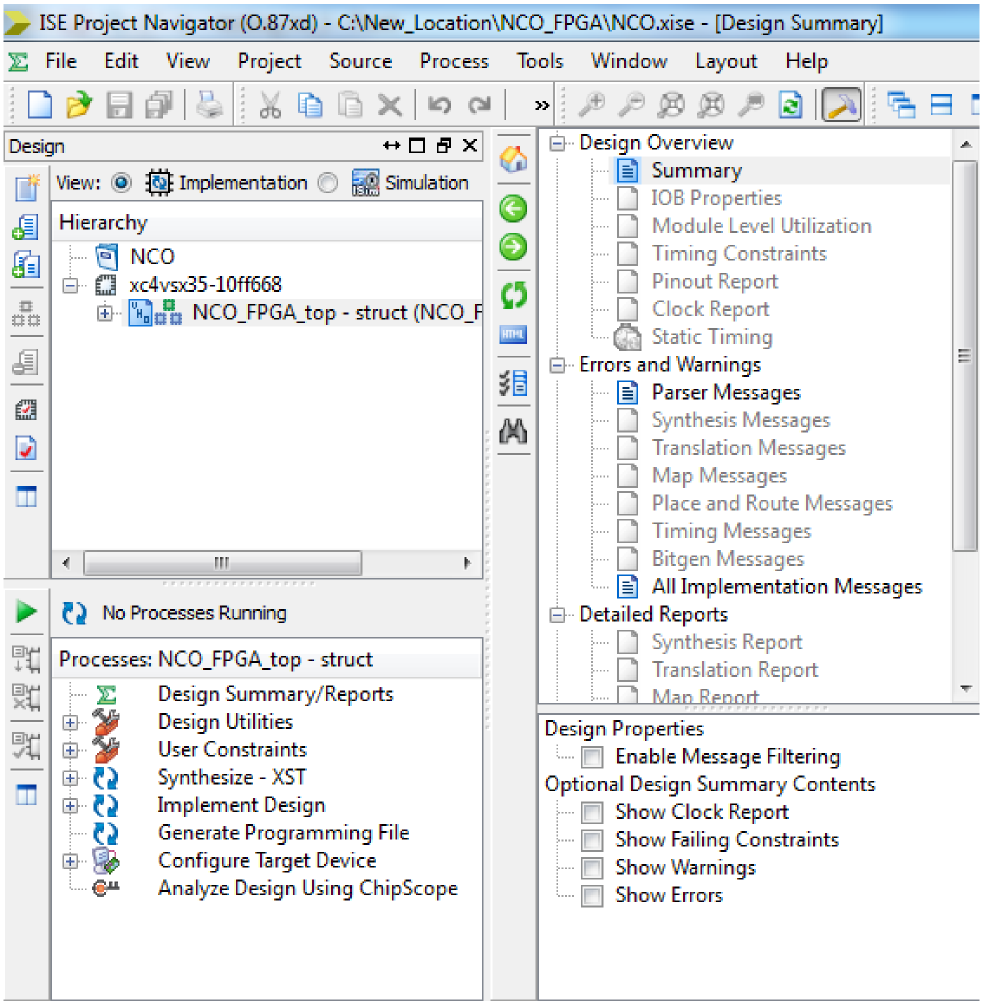
FPGA Device: XC4VSX35

Synthesis Tool: XST

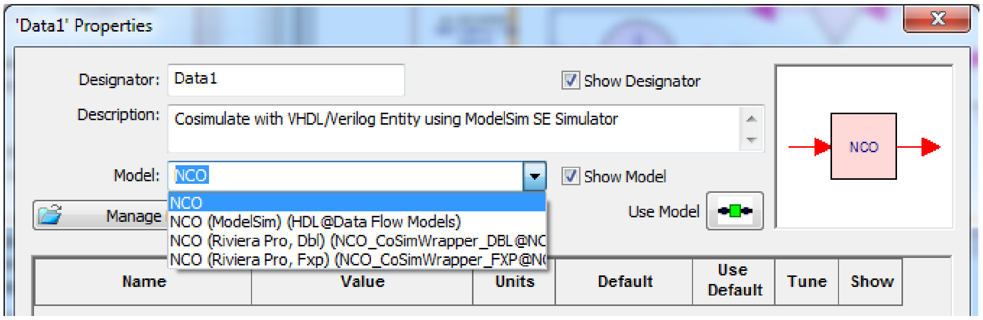
FPGA Package: FF668

FPGA Speed: -10

8. Make sure that Xilinx ISE project will be invoked as shown below. If so, this is a sign that your Xilinx setup is correct.



9. Double-click on the NCO block in the NCO_System schematic and make sure there are four models in the model list as shown in the figure below. If so, this is a sign that your Riviera PRO setup is correct


'Data1' Properties

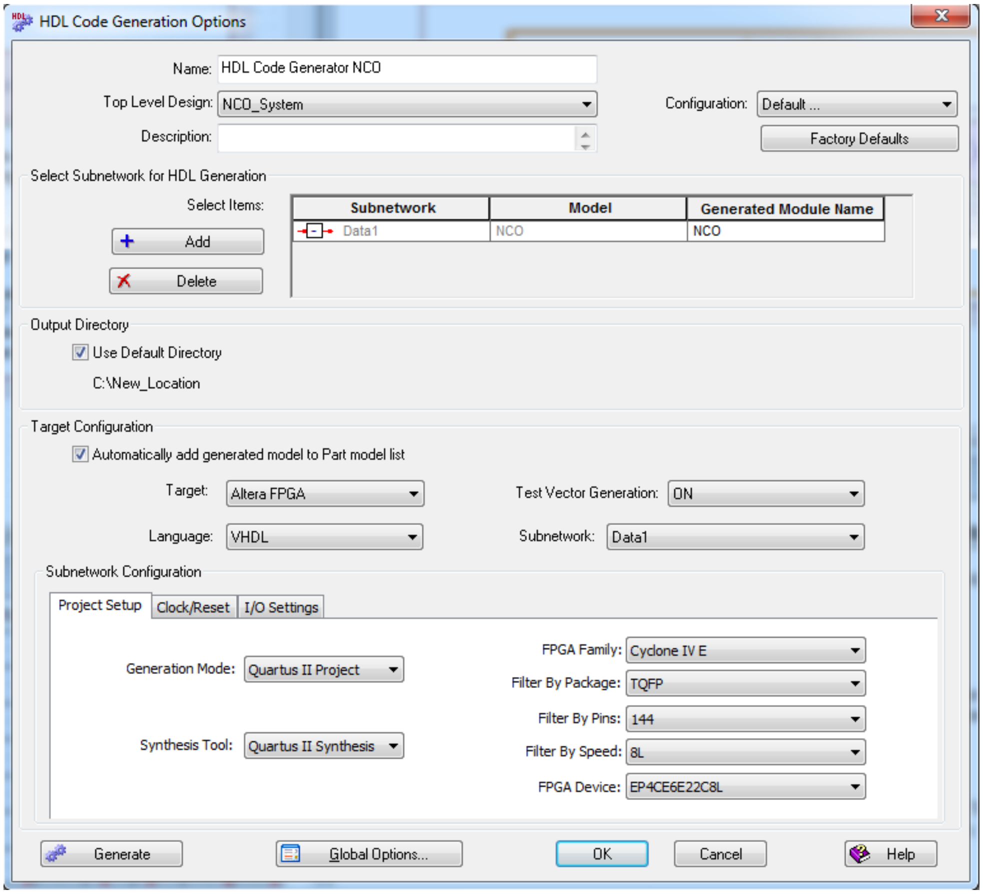
Designator: Data1 ☑ Show Designator

Description: Cosimulate with VHDL/Verilog Entity using ModelSim SE Simulator

Model: NCO ☑ Show Model

NCO
NCO (ModelSim) (HDL@Data Flow Models)
NCO (Riviera Pro, Dbl) (NCO_CoSimWrapper_DBL@NC
NCO (Riviera Pro, Fxp) (NCO_CoSimWrapper_FXP@N(

10. Switch to NCO (ModelSim) model and run the simulation. If it works, this is a sign that ModelSim/Questa Setup is correct
11. Close Xilinx ISE program
12. Double-Click on HDL Code Generation NCO
13. Configure it as shown in the following figures:



HDL Code Generation Options

Name: HDL Code Generator NCO

Top Level Design: NCO_System

Configuration: Default ...

Description:

Select Subnetwork for HDL Generation

Select Items:

| Subnetwork | Model | Generated Module Name |
|---|---|---|
| Data1 | NCO | NCO |

Output Directory

☑ Use Default Directory

C:\New_Location

Target Configuration

☑ Automatically add generated model to Part model list

Target: Altera FPGA

Test Vector Generation: ON

Language: VHDL

Subnetwork: Data1

Subnetwork Configuration

Project Setup

Generation Mode: Quartus II Project

Synthesis Tool: Quartus II Synthesis

FPGA Family: Cyclone IV E

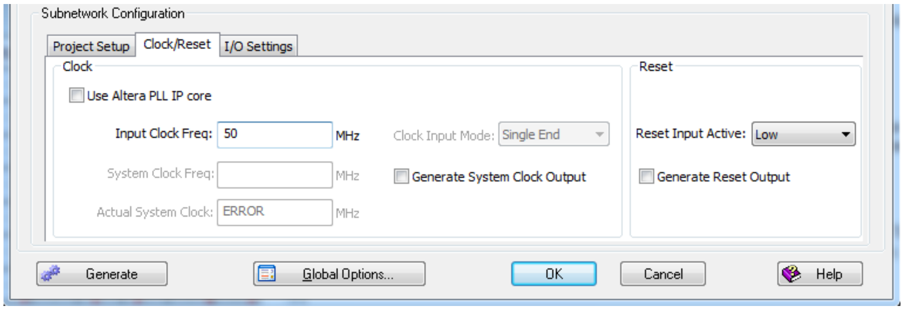
Filter By Package: TQFP

Filter By Pins: 144

Filter By Speed: 8L

FPGA Device: EP4CE6E22C8L

a. Add the clock value as shown below



Subnetwork Configuration

Clock/Reset

Clock

□ Use Altera PLL IP core

Input Clock Freq: 50 MHz

Clock Input Mode: Single End

System Clock Freq: MHz

□ Generate System Clock Output

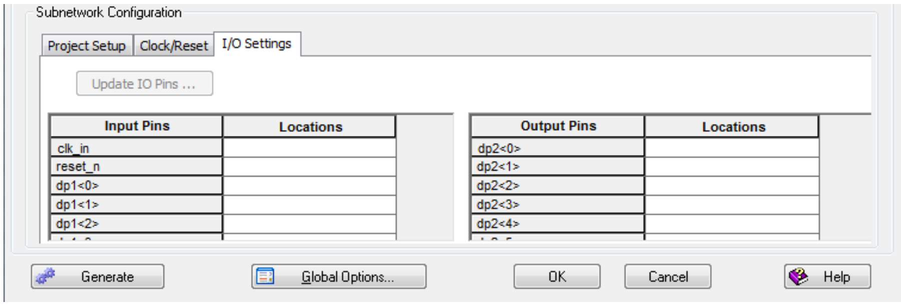
Actual System Clock: ERROR MHz

Reset

Reset Input Active: Low

□ Generate Reset Output

b. Click on "Update IO Pins..." as shown below



Subnetwork Configuration

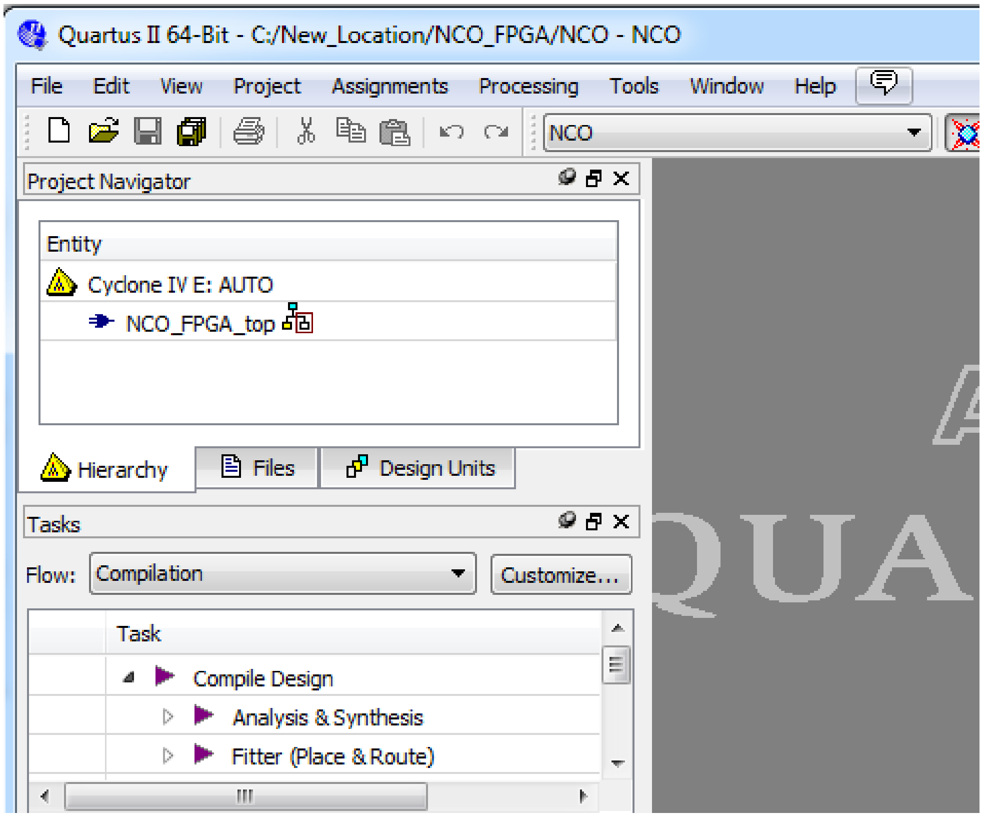
I/O Settings

| Input Pins | Locations |
|---|---|
| clk_in | |
| reset_n | |
| dp1<0> | |
| dp1<1> | |
| dp1<2> | |

| Output Pins | Locations |
|---|---|
| dp2<0> | |
| dp2<1> | |
| dp2<2> | |
| dp2<3> | |
| dp2<4> | |

c. then click Generate

14. Make sure that Altera Quartus project will be invoked as shown in the figure below. If so, this is a sign that your Altera setup is correct.



### Compiling Xilinx IP core simulation libraries

For Using Xilinx IP Cores Tutorial, you need to compile the Xilinx IP core libraries which may take time. It is recommended to compile these libraries in advance by following the steps in Using Xilinx IP Cores > Compiling Xilinx IP core simulation libraries

## Fixed Point Representation

### Prerequisite

- SystemVue 2012.06 or higher with Hardware Design Library

NOTE For software version compatibility, refer to System Requirements

### Associated tutorial workspace

- Examples\Tutorial\Hardware_Design\Basics\Fixed-PointParameters.wsv

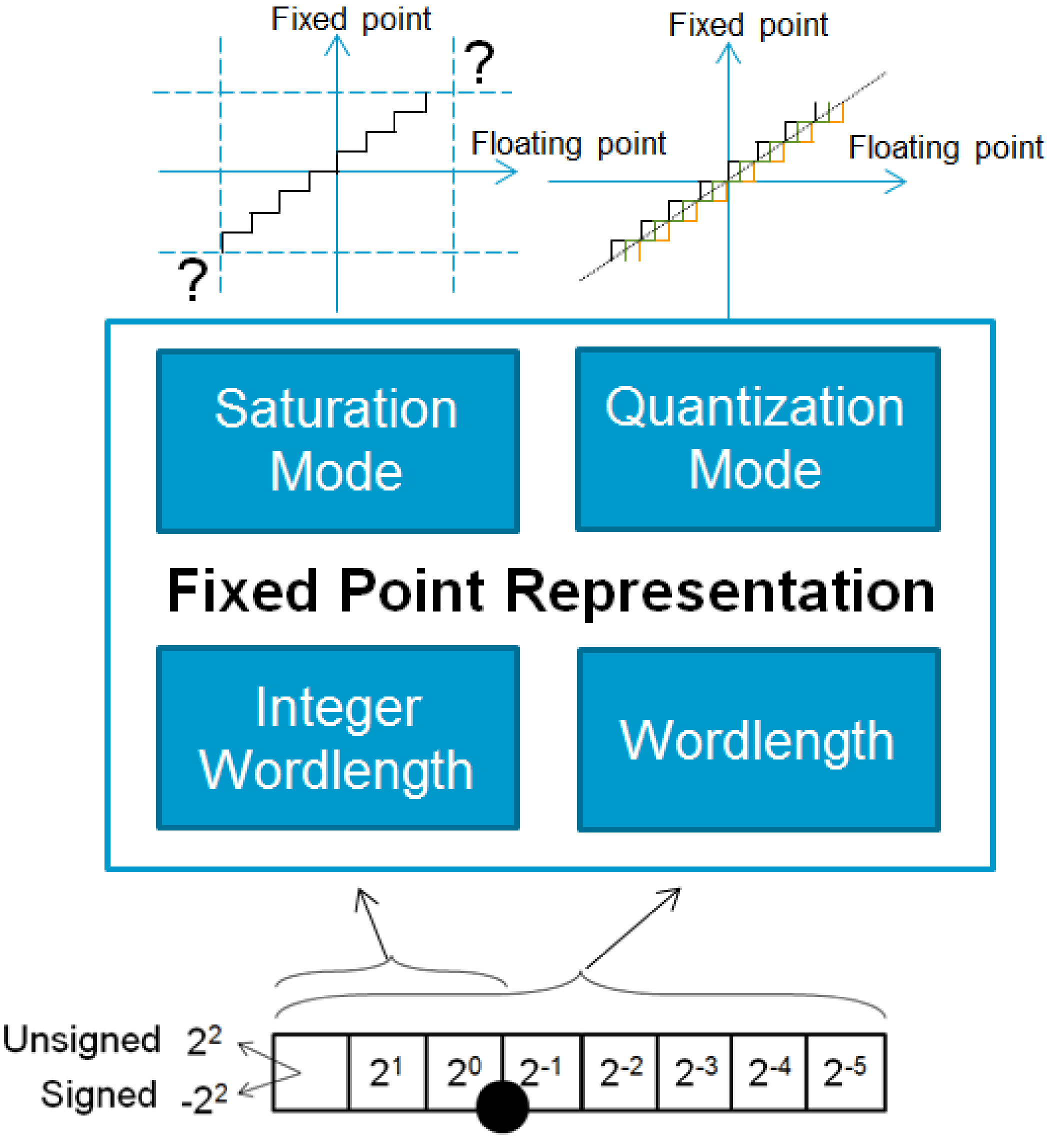
### Introduction

SystemVue FixedPoint Data Type has the computational behavior of SystemC $^{TM}$ 2.2 fixed point type based on IEEE Std. 1666 $^{TM}$ Language Reference Manual (LRM). The fixed point representation based on that standard is as follows:

FxpDataType<WL, IWL, IsSigned, Q_mode, S_mode, n_bits>
where,

- WL - total wordlength
- IWL - integer wordlength
- IsSigned - Unsigned number for *Zero* and Signed Number for *One*.
- Q_mode - Quantization mode; determines the behavior when the number to be represented requires more precision than is available
- S_mode - Saturation mode; determines the behavior when the number to be represented is outside the dynamic range covered
- n_bits - number of saturated bits (used by Saturation mode)

The figure provides an overview of the fixed point representation. Note that the decimal point determines the weights across different bits. Also, the MSB has a negative weight if it is *zero* and a positive weight if it is *one*. The range of numbers using this representation is Asymmetric: -2^(WL-1) to 2^(WL-1)-1, where WL is the wordlength.



This tutorial provides a brief description of the fixed point representation and explains the fixed point parameters used in the parts of Hardware Design library. In addition, it demonstrates the functionality of some bit manipulation parts available in the Hardware Design Library.

NOTE The fixed point representation used in the Hardware Design Library is the **2's complement** type. To negate a number of type, complement its binary representation and then add 1 to the result.

## Fixed Point Parameters

Fixed point data type is described by several parameters that are documented in Fixed Point Simulation. In this tutorial, we will focus on four parameters: Wordlength, Integer wordlength, Saturation mode, Quantization mode.

### Wordlength

A fixed point number of wordlength WL can represent 2^WL different values. The minimum and maximum value can be determined by setting the Integer wordlength and IsSigned parameters. The range of values bounded by the minimum and maximum values is the dynamic range of the fixed point number.

### Integer wordlength

This defines the portion of the WL bits that are used for representing integer numbers. The sign bit is included in the integer wordlength.

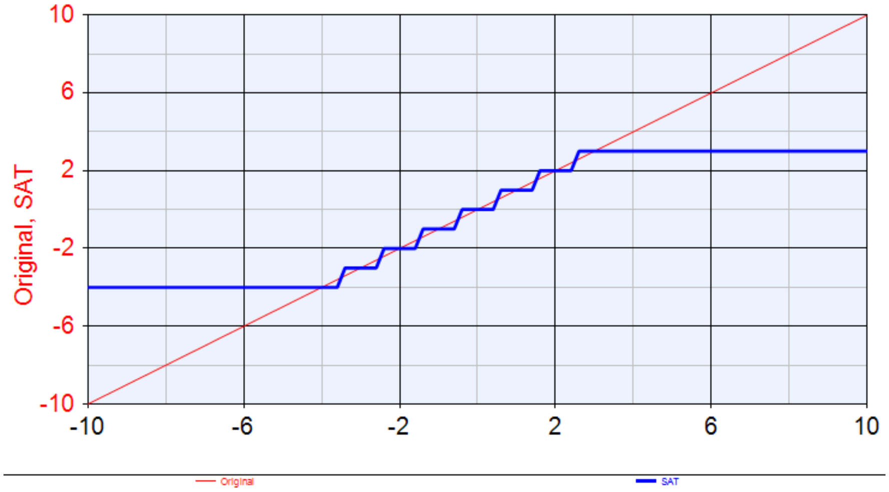
Examples

| <WL,IWL,IsSigned> | **11101001** |
|---|---|
| <8,8,0> | **11101001.** ➔ 128+64+32+0+8+0+0+1= **233** |
| <8,8,1> | **11101001.** ➔ -128+64+32+0+8+0+0+1 = **-23** |
| <8,1,0> | **1.1101001** ➔ 1+1/2+1/4+0+1/16+0+0+1/128= **1.8203125** |
| <8,1,1> | **1.1101001** ➔ -1+1/2+1/4+0+1/16+0+0+1/128= **-0.1796875** |
| <8,0,0> | **.11101001** ➔ 1/2+1/4+1/8+0+1/32+0+0+1/256= **0.91015625** |
| <8,0,1> | **.11101001** ➔ -1/2+1/4+1/8+0+1/32+0+0+1/256= **-0.08984375** |

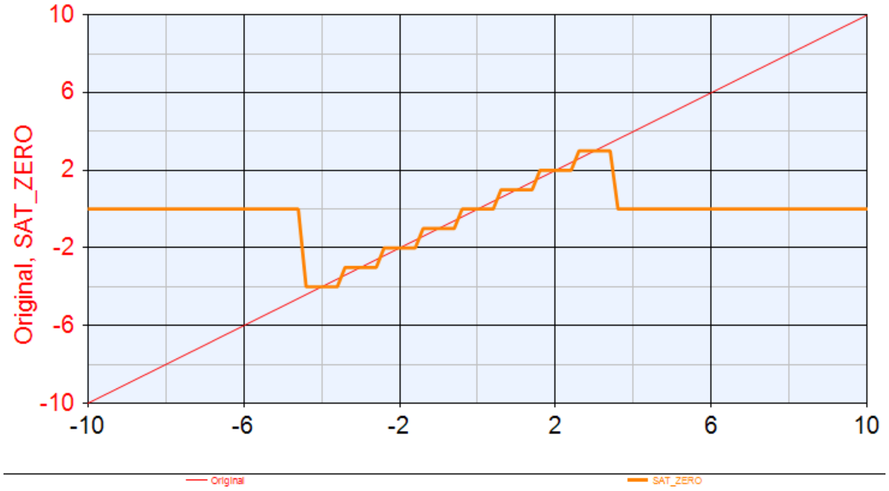
### Saturation mode

This mode defines how to represent numbers that are outside the dynamic range of the fixed point representation. The supported modes are SAT,SAT_ZERO,SAT_SYM, WRAP, and WRAP_SM. The following subsections describe each one of these modes using a fixed point representation x<WL=3,IWL=3,IsSigned=1>.

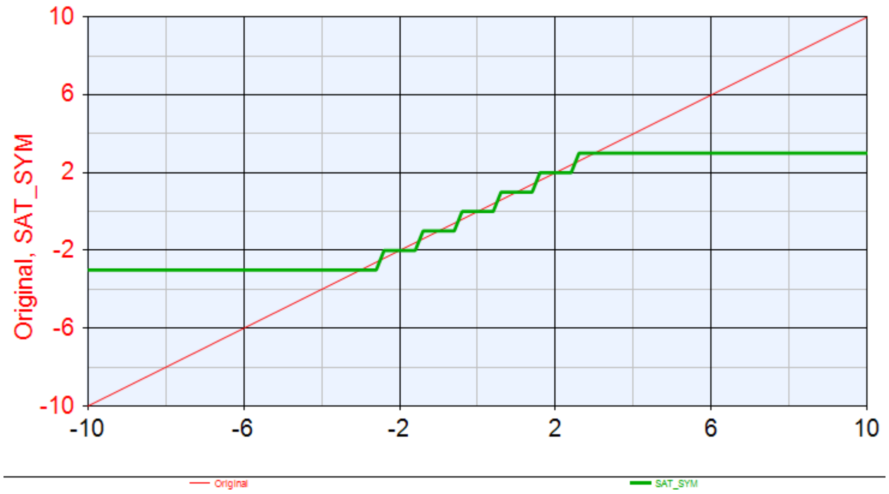
- SAT: Saturation
  - Positive : Set to *MAX* when x > MAX,
  - Negative : Set to *Min* when x < MIN



- SAT_ZERO: Saturation to Zero
  - Positive : Set to *zero* when x > MAX,
  - Negative : Set to *zero* when x < MIN



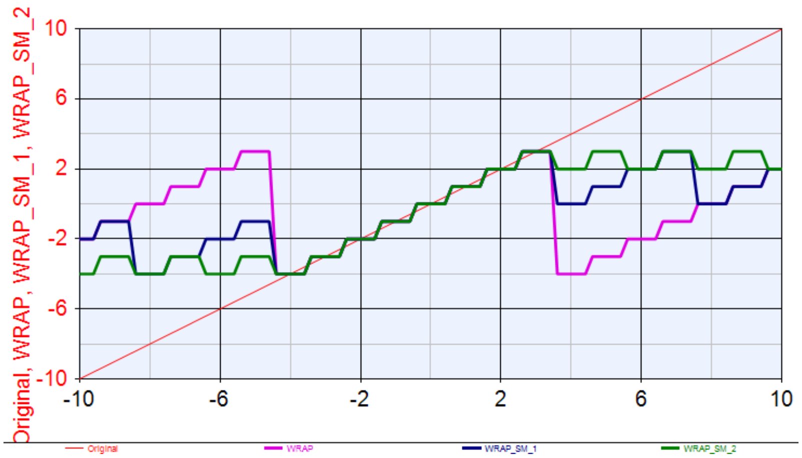
- SAT_SYM: Symmetrical Saturation
  - Positive : Set to *MAX* when x > MAX,
  - Negative : Set to -*MAX* when x < MIN



- WRAP: wrap around overflow operation
  - Saturation Bits are set to *zero*
    - Positive: When x exceeds MAX, it wraps around it and starts from MIN (This is the effect of zero out all bits after MSB)
    - Negative: When x exceeds MIN, it wraps around it and starts from MAX (This is the effect of zero out all bits after MSB)
  - Saturation Bits are set to *nonzero* value.
    - The value of the saturation bits determines the number of MSB bits be saturated or set to 1. The sign bit is retained so that positive numbers remain positive and negative numbers remain negative. To illustrate this mode, the positive and negative cases are explained through example of saturating a fixed point number x<5,5,1> and saturation bits = 3
    - Positive: When x reaches 0 1 1 1 1 = 15, three MSB bits are saturated: the sign bit (which remains as is), MSB-1 and the MSB-2 bits. As a result, incrementing x by 1 will result in 0 1 1 0 0 = 12.
    - Negative: When x reaches 1 0 0 0 0 = -16, three MSB bits are saturated: the sign bit (which remains as is) MSB-1 and the MSB-2 bits. As a result, decrementing x by 1 (adding 1 1 1 1) will result in 1 0 0 1 1 = -13.

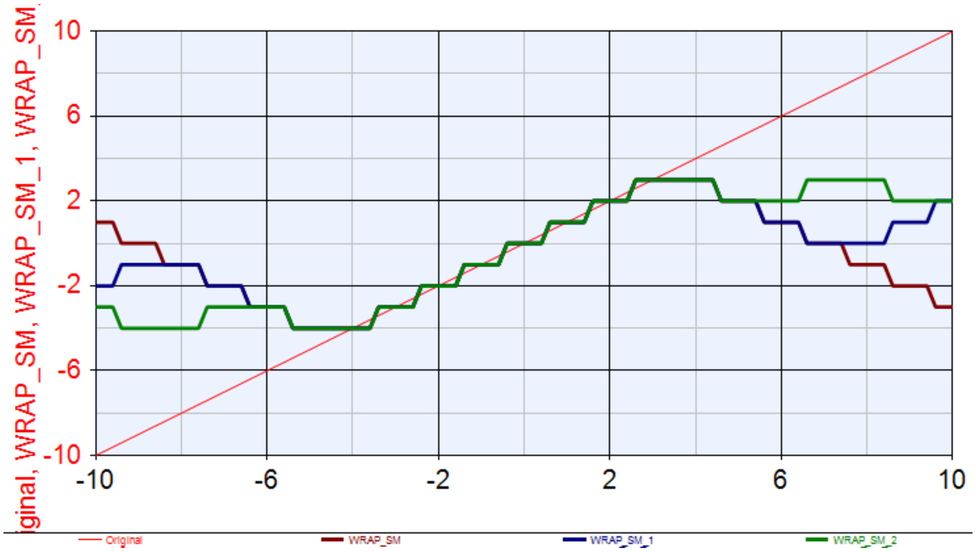
**CAUTION** This is the default Saturation mode, the default value of saturation bits is *zero*.

In the figure below, WRAP_1 refers to WARP mode with saturation bits = 1, WRAP_2 refers to WARP mode with saturation bits = 2.



- WRAP_SM: It is a variation of the WRAP mode that avoids the strong non-linear behavior caused by the wrap-around operation.

  In the figure below, WRAP_SM_1 refers to WRAP_SM mode with saturation bits = 1, WRAP_SM_2 refers to WRAP_SM mode with saturation bits = 2.



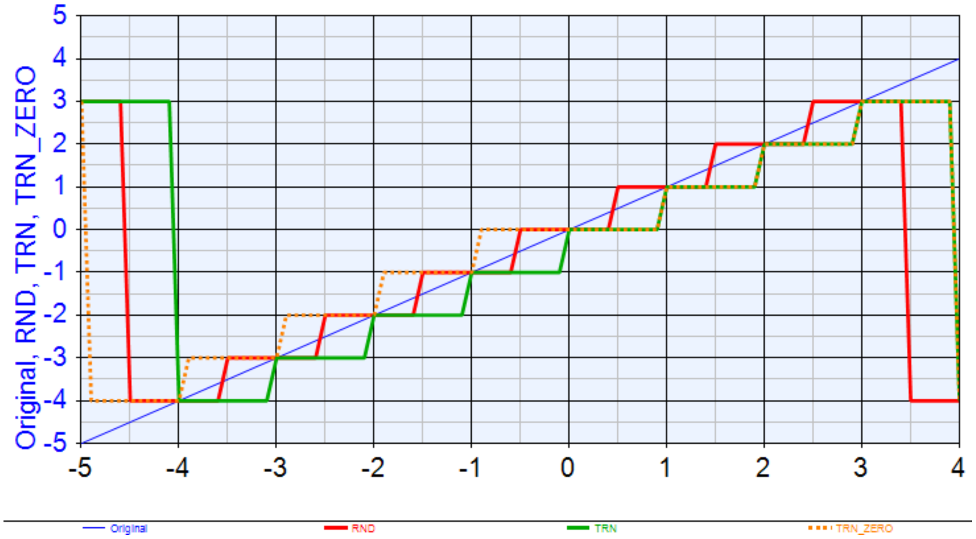
### Quantization mode

This mode defines the quantization threshold that determines the transition between two consequent fixed point values.

- RND: Round towards plus infinity
  - Redundant MSB bit will be added to the remaining bits.( round the value to the closest representable number):
  - has hardware cost but removes negative bias
- RND_CONV: Convergent rounding
- RND_INF: Round towards infinity
- RND_MIN_INF: Round towards minus infinity
  - Redundant bits are dropped.

- RND_ZERO: Round towards zero
  - For positive numbers: redundant bits are deleted
  - For negative numbers: redundant MSB bit is added
- TRN: Truncation (Default): simple but slight negative bias
- TRN_ZERO: Truncation to zero

The following figure shows the behavior of RND, TRN and TRN_ZERO using a fixed point representation x<WL=3,IWL=3,IsSigned=1>.



Below are some additional examples:

Examples

| Quantization mode | <WL=4,IWL=2,IsSigned=0> → <WL=3,IWL=2, IsSigned=0> | <WL=4,IWL=2,IsSigned=1> → <WL=4,IWL=3, IsSigned=1> |
|---|---|---|
| **RND** | **01.01** = 1.25 → **01.1** = 1.5 | **10.11** = -1.25 → **11.0** = -1 |
| **RND_INF** | **01.01** = 1.25 → **01.1** = 1.5 | **10.11** = -1.25 → **10.1** = -1.5 |
| **RND_MIN_INF** | **01.01** = 1.25 → **01.0** = 1.0 | **10.11** = -1.25 → **10.1** = -1.5 |
| **TRN** | **01.01** = 1.25 → **01.0** = 1.0 | **10.11** = -1.25 → **10.1** = -1.5 |

After obtaining some background about different fixed point parameters, open Fixed-PointParameters.wsv workspace at Examples\Tutorial\Hardware_Design\Basics and follow the steps (2) Overflow and (3) Quantization in the Tutorial_Description note to explore the behavior of fixed point parameters at different settings.

## Bit Manipulation

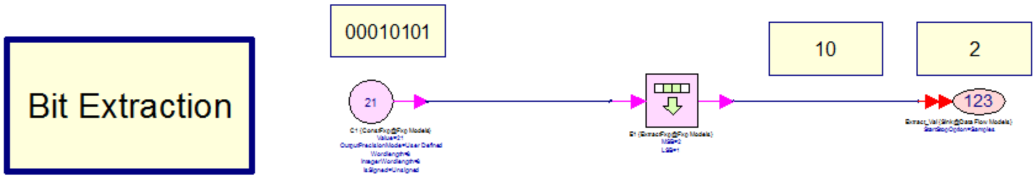
The following are some bit-level controls that can be performed in SystemVue using the Hardware Design library.

### Bit Extraction

Bit extraction is the operation of extracting a fixed point portion with wordlength WL1 from another fixed point number WL2, where WL1 < WL2. This can be done using ExtractFxp, which extracts a group of bits starting from the bit = LSB parameter until bit = MSB parameter.

**CAUTION** **The format and signed/unsigned nature of the input is not taken into account, the output will always be an unsigned integer, i.e. Wordlength = Integer Wordlength.**

The figure below is an example of using ExtractFxp in Fixed-PointParameters.wsv example.

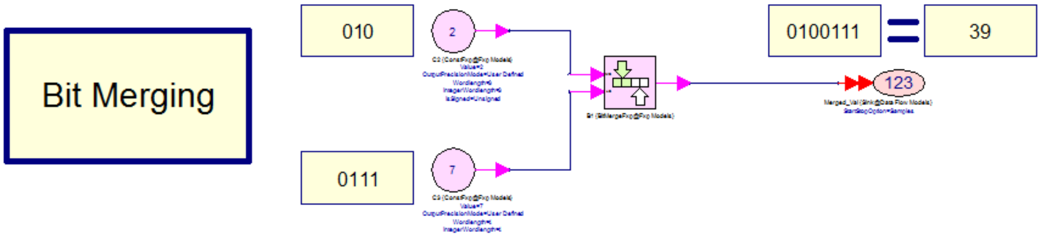


### Bit Merging

Bit merging is the operation of combining two fixed point numbers with wordlength WL1 and WL2 into a larger fixed point number with wordlength WL = WL1 + WL2. This can be done using BitMergeFxp.

**CAUTION** **The formats and signed/unsigned natures of the inputs are not taken into account, the output will always be an unsigned integer, i.e. Wordlength = Integer Wordlength.**

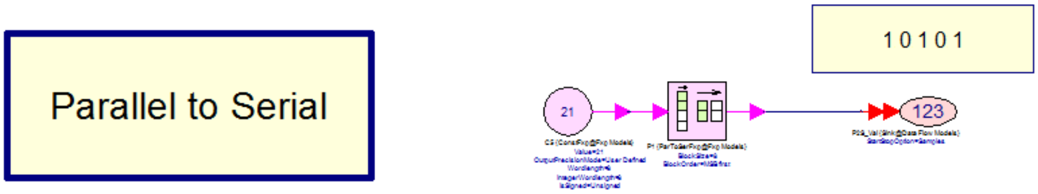
The figure below is an example of using BitMergeFxp in Fixed-PointParameters.wsv example.



### ParallelToSerial

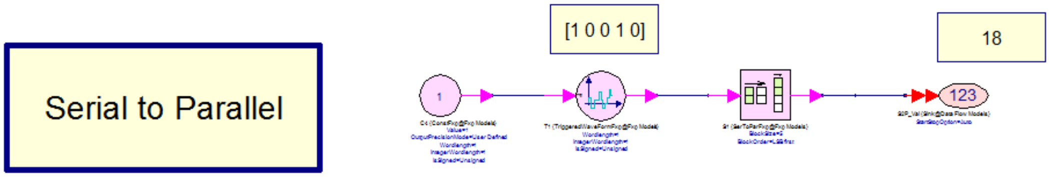
The parallel to serial operation divides a fixed point number of wordlength WL into several fixed point numbers produced at the output of wordlength = BlockSize. When BlockSize is = 1, the fixed point number is produced at the output bit by bit.

This operation is done using ParToSerFxp.
The figure is an example of using ParToSerFxp in Fixed-PointParameters.wsv example.



### SerialToPrallel

The serial to parallel operation combines several fixed point numbers of wordlength = WL into a single fixed point number with wordlength = BlockSize x WL. This operation is done using SerToParFxp.
The figure is an example of using SerToParFxp in Fixed-PointParameters.wsv example.



Open Fixed-PointParameters.wsv workspace at Examples\Tutorial\Hardware_Design\Basics and follow the steps under section 1-Bit Manipulation in the Tutorial_Description note to explore the behavior of bit-level controls introduced above.

## HDL Code Generation

Prerequisite

- SystemVue 2012.06 or higher with Hardware Design Library

NOTE For software version compatibility, refer to System Requirements

Associated workspaces

- HDL_CODE_GEN.wsv in Examples\Tutorials\Hardware_Design\CodeGen\SimpleDesign
- CustomHDL_CodeGen.wsv in Examples\Tutorials\Hardware_Design\CodeGen\ExistingHDL

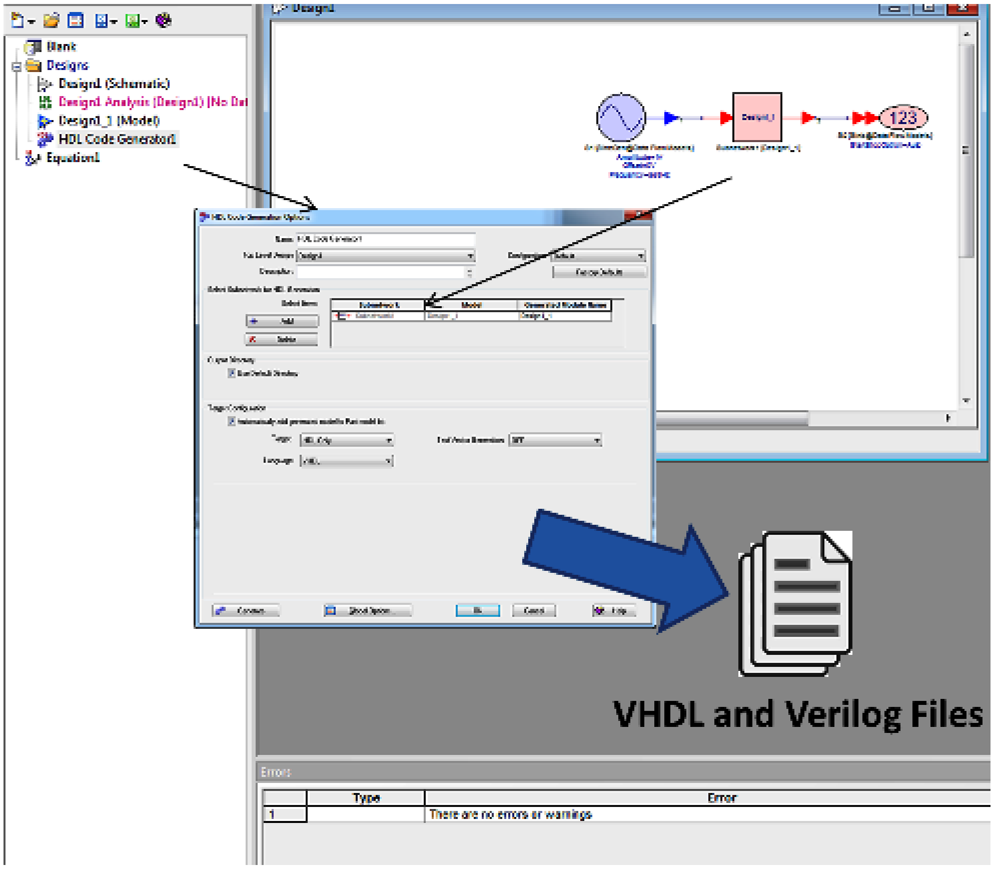
CAUTION We recommend that you copy the complete directory of **Examples\Tutorials\Hardware_Design** to your local directory to avoid file writing permission issues on default installation directory.

## Introduction

The HDL Code Generation capability in SystemVue provides an easy path from schematic design to hardware. The general HDL code generation flow is shown in the figure below. The flow starts by creating SystemVue sub-network model using synthesizeable Fixed-Point parts from Hardware Design Library, as well as imported HDL code through the HDL cosim block and XilinxIPIntegrator. This sub-network can be then used to generate HDL code for the model inside it. The target of the code generation process can be one of the following:

1. HDL only: generates the HDL files of the synthesizeable fixed point parts inside the sub-network in addition to several additional HDL files for simulation, clock and reset handling.
2. Xilinx FPGA: in addition to the HDL files, a Xilinx ISE project is created to target Xilinx FPGA devices (Virtex 4/5/ and 6)
3. Altera FPGA: in addition to the HDL files, a Quartus II project is created to target Altera FPGA devices (Cyclone IV E/GX,Stratix IV, Stratix V).

For more details, refer to Understanding the Generated HDL.



| | Type | Error |
|---|---|---|
| 1 | | There are no errors or warnings |

This tutorial covers the following examples of generating HDL code from a sub-network that contains:

- Example 1: Fixed point parts from Hardware Design Library.
- Example 2: Imported HDL code using the HDL part.

## Example 1: Fixed point parts from Hardware Design library

This tutorial demonstrates

1. The general HDL code generation flow for sub-networks that are built using Fixed point parts from the Hardware Design Library.
2. Examine the output files and folders of the HDL code generation process.
3. Demonstrate the RTL schematic of the generated HDL design on Xilinx FPGA

Open the HDL_CODE_GEN.wsv workspace. This workspace contains four folders:

1. DFF_Design: This is a code generation example of a synchronous element. It illustrates also the difference between different memory elements. i.e., register, delay and latch.
2. FFT_Design: This is a code generation example of synchronous digital processing parts. It illustrates also latency parameter, which can be found in similar parts such as the CORDIC parts.
3. Gain_Design: This is a code generation example of an asynchronous element. It illustrates also the concept of bit growth as a result of the full precision multiplication.
4. MAC_Design: This is a code generation example of a simple digital design. It illustrates also the concept of overflow and pipelining in digital design.

In the following sections, each of the designs above is described. Then the general steps of HDL code generation flow are listed.

### DFF_Design

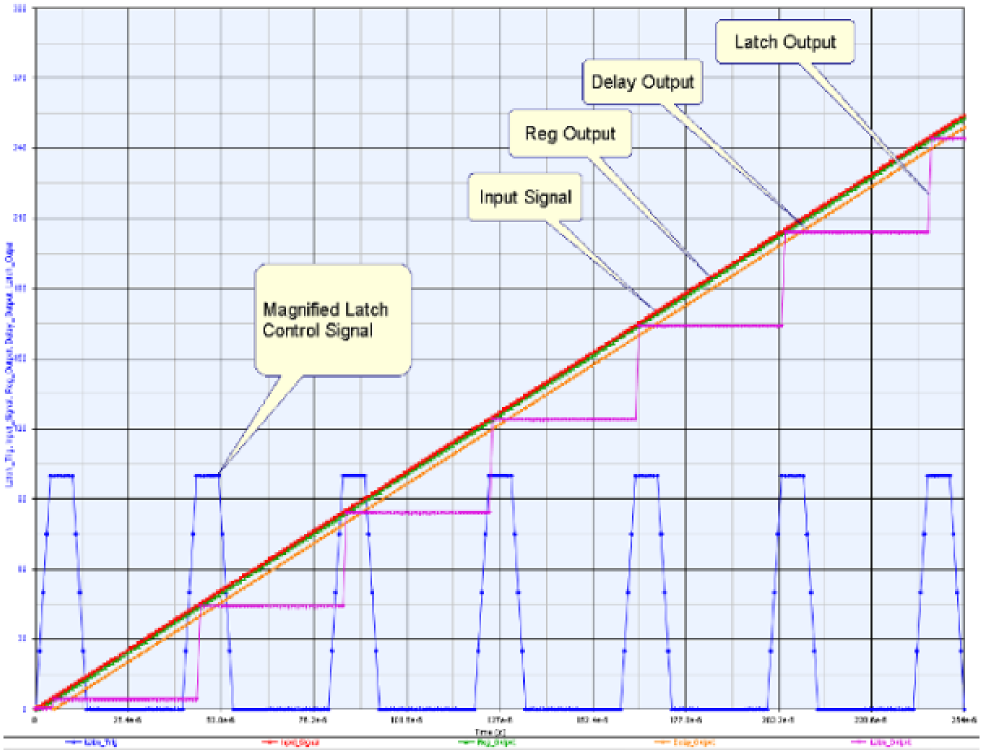
### Design Description

The DFF design is composed of three memory elements:

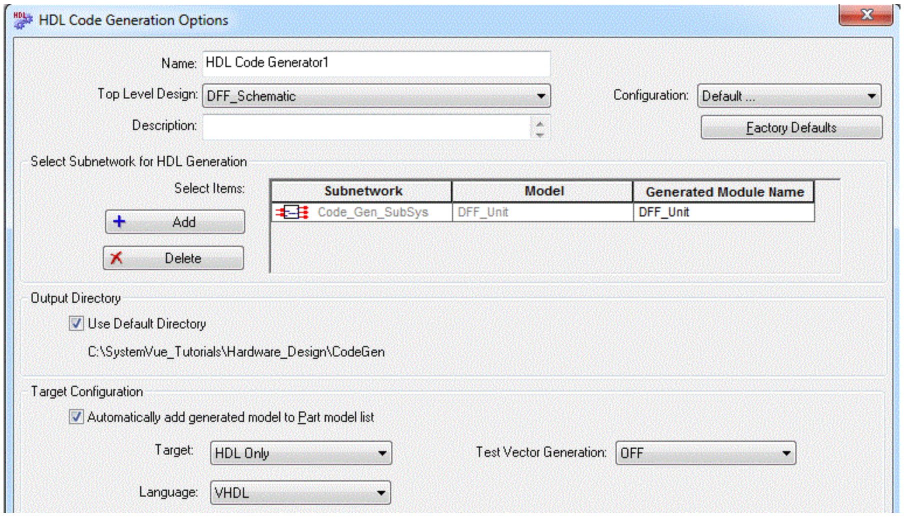
- Delay: this memory element is implemented as a shift register, which upon reset, clear the content of the last register only.
- Register: It is a delay part with delay value = 1 and initial value = 0.
- Latch: This is a register with a control input "latch" to hold the value at its input.

### Tutorial Steps

1. Right click on DFF_Analysis in the workspace tree and select *Run (calculate now)*
2. Check the results on DFF_Graph. The results should be identical to the figure below



3. Double-click on HDL Code Generator1 Add *Code_Gen_Subsys (DFF_Unit)* to the code generation list if it is not on the list yet (refer the following screen capture for the final configuration).
4. Make sure that the following options are set in the HDL Code Generation Options dialog:
    a. The option "Automatically add generated model to Part model list" is checked
    b. The Target is *HDL only* and Test Vector Generation is *OFF*



HDL Code Generation Options

Name: HDL Code Generator1

Top Level Design: DFF_Schematic

Configuration: Default ...

Description:

Select Subnetwork for HDL Generation

Select Items:

| Subnetwork | Model | Generated Module Name |
|---|---|---|
| Code_Gen_SubSys | DFF_Unit | DFF_Unit |

Output Directory

☑ Use Default Directory

C:\SystemVue_Tutorials\Hardware_Design\CodeGen

Target Configuration

☑ Automatically add generated model to Part model list

Target: HDL Only

Test Vector Generation: OFF

Language: VHDL

5. Run the HDL code generation by clicking "Generate"
6. In the DFF_Schematic, double-click on the Code_Gen_SubSys sub-network. Note that the managed model list contains the following models:

a. DFF_Unit

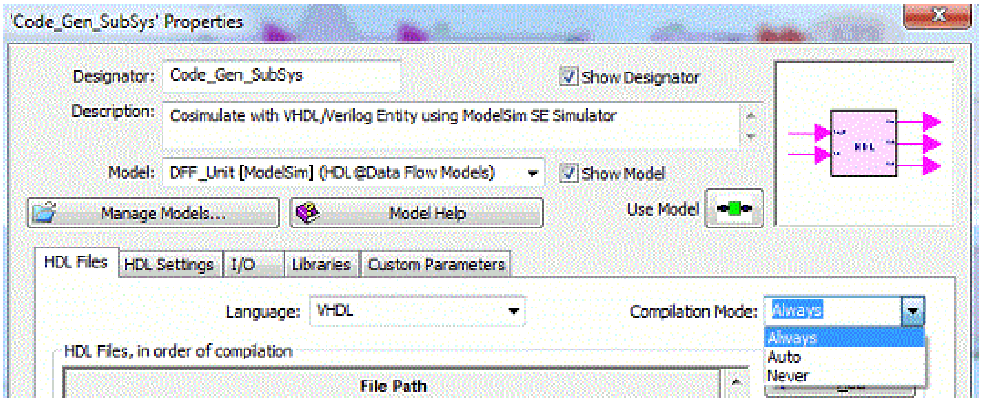
b. DFF_Unit [ModelSim]: Generated automatically.

c. DFF_Unit [Riviera Pro, Fxp]: Generated if the path to Riviera Pro Executable is specified in the Global Options.

d. DFF_Unit [Riviera Pro, Dbl]: Generated if the path to Riviera Pro Executable is specified in the Global Options.
Switch its model to DFF_Unit [ModelSim] or DFF_Unit [Riviera Pro, Fxp] . Make sure that you have a compatible ModelSim or Riviera Pro installed before executing the following step. For more information, refer to System Requirements.

7. Run the Analysis again. SystemVue will co-simulate the generated HDL file and the results will be compared against the output of the memory elements at Diff_Sys1, Diff_Sys2, and Diff_Sys3 sub-networks.

NOTE You may need to set **Compilation Mode** to *Always* as shown below if encountering simulation errors when necessary tools have already been correctly installed:



'Code_Gen_SubSys' Properties

Designator: Code_Gen_SubSys ☑ Show Designator

Description: Cosimulate with VHDL/Verilog Entity using ModelSim SE Simulator

Model: DFF_Unit [ModelSim] (HDL@Data Flow Models) ☑ Show Model

Language: VHDL

Compilation Mode: Always

8. For verification, check whether diff_val1, diff_val2, and diff_val3 are all zero. You will note that diff_val2 is not zero. This is because the first sample of *HDL_DFF_Sig_input_2* in DFF_Data is zero while the first sample of *Ref_DFF_Sig_input_2*is one.

CAUTION The actual implementation of the delay memory will produce slightly different results. Since it is implemented as a shift register of length 5, the reset signal at the beginning of the simulation will reset the last register only in the implemented shift register. Hence, the first output sample of the delay will be zero, followed by four samples at the initial value.

9. Review the generated HDL files in CodeGen\SimpleDesign folder. You will find the following files:

a. DFF_Unit_HDL: It contains the generated HDL source files.

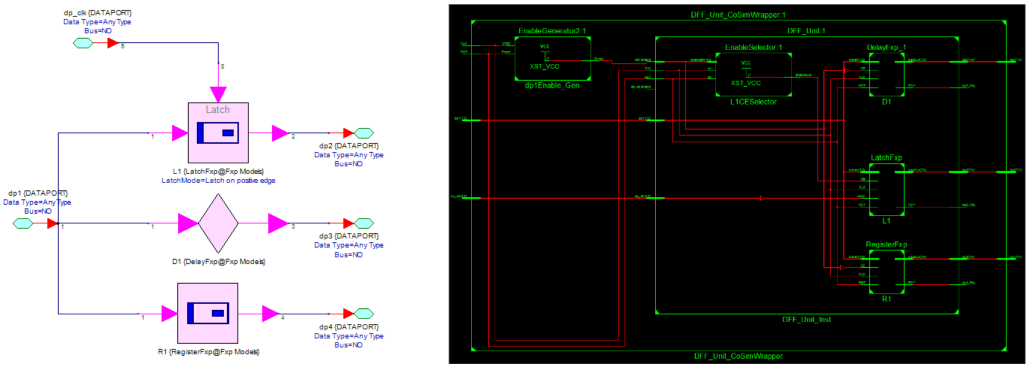
b. DFF_Unit_HDL_Riviera: It contains the Riviera Pro project that was generated for building the XML library of the targeted sub-network.

c. HDL Cosimulation: This folder is generated as a result of cosimulation using ModelSim.

For more information about the generated files and folders refer to Understanding the Generated HDL.

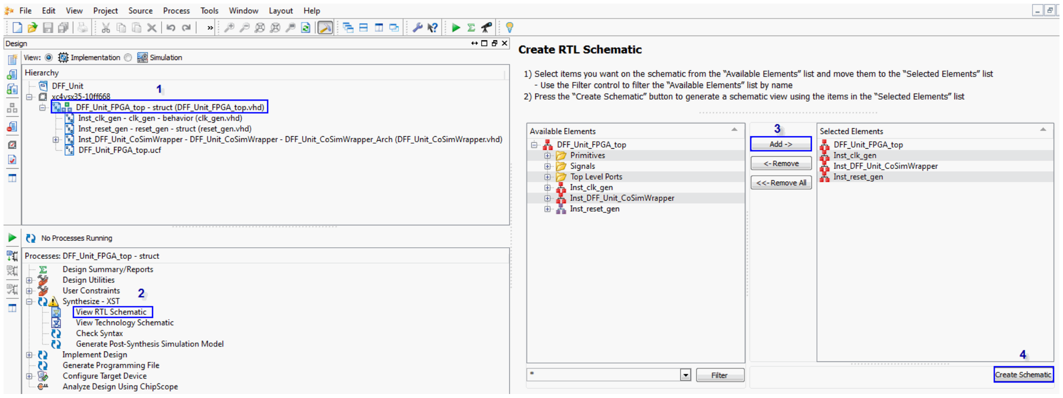
### Design RTL schematic results

The figure below shows DFF_Unit Design in SystemVue and the corresponding RTL schematic of that design in FPGA. Note that the DFF_Unit design is interfaced to a clock and reset generation circuit. The design that combines the DFF_Unit sub-network design and this clock/reset generation circuit is named DFF_Unit_CoSimWrapper as shown on the right side of the figure below. This wrapper design is the top-level design/entity used for co-simulating generated HDL code inside SystemVue.



**NOTE** To generate the RTL schematic diagram using Xilinx ISE tool, you should first generate an HDL code targeting Xilinx FPGA to generate a corresponding ISE project of the design. Open the generated ISE project, and do the following steps below that are also shown in the figure below:

1. Select the top-level design in the project tree
2. Initiate the RTL schematic generation by double-clicking on "View RTL Schematic". Then select "Start the Explorer Wizard"
3. Add all the elements in the "Available Elements" list to the "Selected Elements"
4. Click "Create Schematic"



1) Select items you want on the schematic from the "Available Elements" list and move them to the "Selected Elements" list
- Use the Filter control to filter the "Available Elements" list by name
2) Press the "Create Schematic" button to generate a schematic view using the items in the "Selected Elements" list

## FFT_Design

CAUTION This example generates Verilog files. Since Verilog simulation is only supported by 32-bit ModelSim, make sure you **use 32-bit SystemVue for this exercise** with 32b ModelSim specified for **Global Options** in **HDL Code Generation Options** dialog.

### Design Description

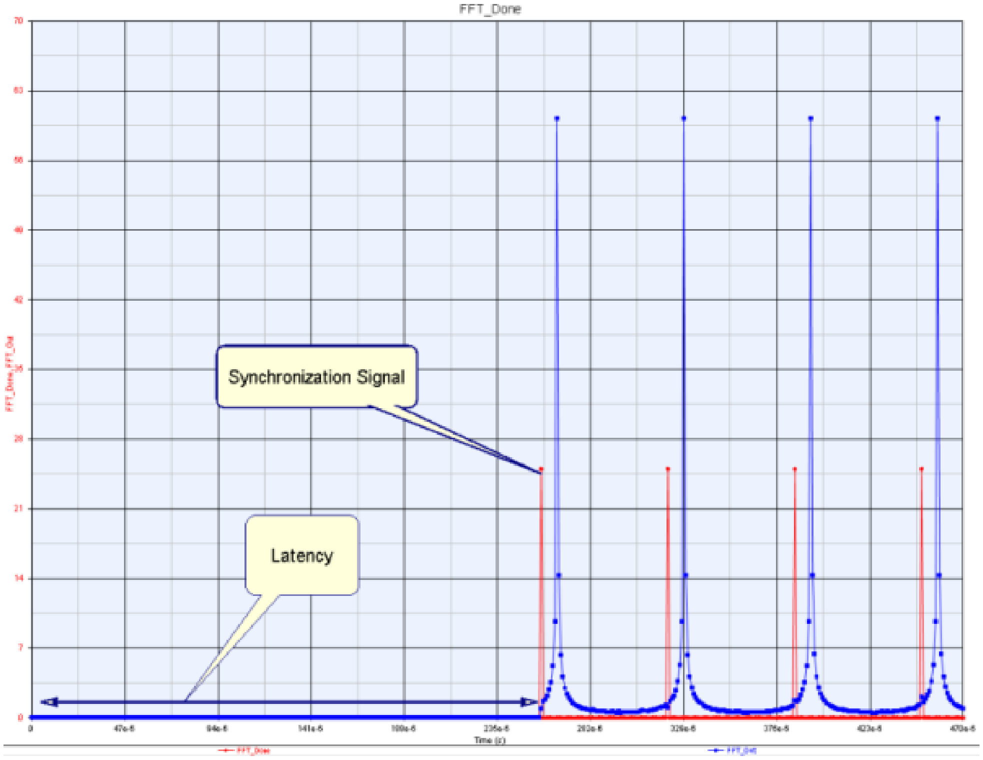
The FFT design is composed of an FFT part that receives a complex sinusoidal signal as an input. The size of FFT is 64. The sampling rate of the simulation is 1000 kHz while the frequency of the generated sinusoidal signal is 128 kHz. Since the FFT size is 64, the sinusoidal signal should appear at the 8th sample (bin) of the 64 samples (bins) at the output.

The complex digital signal processing units such as FFT, FIR and Cordic-based parts, are implemented using pipeline architecture. Hence, there is a latency involved with the actual implementation of these units. The latency specifies the number of clock cycles required after resetting the unit to produce a valid output. The latency can be set manually or automatically. Refer to the part documentation to determine its latency. For the FFT unit in this example, the latency is set to 257.

In order to track the results at the output of the FFT, a separate synchronization unit is implemented which produces a periodic spike every 64 clock cycles, where the first spike is delayed by 257 clock cycles.

### Tutorial Steps

1. Right click on FFT_Analysis in the workspace tree and select *Run (calculate now)*
2. Check the results on FFT_Graph. The results should be identical to the figure below which shows the FFT output and the Synchronization signal.



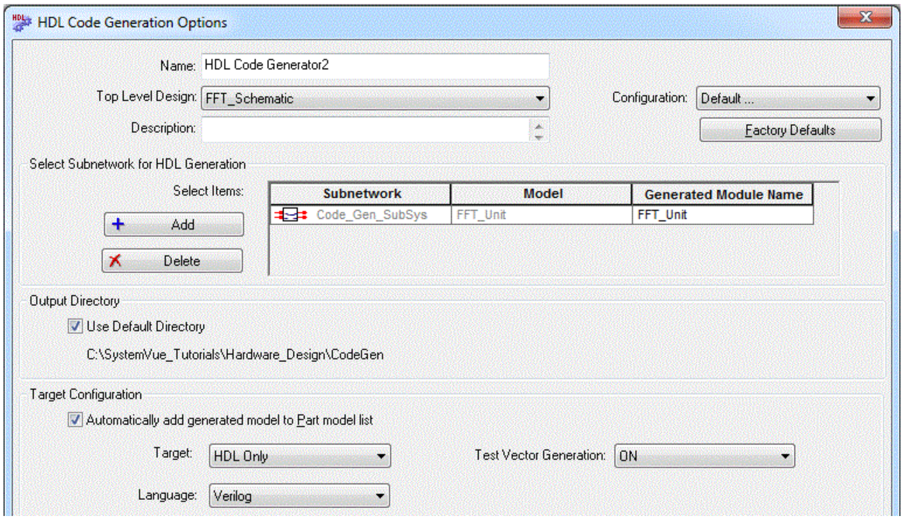
3. Double-click on HDL Code Generator2. Add *Code_Gen_Subsys (FFT_Unit)* to the code generation list if it is not on the list yet (refer the following screen capture for the final configuration).

4. Make sure that the following options are set in the HDL Code Generation Options dialog:

   a. The option "Automatically add generated model to Part model list" is asserted

   b. The Target is *HDL only* and Test Vector Generation is *ON*

**CAUTION** When **Test Vector Generation** is set *ON*, SystemVue run simulation to generate test vectors at first and then generate HDL codes. For complex designs, generating test vectors can be time-consuming. Then generated test vectors are saved as txt files in the generated HDL folder, and they can be read by HDL testbench directly to simulate using ModelSim.



HDL Code Generation Options

Name: HDL Code Generator2

Top Level Design: FFT_Schematic

Configuration: Default ...

Description:

Select Subnetwork for HDL Generation

Select Items:

| Subnetwork | Model | Generated Module Name |
|---|---|---|
| Code_Gen_SubSys | FFT_Unit | FFT_Unit |

Output Directory

☑ Use Default Directory

C:\SystemVue_Tutorials\Hardware_Design\CodeGen

Target Configuration

☑ Automatically add generated model to Part model list

Target: HDL Only

Test Vector Generation: ON

Language: Verilog

5. Run the HDL code generation by clicking Generate
6. Switch the model in the Code_Gen_SubSys sub-network to the code generated model: FFT_Unit [ModelSim] or FFT_Unit [Riviera Pro, Dbl]. Make sure that you have a compatible ModelSim or Riviera Pro installed before executing the following step. For more information, refer to System Requirements.
7. Run FFT_Analysis again. SystemVue will co-simulate the generated HDL file and the results will be compared against the output of the Fixed point model of FFT using AvgSqrErr part.
8. For Verification, make sure that diff_val in FFT_Data dataset is zero.
9. Review the generated HDL files in CodeGen\SimpleDesign folder. Since the Test Vector Generation is ON, the following .txt files are generated inside CodeGen\SimpleDesign\FFT_Unit_HDL folder: Re_In.txt, Im_In.txt, Re_Out_SVU.txt and Im_Out_SVU.txt. These files contain the input and output data samples collected during simulating the schematic that contains the targeted sub-network for HDL code generation.

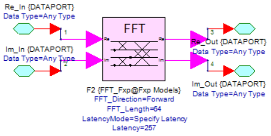
CAUTION

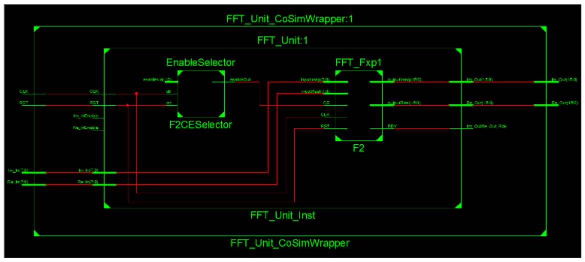
In folder CodeGen\SimpleDesign\FFT_Unit_HDL, a batch file FFT_Unit_SimTB.bat can be executed to run ModelSim simulation automatically using generated input test vectors **Re_In.txt** and **Im_In.txt** , the simulation results at output ports are logged in **Re_Out.txt** and **Im_Out.txt** , which can be compared to the log generated using SystemVue in **Re_Out_SVU.txt** and **Im_Out_SVU.txt**.

For more information about the generated files and folders refer to Understanding the Generated HDL.

### Design RTL schematic results

The figure below shows FFT_Unit Design in SystemVue and the corresponding RTL schematic of that design in FPGA.





## Gain_Design

CAUTION Make sure you also install **Altera Stratix** device package for this exercise, otherwise you will encounter errors during code generation.

### Design Description

The Gain design is composed of a gain part that multiplies the input signal by 2.5 and output the results at full precision. The input precision settings are (WL=3+sign bit, IWL=3+sign bit). The gain part is configured to produce the results of the multiplication at the output at full precision. As a result, the output wordlength at the output will increase. This is known as the bit growth, which is one of the most critical issues encountered in fixed point algorithm design. To determine the precision of the output, determine the least precision required to resemble 2.5. In this case, it is (WL=3, IWL=2). Hence the precision of the output is (WL_input + WL_gain, IWL_input + IWL_gain) = ( 4+3, 4+2 ) = ( 7,6 ).

To verify the results. Perform HDL code generation for this example by following the Tutorial steps below.

### Tutorial Steps

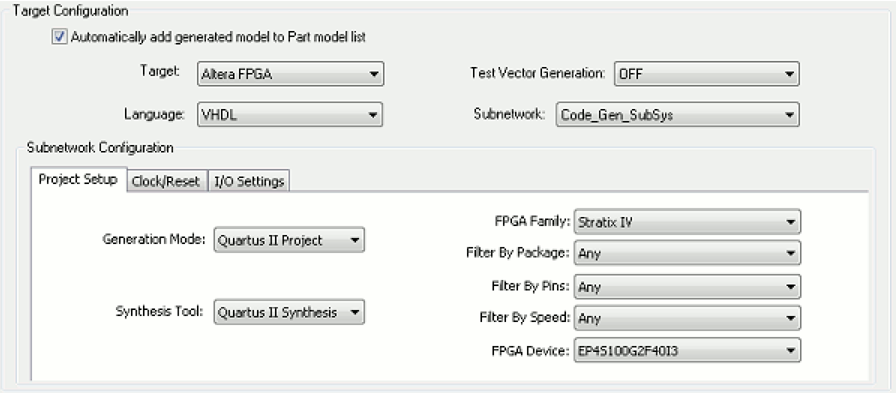
1. Right click on Gain_Analysis in the workspace tree and select *Run (calculate now)*
2. Check the results on Gain_Graph.
3. Double-click on HDL Code Generator 3. Add *Code_Gen_Subsys (Gain_Unit)* to the code generation list.
4. Make sure that the following options are set in the HDL Code Generation Options dialog:
   a. The option "Automatically add generated model to Part model list" is asserted

      CAUTION Make sure that the path to Altera Quartus II executable is specified in the Global Options before performing the next step

   b. The Target is *Altera_FPGA* and The Test Vector Generation is *OFF*

      CAUTION

While HDL code generator can support HDL code generation for multiple sub-networks at the same time when the Target option is set to *HDL_Only*, this is not the case if the code generation target is a specific FPGA such as Altera or Xilinx. In this case, the user has to choose a specific sub-network. For this reason, a new sub-network appears if Xilinx or Altera FPGA target is chosen.



Target Configuration

☑ Automatically add generated model to Part model list

Target: Altera FPGA

Test Vector Generation: OFF

Language: VHDL

Subnetwork: Code_Gen_SubSys

Subnetwork Configuration

Generation Mode: Quartus II Project

Synthesis Tool: Quartus II Synthesis

FPGA Family: Stratix IV

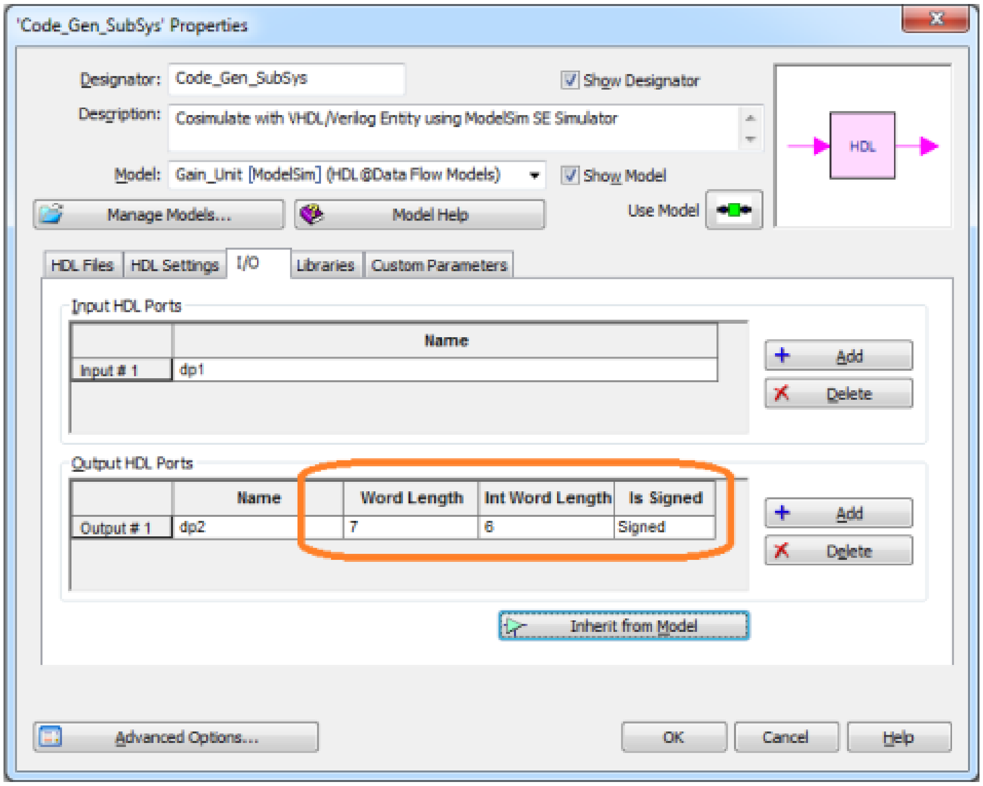
Filter By Package: Any

Filter By Pins: Any

Filter By Speed: Any

FPGA Device: EP4S100G2F40I3

5. Run the HDL code generation by clicking "Generate"
6. Switch the model of the Code_Gen_SubSys part to the code generated model: Gain_Unit [ModelSim] or Gain_Unit [Riviera Pro, Dbl]. Make sure that you have a compatible ModelSim or Riviera Pro installed before executing the following step. For more information, refer to System Requirements.
7. Run the Analysis again. SystemVue will co-simulate the generated HDL file and the results will be compared against the output of the Gain part at Diff_Sys sub-network.
8. To verify, make sure that diff_val is zero. Also, to verify the precision of the Gain output, go to the I/O page of Gain_Unit [ModelSim] (HDL@Data Flow Models) Properties dialog as shown in the figure below:



'Code_Gen_SubSys' Properties

Designator: Code_Gen_SubSys ☑ Show Designator

Description: Cosimulate with VHDL/Verilog Entity using ModelSim SE Simulator

Model: Gain_Unit [ModelSim] (HDL@Data Flow Models) ☑ Show Model

Use Model

Input HDL Ports

| | Name |
|---|---|
| Input # 1 | dp1 |

Output HDL Ports

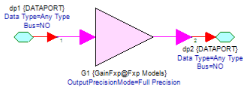
| | Name | | Word Length | Int Word Length | Is Signed |
|---|---|---|---|---|---|
| Output # 1 | dp2 | | 7 | 6 | Signed |

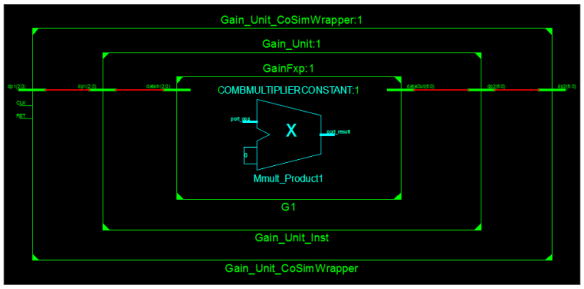
9. Review the generated files in Gain_Unit_FPGA and Gain_Unit_HDL folder.

Try to change the precision of the FloatToFixed point converter at the input of the gain, or change the gain value inside the Gain_unit model. Examine the bit growth at the output.

### Design RTL schematic results

The figure below shows Gain_Unit Design in SystemVue and the corresponding RTL schematic of that design in FPGA.





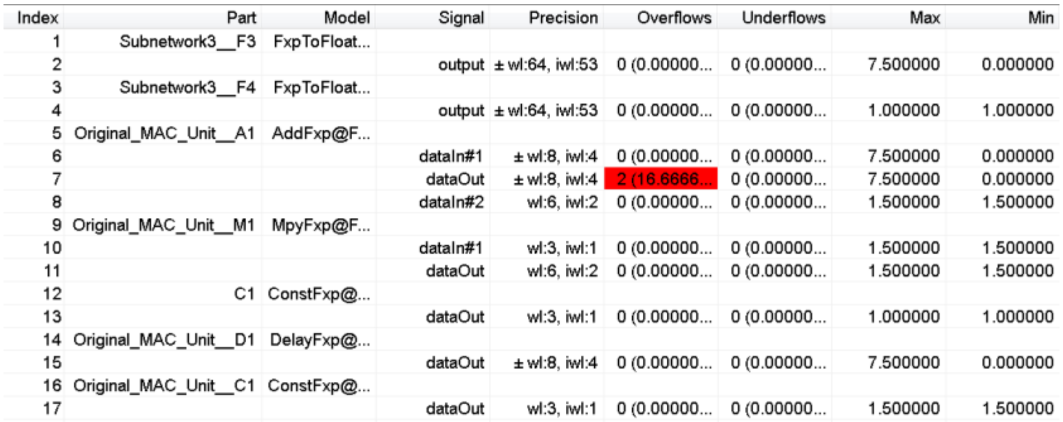
## MAC_Design

### Design Description

This design contains three sub-networks: The original Multiply and accumulate unit (MAC unit), a copy of this unit for code generation (*Code_Gen_Subsys1*) and a sub-network that includes four cascaded MAC units (*Code_Gen_Subsys2*).
Examine the MAC_Unit design. Note that for fixed input value, the output of the multiplier is fixed. Since the input of the model is *one*, the output of the multiplier is also fixed at *1.5*. However, at the adder, the input grows continuously by adding

the fixed value 1.5 to the accumulated one from previous clock cycle. Hence, the output of the adder is incrementing by *1.5* every clock cycle. The output resolution of the adder is Signed (8, 4), an overflow will occur when the adder output change from *7.5* to *9* since *9* is beyond the dynamic range of Signed (8, 4). For a simulation duration of 12 samples, this overflow will occur twice.

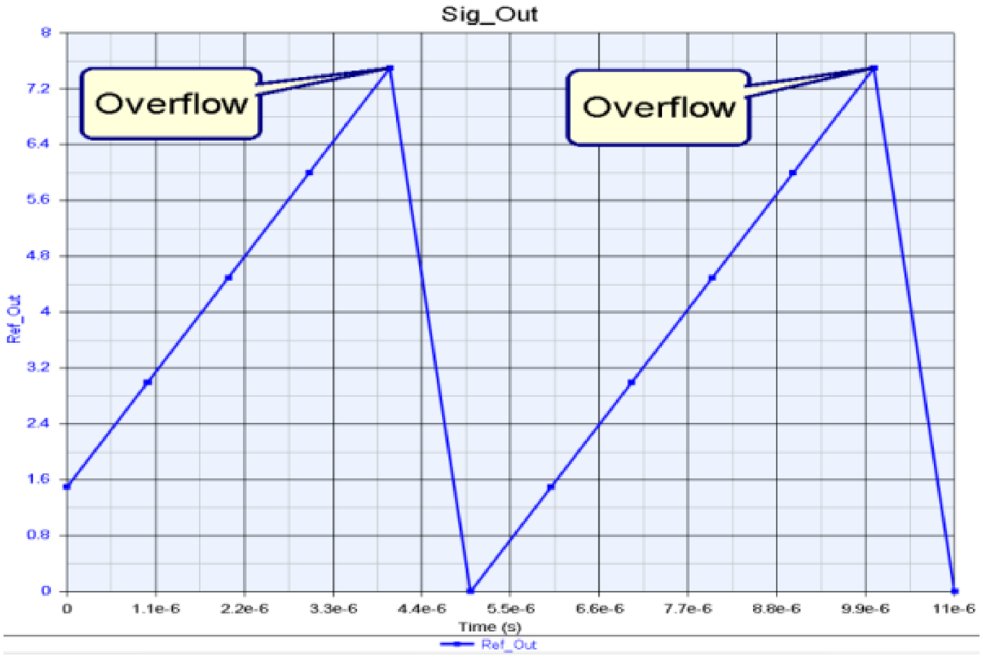
The overflow events can be discovered easily using Fixed Point Analysis table. To produce this table after the simulation, Double-click on MAC_Analysis, go to the Options page and Check "Collect Fixed Point Statistics" option. The content of the table at the end of the simulation is shown below. The overflow is marked in red.



| Index | Part | Model | Signal | Precision | Overflows | Underflows | Max | Min |
|---|---|---|---|---|---|---|---|---|
| 1 | Subnetwork3__F3 | FxpToFloat... | | | | | | |
| 2 | | | output | ± wl:64, iwl:53 | 0 (0.00000... | 0 (0.00000... | 7.500000 | 0.000000 |
| 3 | Subnetwork3__F4 | FxpToFloat... | | | | | | |
| 4 | | | output | ± wl:64, iwl:53 | 0 (0.00000... | 0 (0.00000... | 1.000000 | 1.000000 |
| 5 | Original_MAC_Unit__A1 | AddFxp@F... | | | | | | |
| 6 | | | dataIn#1 | ± wl:8, iwl:4 | 0 (0.00000... | 0 (0.00000... | 7.500000 | 0.000000 |
| 7 | | | dataOut | ± wl:8, iwl:4 | 2 (16.6666... | 0 (0.00000... | 7.500000 | 0.000000 |
| 8 | | | dataIn#2 | wl:6, iwl:2 | 0 (0.00000... | 0 (0.00000... | 1.500000 | 1.500000 |
| 9 | Original_MAC_Unit__M1 | MpyFxp@F... | | | | | | |
| 10 | | | dataIn#1 | wl:3, iwl:1 | 0 (0.00000... | 0 (0.00000... | 1.500000 | 1.500000 |
| 11 | | | dataOut | wl:6, iwl:2 | 0 (0.00000... | 0 (0.00000... | 1.500000 | 1.500000 |
| 12 | C1 | ConstFxp@... | | | | | | |
| 13 | | | dataOut | wl:3, iwl:1 | 0 (0.00000... | 0 (0.00000... | 1.000000 | 1.000000 |
| 14 | Original_MAC_Unit__D1 | DelayFxp@... | | | | | | |
| 15 | | | dataOut | ± wl:8, iwl:4 | 0 (0.00000... | 0 (0.00000... | 7.500000 | 0.000000 |
| 16 | Original_MAC_Unit__C1 | ConstFxp@... | | | | | | |
| 17 | | | dataOut | wl:3, iwl:1 | 0 (0.00000... | 0 (0.00000... | 1.500000 | 1.500000 |

## Tutorial Steps

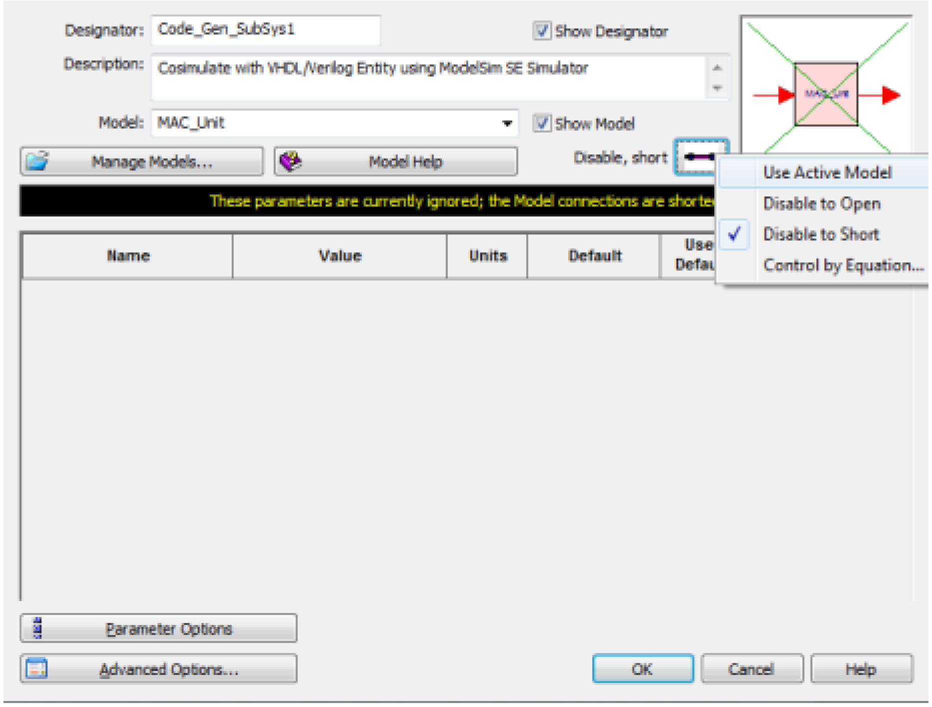
1. Right click on MAC_Analysis in the workspace tree and select *Run (calculate now)*
2. Check the results on MAC_Graph. The results should be identical to the figure below



CAUTION

Since the Overflow option in the adder is set to *SAT_ZERO*, the value of the output upon overflow will be reset to zero. Hence, the accumulator process will be reset. Try setting the Overflow option to *Wrap* and note the difference in the output.

3. Double-click on Code_Gen_SubSys1 sub-network and change the Part behavior from Disable, short to Use Active Model as shown in the figure below:



Designator: Code_Gen_SubSys1

☑ Show Designator

Description: Cosimulate with VHDL/Verilog Entity using ModelSim SE Simulator

Model: MAC_Unit

☑ Show Model

Disable, short

These parameters are currently ignored; the Model connections are shorte

| Name | Value | Units | Default | Use Defau |
|---|---|---|---|---|

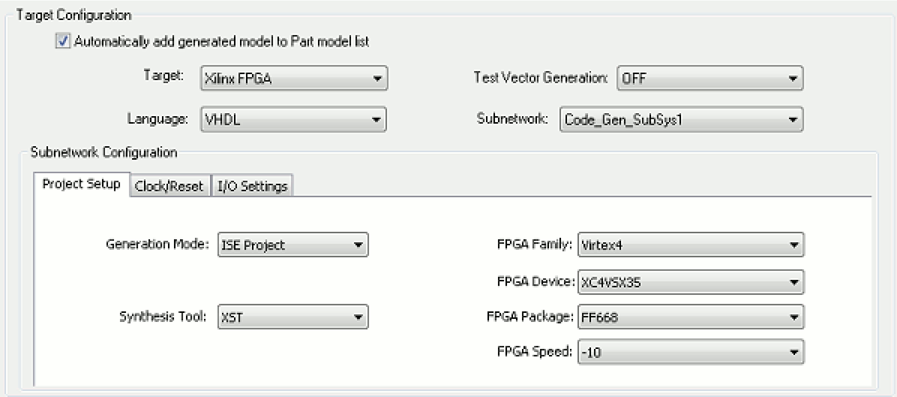
4. Double-click on HDL Code Generator 4. Add *Code_Gen_Subsys1 (MAC_Unit)* to the code generation list.

5. Make sure that the following options are set in the HDL Code Generation Options dialog:

   a. The option "Automatically add generated model to Part model list" is asserted

   b. The Target is *Xilinx FPGA* and Test Vector Generation is *OFF*

   c. Leave the default settings in the Sub-network Configuration section as is.



☑ Automatically add generated model to Part model list

Target: Xilinx FPGA

Test Vector Generation: OFF

Language: VHDL

Subnetwork: Code_Gen_SubSys1

Generation Mode: ISE Project

FPGA Family: Virtex4

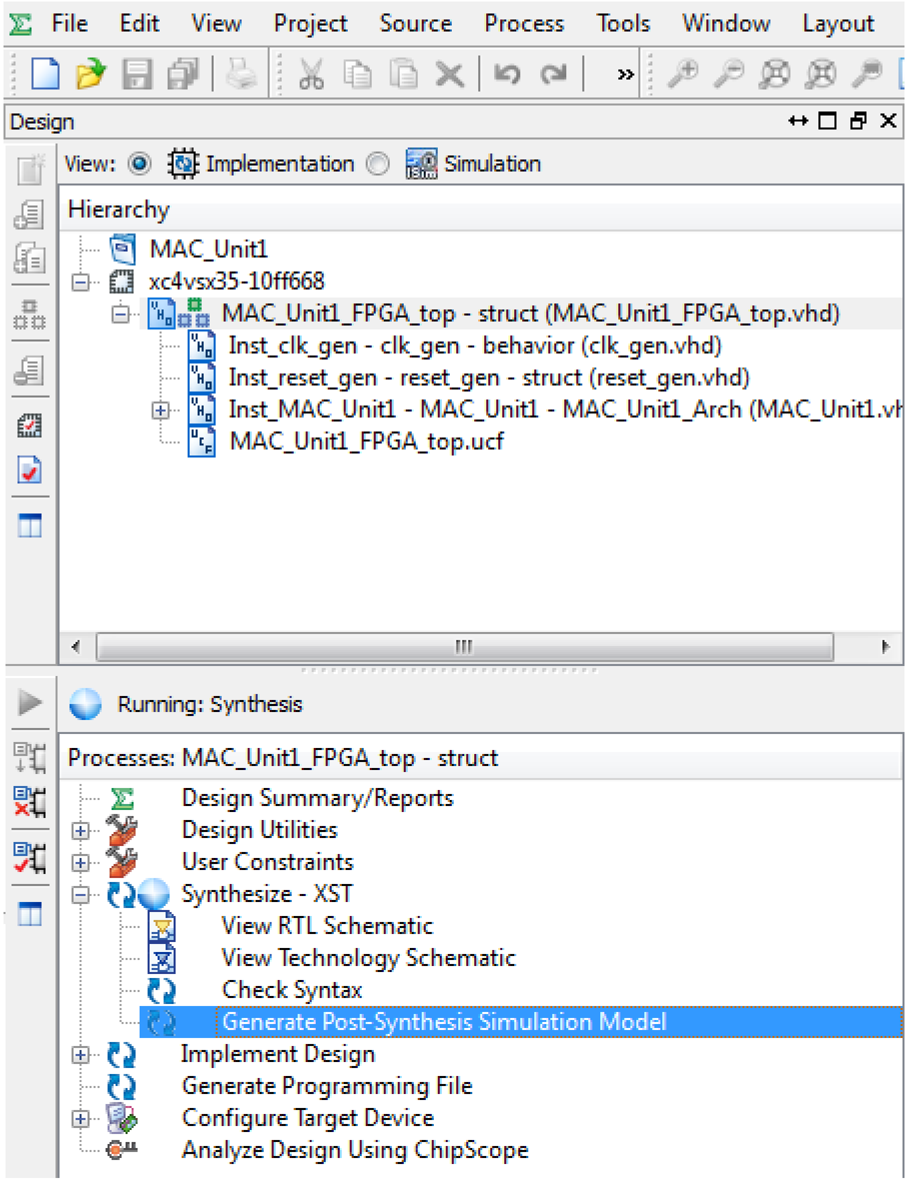
FPGA Device: XC4VSX35

Synthesis Tool: XST

FPGA Package: FF668

FPGA Speed: -10

6. Run the HDL code generation by clicking "Generate"
7. As a result, Xilinx ISE program will be invoked. Review the generated files in MAC_Unit_FPGA and MAC_Unit_HDL folder.
8. To examine the maximum speed at which the design can run on the targeted FPGA, double-click on Generate Post-Synthesis Simulation Model as shown in the figure below


File Edit View Project Source Process Tools Window Layout
Design
View: ● Implementation ○ Simulation
Hierarchy
MAC_Unit1
xc4vsx35-10ff668
MAC_Unit1_FPGA_top - struct (MAC_Unit1_FPGA_top.vhd)
Inst_clk_gen - clk_gen - behavior (clk_gen.vhd)
Inst_reset_gen - reset_gen - struct (reset_gen.vhd)
Inst_MAC_Unit1 - MAC_Unit1 - MAC_Unit1_Arch (MAC_Unit1.vh
MAC_Unit1_FPGA_top.ucf
Running: Synthesis
Processes: MAC_Unit1_FPGA_top - struct
Design Summary/Reports
Design Utilities
User Constraints
Synthesize - XST
View RTL Schematic
View Technology Schematic
Check Syntax
Generate Post-Synthesis Simulation Model
Implement Design
Generate Programming File
Configure Target Device
Analyze Design Using ChipScope

9. Check the Timing Summary at the end of the generated report in the Console window, you will find the following results: *Minimum period: 2.887ns (Maximum Frequency: 346.392MHz)*

CAUTION In Xilinx ISE, you can generate the RTL schematic of the MAC_Unit design by double-clicking on **View RTL Schematic** under **Synthesize-XST** Process.

10. Close the invoked Xilinx ISE program.

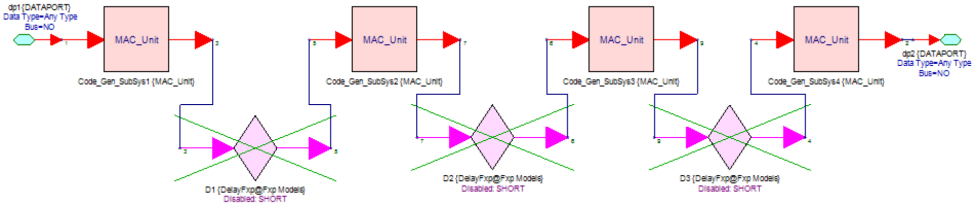
11. Switch the model of the Code_Gen_SubSys1 part to the code generated model: MAC_Unit [ModelSim] or MAC_Unit [Riviera Pro, Dbl]. Make sure that you have a compatible ModelSim or Riviera Pro installed before executing the following step. For more information, refer to System Requirements.

12. Run MAC_Analysis again. SystemVue will co-simulate the generated HDL file and the results will be compared against the output of the original MAC_Unit sub-networks.

13. For verification, make sure that *diff_val* in MAC_Data dataset is zero.

14. Double-click on Code_Gen_SubSys2 sub-network and change the Part behavior from Disable, short to Use Active Model.

15. Check the content of the Cascaded_MAC_Unit. Note that this model uses four cascaded stages of MAC_Unit design. The stages are separated by delays which are currently shorted. These pipelining delays are used to improve the Maximum clock rate that can be achieved with the cascaded design.



**CAUTION** Pipeline registers are usually used to reduce the critical path that can be developed by cascading processing units.

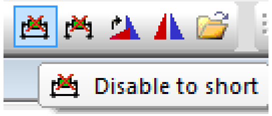
16. Double-click on HDL Code Generator 5. Add *Code_Gen_Subsys2 (MAC_Unit)* to the code generation list and click Generate

17. On the invoked Xilinx ISE program, double-click on Generate Post-Synthesis Simulation Model.

18. Check the Timing Summary at the end of the generated report in the Console window, you will find the following results: *Minimum period: 10.970ns (Maximum Frequency: 91.156MHz)*

19. Close the invoked Xilinx ISE program.

20. Open the Cascaded_MAC_Unit Design and remove the Short status from all the delay elements. This can be done by holding the Ctrl key and clicking on each delay part then lick on the Short icon as shown in the figure below:



21. Double-click on HDL Code Generator 5. Click Generate again.

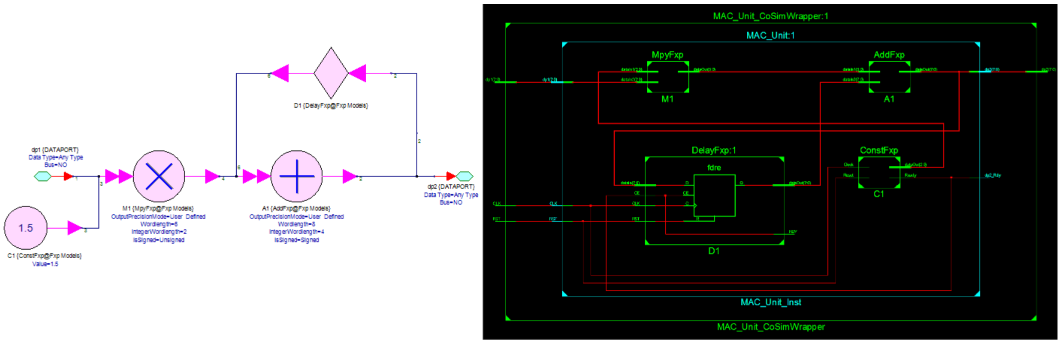
22. On the invoked Xilinx ISE program, double-click on Generate Post-Synthesis Simulation Model.

23. Check the Timing Summary at the end of the generated report in the Console window, you will find the following results: *Minimum period: 4.102ns (Maximum Frequency: 243.795MHz)*

Note how the speed of the cascaded system has increased from 91.156MHz to 243.795MHz as a result of the use of pipelining registers.

### Design RTL schematic results

The figure below shows MAC_Unit Design in SystemVue and the corresponding RTL schematic of that design in FPGA.



### Example 2: Imported HDL code using the HDL part

In addition to the ability of generating HDL code for digital systems that are built with fixed point parts from the Hardware Design library, it is possible also to import customized HDL code into the digital system using the HDL block and to generate HDL code for the entire system. In this example, two designs are considered, a combinational logic design and a sequential logic design.

In many sequential logic designs, a Clock and Reset input ports may exist. In such cases, users have two options:

1. Generate the Clock and Reset signals explicitly in the design and connect them to HDL block.
2. Hide Clock and Reset input ports and let SystemVue generates them automatically upon cosimulation (This option is demonstrated in the sequential logic design case below ).

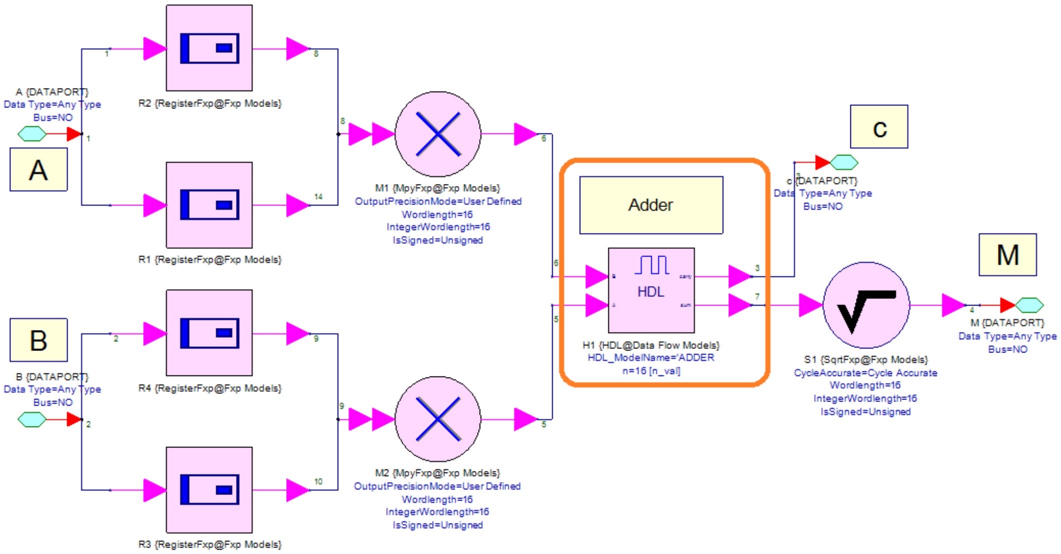
**CAUTION** **To generate HDL code for designs that contain HDL block, a ModelSim license is required since ModelSim is used by this block to compile and verify imported HDL code before HDL code generation for the entire design.**

### Combinational Logic Design

This design takes two inputs A and B and calculates the output M as follows:

$$M = \sqrt{A^2 + B^2}$$

In addition, the output c is asserted in case the summation exceeds the dynamic range of 16 bits. The design is shown in the figure below:



As seen in this design, the n-bit adder circuit is presented using a custom HDL code. The HDL code (VHDL) is shown below:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
--------------------------------------------------------
entity ADDER is
generic(n: natural :=4);
port(    A:    in std_logic_vector(n-1 downto 0);
    B:    in std_logic_vector(n-1 downto 0);
    carry:    out std_logic;
    sum:    out std_logic_vector(n-1 downto 0)
);
end ADDER;
--------------------------------------------------------
architecture behv of ADDER is
-- define a temporary signal to store the result
signal result: std_logic_vector(n downto 0);
begin
    -- the 3rd bit carry the
    result <= ('0' & A)+('0' & B);
    sum <= result(n-1 downto 0);
    carry <= result(n);
end behv;
--------------------------------------------------------
```

To add this HDL code into the design, drag and drop an HDL block from the Hardware Design Library and apply the following settings:

1. HDL Files page

a. Language: select the language based on the type of the HDL code to be added. In this design, the adder code is *VHDL*

**CAUTION** **Mixed Language is not supported for HDL Code Generation.**

b. Add the Adder.vhdfile from the follow location: CodeGen\ExistingHDL\HDL_Source\VHDL\Combinational

**CAUTION** **In the case of multiple files design using VHDL, the compilation order is important, you can change the file order by using Up and Down buttons. The files at the top are compiled first.**

c. Set Compilation Mode to *Always*.

2. HDL Settings page

   a. Top Level Entity/Module: As inferred from the VHDL code, the name of the entity is *ADDER*.

   b. Iteration Time: This is the amount of time for which the HDL simulation is run in ModelSim during each invocation of the HDL model. The value of iteration time must be a positive value. In this example, the iteration time is set to *100 ns*

3. I/O page

   a. The name of the ports must match the names of input and output ports of the top-level entity.

   b. The wordlength of output ports is determined from the HDL code. Since the Wordlength is parameterized (using parameter n), we define a variable n_val in the Equation page "Equation1" in the workspace tree. This variable is then used to represent the value of wordlength of the output port sum.

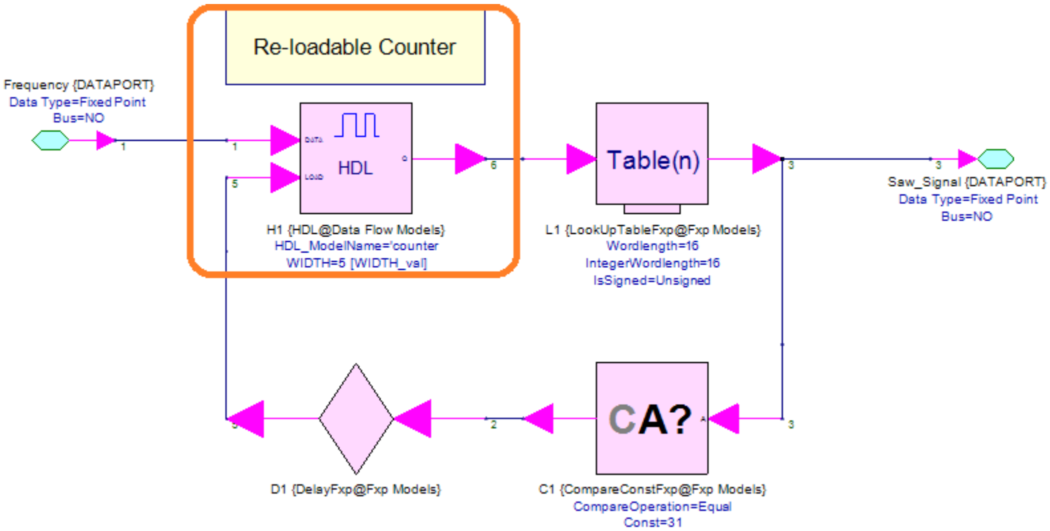
   c. The integer wordlength and the Is Signed settings are determined based on the design. Since the output is always positive, the unsigned setting is chosen. Also, since the input values are all integers, the output of the summation is an integer. Hence, the wordlength = integer wordlength (no fraction part)

4. Custom Parameters page

   a. In this page, we set the values of the entity parameters. In this case, parameter n is set to the value of variable n_val. If parameters are not set, the default value specified in the HDL code will be applied.

### Sequential Logic Design

This design generates a saw signal with configurable period. The values 1 to 32 are stored in the LookUpTable, and a configurable address generator using re-loadable counter generates the memory address to the LookUpTable. Since the number of entries in the LookUpTable is 32, the Wordlength or the address should be 5. Upon accessing the last value of the LookUpTable, a Load control signal is asserted at the Load input port of the counter. The design is shown in the figure below:



As seen in this design, the Re-loadable Counter is presented using a custom HDL code. The HDL code (VHDL) is shown below:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;    -- for the unsigned type

entity COUNTER is
  generic (
    WIDTH : in integer := 32);
  port (
    RST   : in std_logic;
    CLK   : in std_logic;
    LOAD  : in std_logic;
    DATA  : in std_logic_vector(WIDTH-1 downto 0);
    Q     : out std_logic_vector(WIDTH-1 downto 0));
end entity COUNTER;

architecture RTL of COUNTER is
  signal CNT : unsigned(WIDTH-1 downto 0);
begin
  process(RST, CLK) is
  begin
    if RST = '1' then
      CNT <= (others => '0');
    elsif rising_edge(CLK) then
      if LOAD = '1' then
        CNT <= unsigned(DATA); -- type is converted to
unsigned
      else
        CNT <= CNT + 1;
      end if;
    end if;
  end process;

  Q <= std_logic_vector(CNT); -- type is converted back
to std_logic_vector
end architecture RTL;
```

To add this HDL code into the design, drag and drop an HDL block from the Hardware Design Library and apply the following settings:

1. HDL Files page
    a. Language: select the language based on the type of the HDL code to be added. In this design, the adder code is *VHDL*

        CAUTION Mixed Language is not supported for HDL Code Generation.

    b. Add the Counter.vhdfile from the follow location: CodeGen\ExistingHDL\HDL_Source\VHDL\Sequential

        CAUTION In the case of multiple files design using VHDL, the compilation order is important, you can change the file order by using Up and Down buttons. The files at the top are compiled first.

    c. Set Compilation Mode to *Always*.
2. HDL Settings page
    a. Top Level Entity/Module: As can be inferred from the VHDL code, the name of the entity is the *counter*.
    b. Iteration Time: This is the amount of time for which the HDL simulation is run in ModelSim during each invocation of the HDL model. The value of iteration time must be a positive value. Since the top-level entity/module contains a clock then this is set as the clock period of the Auto Generated Clock. So, the iteration time is set to the sampling rate *1000 ns*
3. I/O page
    a. The name of the ports must match the names of input and output ports of the top-level entity.
    b. The wordlength of output ports is determined from the HDL code. Since the Wordlength is parameterized (using parameter WIDTH), we define a variable WIDTH_val in the Equation page "Equation1" in the workspace tree. This variable is then used to represent the value of wordlength of the output port Q.

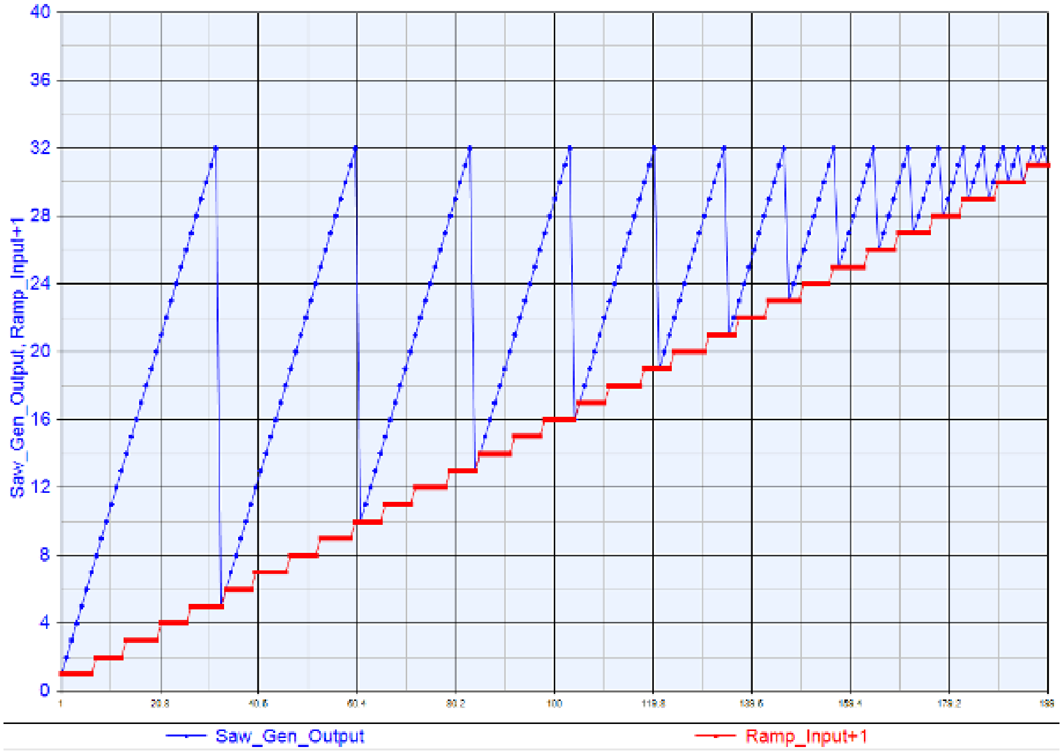
c. The integer wordlength and the Is Signed settings are determined based on the design. Since the output is always positive, unsigned setting is chosen. Also, since the counter incrementing by '1', the output value is always integer. Therefore, integer wordlength = wordlength (no fractional part)

4. Custom Parameters page

   a. In this page, we set the values of the entity parameters. In this case, parameter WIDTH is set to the value of variable WIDTH_val. If parameters are not set, the default value specified in the HDL code will be applied.

After successful HDL code generation for the subnetwork "Subnetwork1", double-click on Subnetwork1 and switch the model to Saw_Sig_Generator [ModelSim] or Saw_Sig_Generator [Riviera Pro, Dbl]. Make sure that you have a compatible ModelSim or Riviera Pro installed before executing the following step. For more information, refer to System Requirements.

Perform HDL cosimulation to verify the functionality of the generated HDL code. The results should be similar to the figure below:



## What Can Go wrong?

**CAUTION** The Fixed-Point sub-network model used to generate HDL, contains only synthesizeable Fixed-Point parts from Hardware Design Library. For example FloatToFxp part or Sink part are not synthesizeable. To ensure that a Fixed-Point part is synthesizeable, see its documentation.

CAUTION The workspace has to be saved in user's local system before HDL code generation.

## C + + to HDL Using Catapult Design

### Prerequisite

- SystemVue 2012.06 or higher with Hardware Design Library
- Catapult C (System Level Synthesis)
- Visual Studio C++

NOTE For software version compatibility, refer to System Requirements

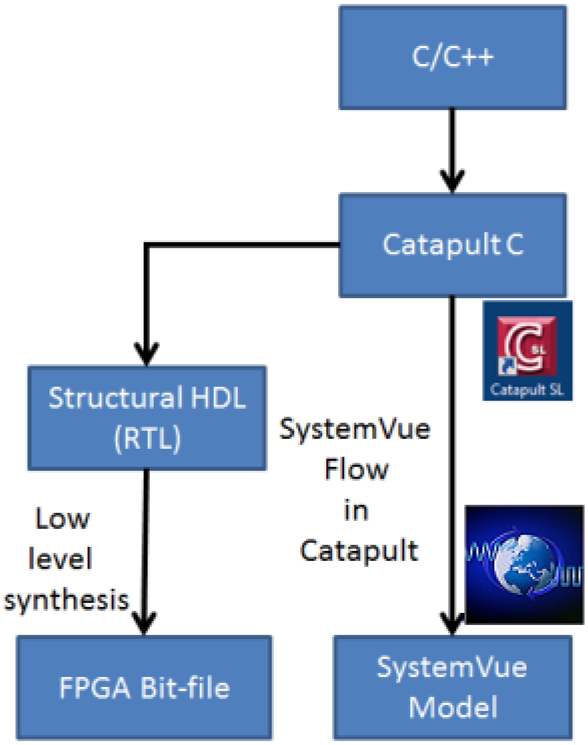
### Associated workspaces

- ConceptToImplementation_Mapper.wsv in Examples\Hardware Design\ConceptToImplementation

  CAUTION We recommend you to copy the complete directory of **Examples\Hardware Design\ConceptToImplementation** to your local directory to avoid file writing permission issues on default installation directory.

### Introduction

This tutorial describes in detail the flow of creating a SystemVue model starting from Bit-true C++ code for HLS Synthesis. Hardware designs are usually developed using bit-accurate datatypes in hardware design languages such as VHDL and Verilog. Bit accurate C/C++ code is based on native C integer types, bit-accurate integer and fixed-point types. The transition from Bit-Accurate C++ code to SystemVue model is done using SystemVue flow in Catapult C HLS software as shown in the figure below.



As seen from the figure, the flow starts with Bit-true C++ code. Catapult C uses this code as an input and generates two outputs: HDL code that can be used later to generate bit files for FPGAs and also a SystemVue Model library (.dll) which is generated through SystemVue flow in Catapult C. This library can be then imported into SystemVue.

In this tutorial, we use the bit-true C++ code mapper.h and mapper.c source files that can be found in Examples\Hardware Design\ConceptToImplementation\source. The contents of these files are listed below.

mapper.h

```
#ifndef _MAPPER_H
#define _MAPPER_H
// Copyright 1983 Keysight Technologies, Inc

#include "ac_fixed.h"


void Mapper(ac_int<6,false> *dataIn, ac_int<4,false>
*realOut, ac_int<4,false> * imagOut);

#endif
```

mapper.cpp

```
// Copyright 1983 Keysight Technologies, Inc
#include "mapper.h"
#include <complex>

const std::complex<int> QAM_64[64] =
{
    std::complex<int>(1, 1),    std::complex<int>(3, 1),
std::complex<int>(1, 3),    std::complex<int>(3, 3),
std::complex<int>(7, 1),    std::complex<int>(5, 1),
std::complex<int>(7, 3),    std::complex<int>(5, 3),
 //s5s4s3s2s1s0=000000
=000001
=000010
=000011
=000100
=000101
=000110                                =000111
    std::complex<int>(1, 7),    std::complex<int>(3, 7),
std::complex<int>(1, 5),    std::complex<int>(3, 5),
std::complex<int>(7, 7),    std::complex<int>(5, 7),
std::complex<int>(7, 5),    std::complex<int>(5, 5),
 //s5s4s3s2s1s0=001000
=001001
=001010
=001011
=001100
=001101
=001110                        =001111
    std::complex<int>(15, 1),    std::complex<int>(13, 1
),    std::complex<int>(15, 3),    std::complex<int>(13,
3),    std::complex<int>(9, 1),    std::complex<int>(11,
1),    std::complex<int>(9, 3),    std::complex<int>(11,
3),
 //s5s4s3s2s1s0=010000
=010001
=010010
=010011
=010100
=010101
=010110                        =010111
    std::complex<int>(15, 7),    std::complex<int>(13, 7
),    std::complex<int>(15, 5),    std::complex<int>(13,
5),    std::complex<int>(9, 7),    std::complex<int>(11,
7),    std::complex<int>(9, 5),    std::complex<int>(11,
5),
 //s5s4s3s2s1s0=011000
=011001
=011010
=011011
=011100
=011101
=011110                        =011111
```

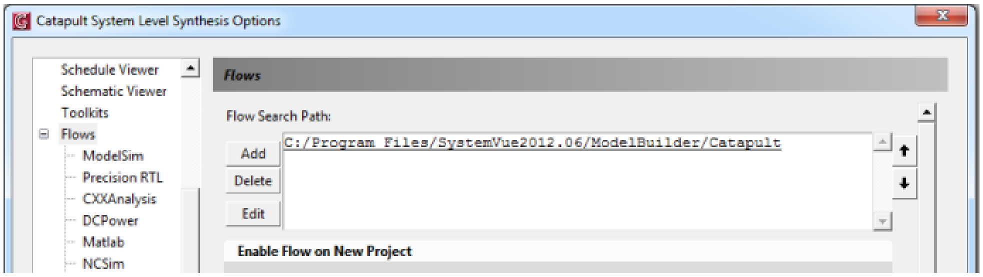
```
    std::complex<int>(1, 15),    std::complex<int>(3, 15
),    std::complex<int>(1, 13),    std::complex<int>(3,
13),    std::complex<int>(7, 15),    std::complex<int>(5
, 15),    std::complex<int>(7, 13),    std::complex<int>
(5, 13),
 //s5s4s3s2s1s0=100000
=100001
=100010
=100011
=100100
=100101
=100110                        =100111
    std::complex<int>(1, 9),    std::complex<int>(3, 9),
std::complex<int>(1, 11),    std::complex<int>(3, 11),
std::complex<int>(7, 9),    std::complex<int>(5, 9),
std::complex<int>(7, 11),    std::complex<int>(5, 11),
 //s5s4s3s2s1s0=101000
=101001
=101010
=101011
=101100
=101101
=101110                        =101111
    std::complex<int>(15, 15),    std::complex<int>(13,
15),    std::complex<int>(15, 13),    std::complex<int>(
13, 13),    std::complex<int>(9, 15),    std::complex<in
t>(11, 15),    std::complex<int>(9, 13),    std::
complex<int>(11, 13),
 //s5s4s3s2s1s0=110000
=110001
=110010
=110011
=110100
=110101
=110110                            =110111
    std::complex<int>(15, 9),    std::complex<int>(13, 9
),    std::complex<int>(15, 11),    std::complex<int>(13
, 11),    std::complex<int>(9, 9),    std::complex<int>(
11, 9),    std::complex<int>(9, 11),    std::complex<int
>(11, 11)
 //s5s4s3s2s1s0=111000
=111001
=111010
=111011
=111100
=111101
=111110                        =111111
};

#pragma design top
void Mapper(ac_int<6,false> *dataIn, ac_int<4,false>
*realOut, ac_int<4,false> * imagOut)
{
    unsigned short index = *dataIn;
    *realOut = QAM_64[index].real();
    *imagOut = QAM_64[index].imag();
}
```

## SystemVue Flow in Catapult C Example

The following sections describe in detail the steps required to transform a Bit-True C/C++ code into a SystemVue Model. Before moving to the next sections, copy the mapper.h and mapper.cpp files into another location, we will refer to it in this tutorial as <CatapultExampleFolder>.

## Setting SystemVue Flow in Catapult SL

1. Start Catapult SL software
2. Open Tools->Set Options .... and select Flows at the left side of the dialog box
3. Click on Add and add the directory <SystemVue Installation>\ModelBuilder\Catapult in the Flow Search Path, and click Apply & Save, click OK to close the dialog. Close Catapult and restart it.



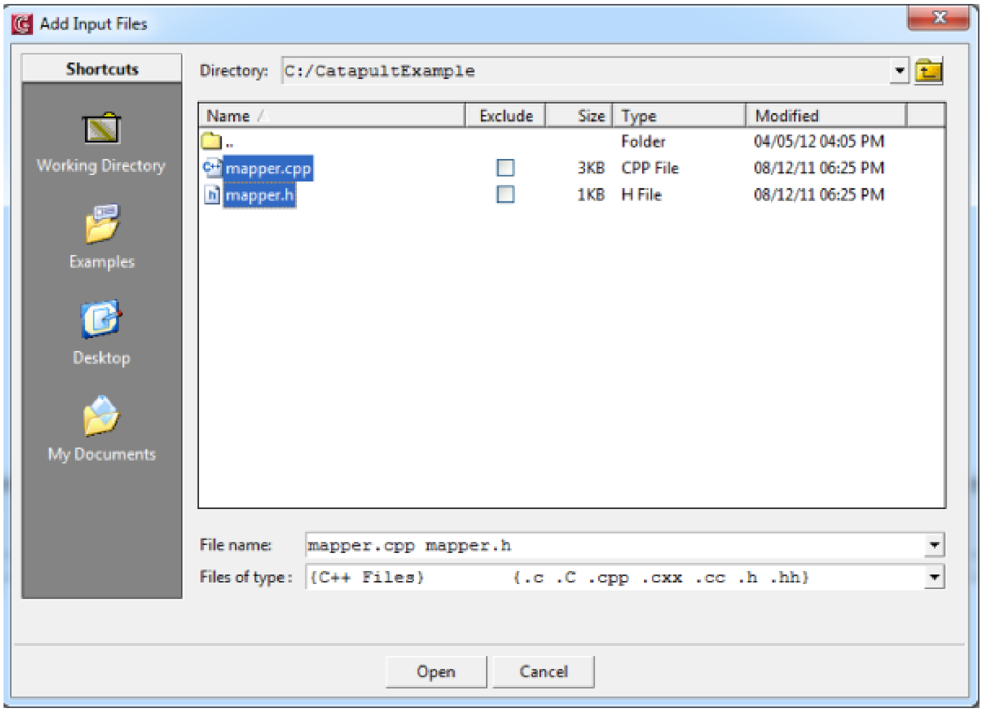
Flow Search Path: C:/Program Files/SystemVue2012.06/ModelBuilder/Catapult

## Creating Catapult C Project

1. Prepare a project folder in a write-permissible location. We will refer to that folder as <CatapultExampleFolder>. Ignore this step if you have already done so.

   **CAUTION** Make sure that you have copied the contents of **<SystemVue Installation>\Examples\Hardware Design\ConceptToImplementation\source** to **<CatapultExampleFolder>**

2. Click File -> Set Working Directory ... and set <CatapultExampleFolder> as working directory
3. Click File -> New -> Project and name the project *Mapper* and click OK
4. Click on Add Input Files in the Task Bar and add mapper.h and mapper.cpp that have been copied to <CatapultExampleFolder>. See the figure below



Directory: C:/CatapultExample

| Name | Exclude | Size | Type | Modified |
|---|---|---|---|---|
| .. | | | Folder | 04/05/12 04:05 PM |
| mapper.cpp | □ | 3KB | CPP File | 08/12/11 06:25 PM |
| mapper.h | □ | 1KB | H File | 08/12/11 06:25 PM |

File name: mapper.cpp mapper.h

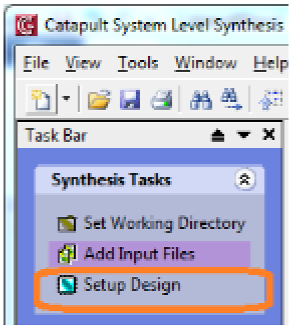
Files of type: (C++ Files) {.c .C .cpp .cxx .cc .h .hh}

5. Click View -> Other Windows -> Flow Manager, you will see SystemVue available in Flow Manager
6. Click on SystemVue in the Flow Manager window and then click Enable
7. Make sure to Set SystemVue model directory to <CatapultExampleFolder>\SystemVueFlow and click Apply. This will create SystemVue Folder automatically if it is not created in <CatapultExampleFolder>.

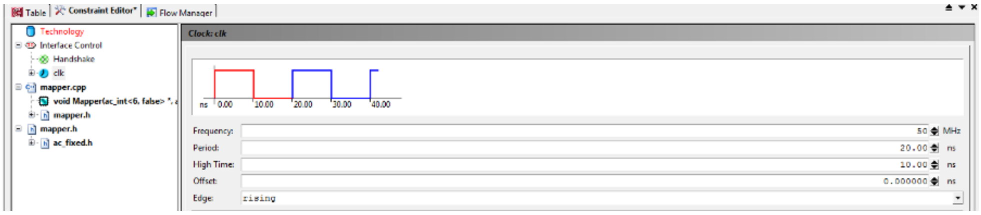
## Setting Up the Design

**CAUTION** Make sure that you are using a compatible compiler with Catapult-C. The compiler used by Catapult-C can be set at **Tools -> Set Options... -> Input** and then set the appropriate **Type** at **Compiler Home** Section. Refer to Catapult-C Release Documentation for the compatible compiler versions

1. Under Task Bar click Setup Design

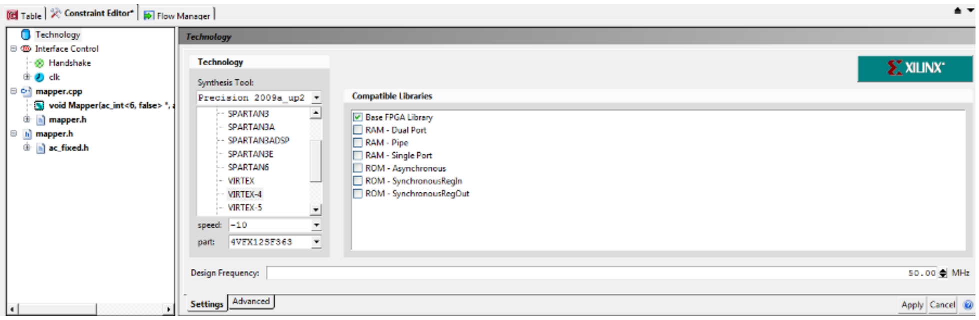


2. Select clk and Set Design Frequency to 50 MHz



Frequency: 50 MHz
Period: 20.00 ns
High Time: 10.00 ns
Offset: 0.000000 ns
Edge: rising

3. Select Technology -> Xilinx -> VIRTEX-4 -> Base FPGA Library and click Apply



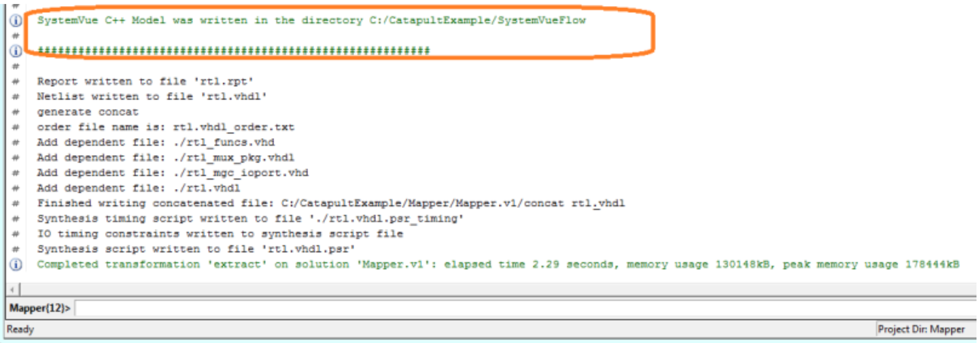
Synthesis Tool: Precision 2009a_up2
speed: -10
part: 4VFX12SF363

☑ Base FPGA Library
☐ RAM - Dual Port
☐ RAM - Pipe
☐ RAM - Single Port
☐ ROM - Asynchronous
☐ ROM - SynchronousRegIn
☐ ROM - SynchronousRegOut

Design Frequency: 50.00 MHz

## Running Catapult Design Flow

1. Click On Generate RTL to generate RTL
2. This will generate VHDL under Output Files under Project Files
3. Also, at the end of the flow, SystemVue C++ model is generated as indicated by the generation log. See the figure below.



```
#
   SystemVue C++ Model was written in the directory C:/CatapultExample/SystemVueFlow
#
   ###########################################################
#
#  Report written to file 'rtl.rpt'
#  Netlist written to file 'rtl.vhdl'
#  generate concat
#  order file name is: rtl.vhdl_order.txt
#  Add dependent file: ./rtl_funcs.vhd
#  Add dependent file: ./rtl_mux_pkg.vhdl
#  Add dependent file: ./rtl_mgc_ioport.vhd
#  Add dependent file: ./rtl.vhdl
#  Finished writing concatenated file: C:/CatapultExample/Mapper/Mapper.v1/concat rtl.vhdl
#  Synthesis timing script written to file './rtl.vhdl.psr_timing'
#  IO timing constraints written to synthesis script file
#  Synthesis script written to file 'rtl.vhdl.psr'
   Completed transformation 'extract' on solution 'Mapper.v1': elapsed time 2.29 seconds, memory usage 130148kB, peak memory usage 178444kB
Mapper(12)>
Ready                                                                Project Dir: Mapper
```

4. As a result, the followings are generated under <CatapultExampleFolder>\SystemVueFlowfolder:
    a. source folder: it contains the source code for the SystemVue model of the mapper.
    b. Configure-for-vs2012.bat: is a batch file for generating Visual Studio solutions (32 and 64-bits) using CMake.
5. Double-click on Configure-for-vs2012.bat to Run this batch file. As a result, two folders are generated:
    a. build-win32-vs2012: contains the 32-bit SystemVue model visual studio solution.

b. build-win64-vs2012: contains the 64-bit SystemVue model visual studio solution.

### Building SystemVue Model in Visual Studio

1. To build SystemVue model for SystemVue 32-bit (64-bits), double-click to open the *SystemVueFlow* solution under build-win32-vs2012 (build-win64-vs2012). It should start Visual Studio 2012 with the solution loaded.
2. Inside Visual Studio 2012, select the *Release* Build configuration
3. Right-click on the *INSTALL* project and select Rebuild.
   This will produce a new folder output-vs2012 inside <CatapultExampleFolder>\SystemVueFlow directory. This folder contains: Mapper.dll file and Mapper.pdb file under Release-SystemVue-Win32 ( Release-SystemVue-Win64) folder.

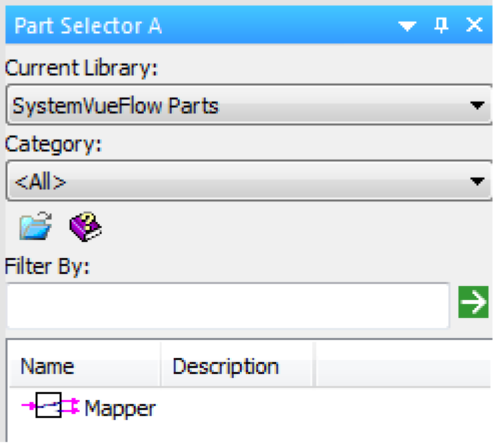
CAUTION **SystemVueFlow** in Examples\Hardware Design\ConceptToImplementation contains a reference of the correctly generated dll file, you can compare the generated dll file against the ones stored in that folder

### Using Generated Library in SystemVue

1. Copy ConceptToImplementation_Mapper.wsv in Examples\Hardware Design\ConceptToImplementation to <CatapultExampleFolder>
2. Open the workspace ConceptToImplementation_Mapper.wsv.
3. Drag and drop the generated Mapper.dll to SystemVue. Alternatively, Go to Tools -> Library Manager and add the library by click Add From File ...

CAUTION Make sure that there is no older version of this .dll file in the library list. If you found one, remove it before adding the generated **Mapper.dll** file

4. You will be able to see SystemVueFlow Parts in the Part Selector A List.


Part Selector A

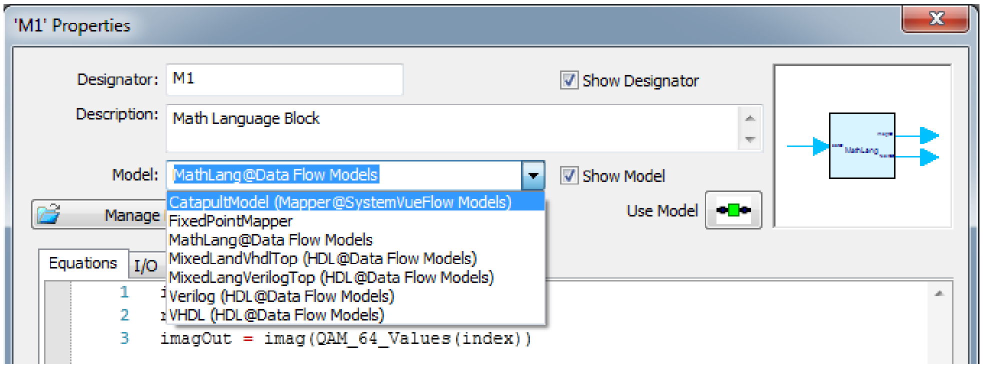
Current Library: SystemVueFlow Parts

Category: <All>

Filter By:

| Name | Description |
|---|---|
| Mapper | |

5. The part M1 in the design was already setup to have Mapper @ SystemVueFlow models in its managed model list. See the figure below.


'M1' Properties

Designator: M1

☑ Show Designator

Description: Math Language Block

Model: MathLang@Data Flow Models

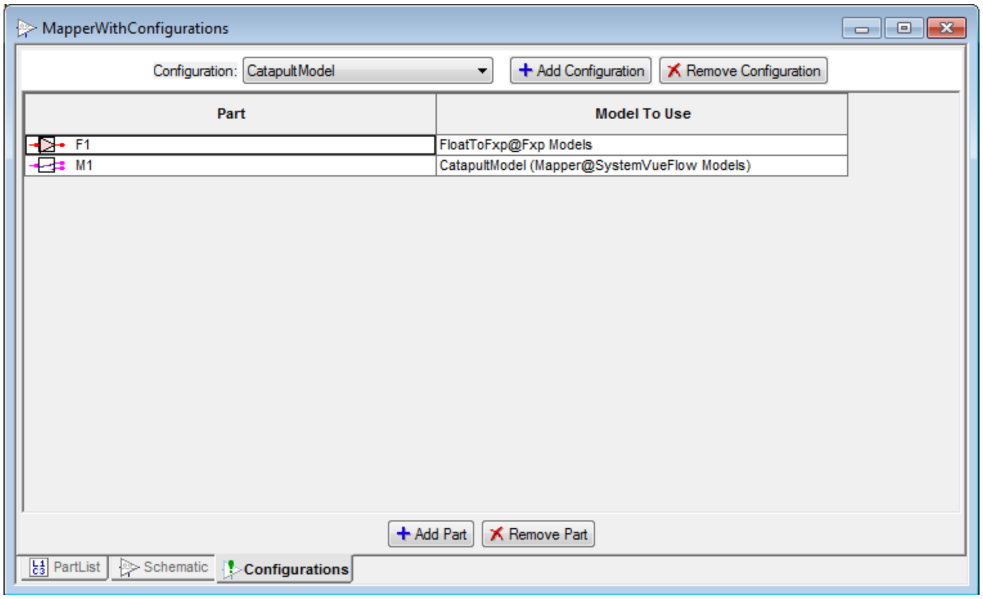
☑ Show Model

Manage
Use Model
Equations
I/O
MathLang
CatapultModel (Mapper@SystemVueFlow Models)
FixedPointMapper
MathLang@Data Flow Models
MixedLandVhdlTop (HDL@Data Flow Models)
MixedLangVerilogTop (HDL@Data Flow Models)
Verilog (HDL@Data Flow Models)
VHDL (HDL@Data Flow Models)

```
i
i
imagOut = imag(QAM_64_Values(index))
```

6. Run the simulation using Catapultconfiguration and verify its functionality compared to other models using with other configurations.

**CAUTION** In Catapult configuration, the part **M1** uses **Mapper@SystemVueFlow** Model and the part **F1** uses **FloatToFxp@Fxp** Model


MapperWithConfigurations

Configuration: CatapultModel

| Part | Model To Use |
|---|---|
| F1 | FloatToFxp@Fxp Models |
| M1 | CatapultModel (Mapper@SystemVueFlow Models) |

## Fixed Point Optimization

### Prerequisite

- SystemVue with Hardware Design Library

**NOTE** For software version compatibility, refer to System Requirements

### Associated workspaces

- OptimizingCordic.wsv in Examples\Tutorials\Hardware_Design\Demos\OptimizingCordic
- RecursiveConstructDemo.wsv in Examples\Tutorials\Hardware_Design\Demos\RecursiveConstructs

- MovingAverage.wsv in Examples\Tutorials\Hardware_Design\Demos\FixedPointStatistics

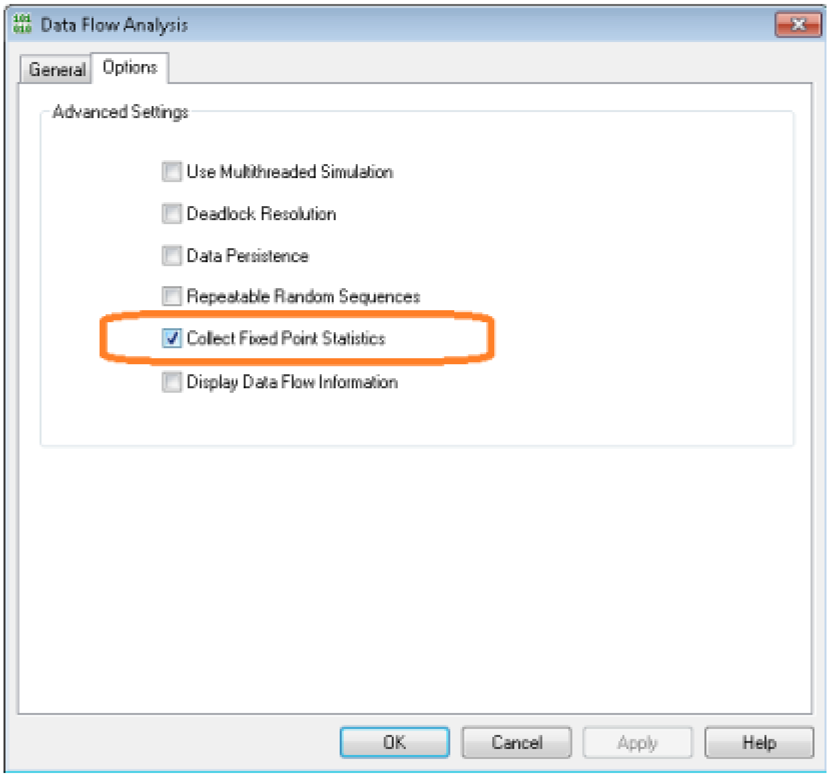
**CAUTION** We recommend you to copy the complete directory of **Examples\Tutorials\Hardware_Design** to your local directory to avoid file writing permission issues on default installation directory.

## Introduction

FPGA-based implementations of digital signal processing algorithms are usually done based on fixed point arithmetic to strive for cost and speed. To reduce cost, it is necessary to use the fewest number of bits possible to represent signals in the design. To improve speed, the number of iterations or stages in iterative algorithms has to be as low as possible. Simulation-based optimization is one method to determine the wordlength and other fixed point-related parameters. This tutorial explains how to use the following features in SystemVue to optimize several fixed point parameters in digital designs: Fixed point analysis table and Sweep analysis.

## Fixed Point Analysis Table

Fixed point table is created automatically after simulating a design that contains fixed point parts and when data flow analysis options are set properly to collect fixed point analysis data as shown in the figure below:



Advanced Settings

□ Use Multithreaded Simulation

□ Deadlock Resolution

□ Data Persistence

□ Repeatable Random Sequences

☑ Collect Fixed Point Statistics

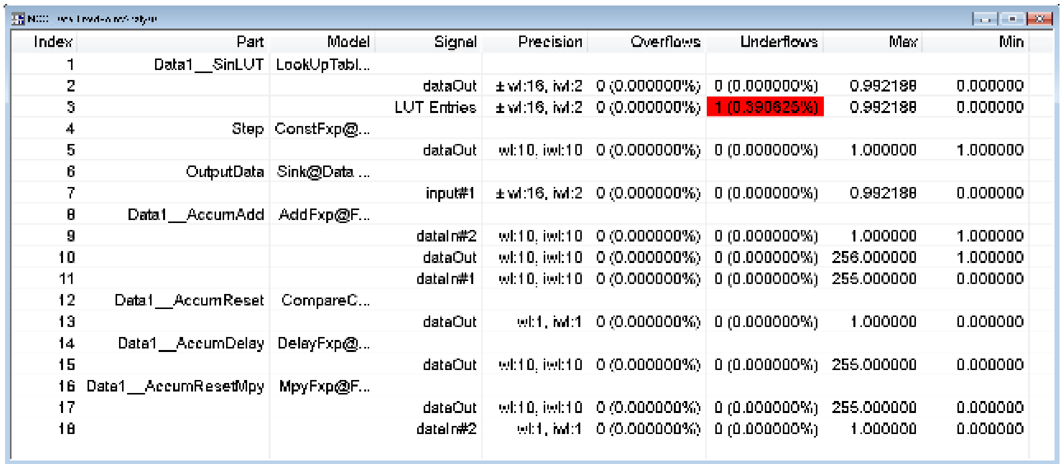
□ Display Data Flow Information

The fixed point analysis table contains 8 columns:

- Part- the instance name of the design block.

- Model - the model used in the Part.
- Signal - the name of the fixed point signal.
- Precision - the fixed point number representation ( ± indicates signed, wl denotes the register word length, iwl denotes the length of the integer part).
- Overflows - shows the total number and percentage of overflows on the signal.
- Underflows - shows the total number and percentage of underflows on the signal.
- Max - the maximum value of the signal.
- Min - the minimum value of the signal.

Overflows and underflows are generally undesirable, they are shown in red. The figure is an example fixed point analysis table:

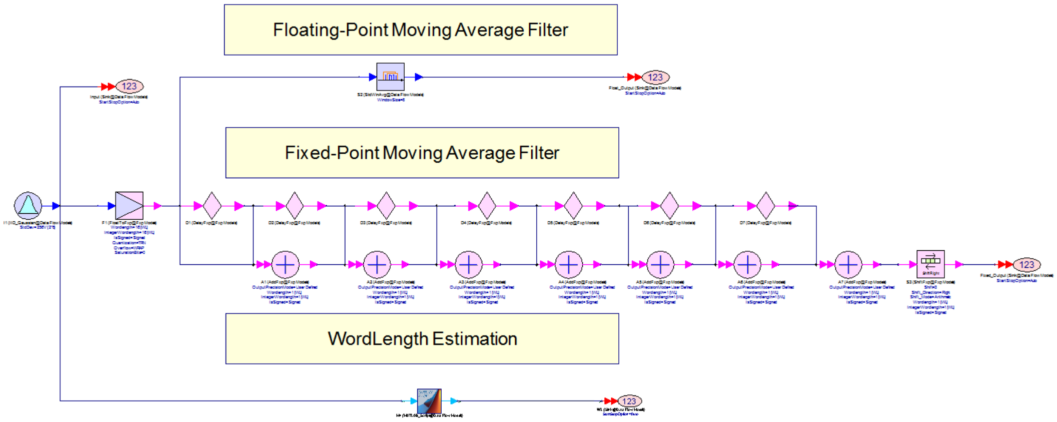


| Index | Part | Model | Signal | Precision | Overflows | Underflows | Max | Min |
|---|---|---|---|---|---|---|---|---|
| 1 | Data1__SinLUT | LookUpTabl... | | | | | | |
| 2 | | | dataOut | ± wl:16, iwl:2 | 0 (0.000000%) | 0 (0.000000%) | 0.992188 | 0.000000 |
| 3 | | | LUT Entries | ± wl:16, iwl:2 | 0 (0.000000%) | 1 (0.390625%) | 0.992188 | 0.000000 |
| 4 | Step | ConstFxp@... | | | | | | |
| 5 | | | dataOut | wl:10, iwl:10 | 0 (0.000000%) | 0 (0.000000%) | 1.000000 | 1.000000 |
| 6 | OutputData | Sink@Data ... | | | | | | |
| 7 | | | input#1 | ± wl:16, iwl:2 | 0 (0.000000%) | 0 (0.000000%) | 0.992188 | 0.000000 |
| 8 | Data1__AccumAdd | AddFxp@F... | | | | | | |
| 9 | | | dataIn#2 | wl:10, iwl:10 | 0 (0.000000%) | 0 (0.000000%) | 1.000000 | 1.000000 |
| 10 | | | dataOut | wl:10, iwl:10 | 0 (0.000000%) | 0 (0.000000%) | 256.000000 | 1.000000 |
| 11 | | | dataIn#1 | wl:10, iwl:10 | 0 (0.000000%) | 0 (0.000000%) | 255.000000 | 0.000000 |
| 12 | Data1__AccumReset | CompareC... | | | | | | |
| 13 | | | dataOut | wl:1, iwl:1 | 0 (0.000000%) | 0 (0.000000%) | 1.000000 | 0.000000 |
| 14 | Data1__AccumDelay | DelayFxp@... | | | | | | |
| 15 | | | dataOut | wl:10, iwl:10 | 0 (0.000000%) | 0 (0.000000%) | 255.000000 | 0.000000 |
| 16 | Data1__AccumResetMpy | MpyFxp@F... | | | | | | |
| 17 | | | dataOut | wl:10, iwl:10 | 0 (0.000000%) | 0 (0.000000%) | 255.000000 | 0.000000 |
| 18 | | | dataIn#2 | wl:1, iwl:1 | 0 (0.000000%) | 0 (0.000000%) | 1.000000 | 0.000000 |

CAUTION In the case of overflow or underflow events, the values of in the Max and Min column the values depend on the settings of the **Saturation** parameter of the associated part.

### Example: Moving Average Filter

This example demonstrates how to use the Fixed Point Analysis Table to configure the Wordlength parameter in fixed point implementation of a moving averaging window filter. Open Design1 in MovingAverage.wsv workspace. This design contains a floating point and a fixed point implementation of 8-tap moving average filter. The fixed point implementation has a fixed wordlength of size WL across all filtering stages. In addition, the minimal wordlength at the output of the fixed point filter is analytically estimated in the MATLAB_Script part based on Schwarz's inequality. The initial WL setting for the design is 16 and the standard deviation of IID Gaussian Noise Waveform source is 2^8 = 256. The design is shown in the figure below.

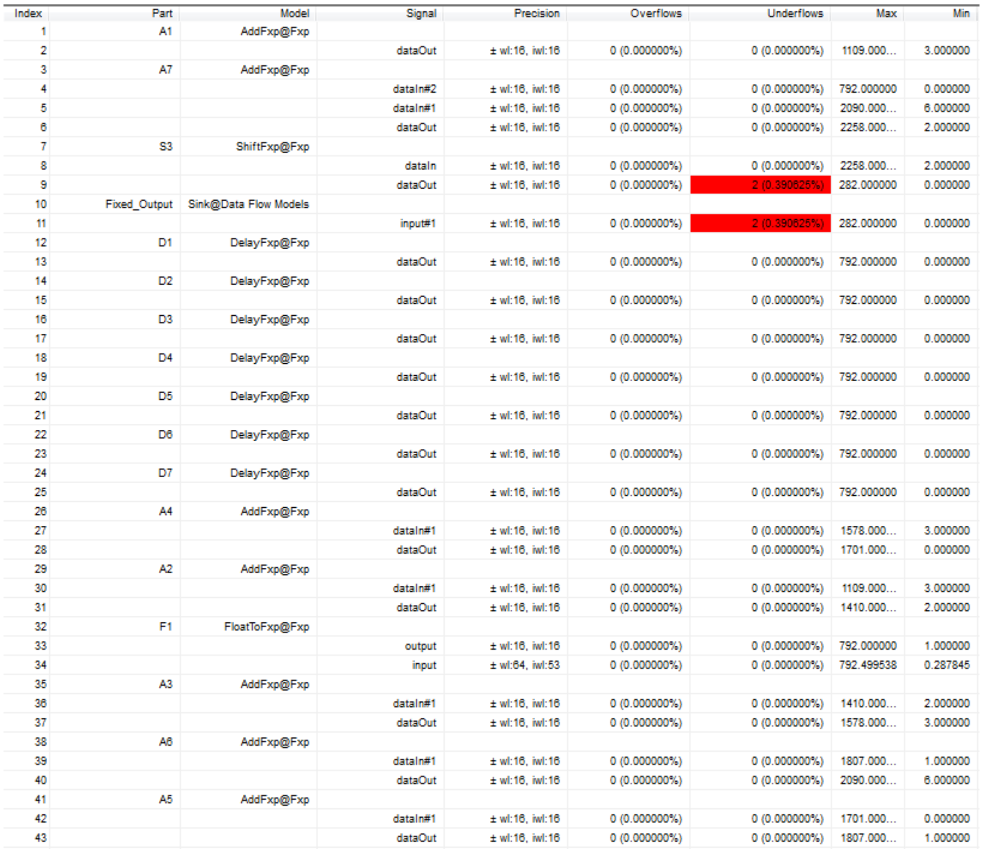

Floating-Point Moving Average Filter
Fixed-Point Moving Average Filter
WordLength Estimation
123
123
123
123

NOTE

Hint: Since the number of coefficients of the averaging filter is a power of two (8 = 2^3). The division by the number of samples ( **8** ) can be done *efficiently* using a bit shift operator.

## Wordlength Customization

1. The initial wordlength setting of the fixed point filter design is sufficient to cover the bitgrowth of the signal along the fixed point implementation of the averaging window. The Fixed Point Analysis Table allows the user to examine this bit growth and adjust the wordlength setting at each stage of the fixed point design. To examine that, follow the step below
2. Right click on Design1 Analysis in the workspace tree and select Run (Calculate Now).
3. Double-click on Design1_Data_FixedPointAnalysis. Note the bit growth along the outputs of the consecutive adders. The maximum value at the output A1 is 1109 while the maximum value at the output of A7 is 2285.



| Index | Part | Model | Signal | Precision | Overflows | Underflows | Max | Min |
|---|---|---|---|---|---|---|---|---|
| 1 | A1 | AddFxp@Fxp | | | | | | |
| 2 | | | dataOut | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 1109.000... | 3.000000 |
| 3 | A7 | AddFxp@Fxp | | | | | | |
| 4 | | | dataIn#2 | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 792.000000 | 0.000000 |
| 5 | | | dataIn#1 | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 2090.000... | 6.000000 |
| 6 | | | dataOut | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 2258.000... | 2.000000 |
| 7 | S3 | ShiftFxp@Fxp | | | | | | |
| 8 | | | dataIn | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 2258.000... | 2.000000 |
| 9 | | | dataOut | ± wl:16, iwl:16 | 0 (0.000000%) | 2 (0.390625%) | 282.000000 | 0.000000 |
| 10 | Fixed_Output | Sink@Data Flow Models | | | | | | |
| 11 | | | input#1 | ± wl:16, iwl:16 | 0 (0.000000%) | 2 (0.390625%) | 282.000000 | 0.000000 |
| 12 | D1 | DelayFxp@Fxp | | | | | | |
| 13 | | | dataOut | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 792.000000 | 0.000000 |
| 14 | D2 | DelayFxp@Fxp | | | | | | |
| 15 | | | dataOut | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 792.000000 | 0.000000 |
| 16 | D3 | DelayFxp@Fxp | | | | | | |
| 17 | | | dataOut | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 792.000000 | 0.000000 |
| 18 | D4 | DelayFxp@Fxp | | | | | | |
| 19 | | | dataOut | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 792.000000 | 0.000000 |
| 20 | D5 | DelayFxp@Fxp | | | | | | |
| 21 | | | dataOut | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 792.000000 | 0.000000 |
| 22 | D6 | DelayFxp@Fxp | | | | | | |
| 23 | | | dataOut | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 792.000000 | 0.000000 |
| 24 | D7 | DelayFxp@Fxp | | | | | | |
| 25 | | | dataOut | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 792.000000 | 0.000000 |
| 26 | A4 | AddFxp@Fxp | | | | | | |
| 27 | | | dataIn#1 | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 1578.000... | 3.000000 |
| 28 | | | dataOut | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 1701.000... | 0.000000 |
| 29 | A2 | AddFxp@Fxp | | | | | | |
| 30 | | | dataIn#1 | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 1109.000... | 3.000000 |
| 31 | | | dataOut | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 1410.000... | 2.000000 |
| 32 | F1 | FloatToFxp@Fxp | | | | | | |
| 33 | | | output | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 792.000000 | 1.000000 |
| 34 | | | input | ± wl:64, iwl:53 | 0 (0.000000%) | 0 (0.000000%) | 792.499538 | 0.287845 |
| 35 | A3 | AddFxp@Fxp | | | | | | |
| 36 | | | dataIn#1 | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 1410.000... | 2.000000 |
| 37 | | | dataOut | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 1578.000... | 3.000000 |
| 38 | A6 | AddFxp@Fxp | | | | | | |
| 39 | | | dataIn#1 | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 1807.000... | 1.000000 |
| 40 | | | dataOut | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 2090.000... | 6.000000 |
| 41 | A5 | AddFxp@Fxp | | | | | | |
| 42 | | | dataIn#1 | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 1701.000... | 0.000000 |
| 43 | | | dataOut | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 1807.000... | 1.000000 |

4. Note the underflow events occurred twice during the simulation at the output of the ShiftFxp part. Underflow event occurs when the fixed point representation of the resulting number does not have enough precision because the ideal value is very small. As a result, the number becomes zero. This is common in fixed point math when dividing a small number by a big number.

5. To verify that, let us analyze the output of the floating point filter and find how many positive values are below 1. Double-click on Design1_Data and right click on the Variable column, select Add New Variable...

6. Set the Name to *NumOfUnderflows*, and set Formula to the following equation

   ```
   length( find ( (Float_Output<1)&(Float_Output>0) ) )
   ```

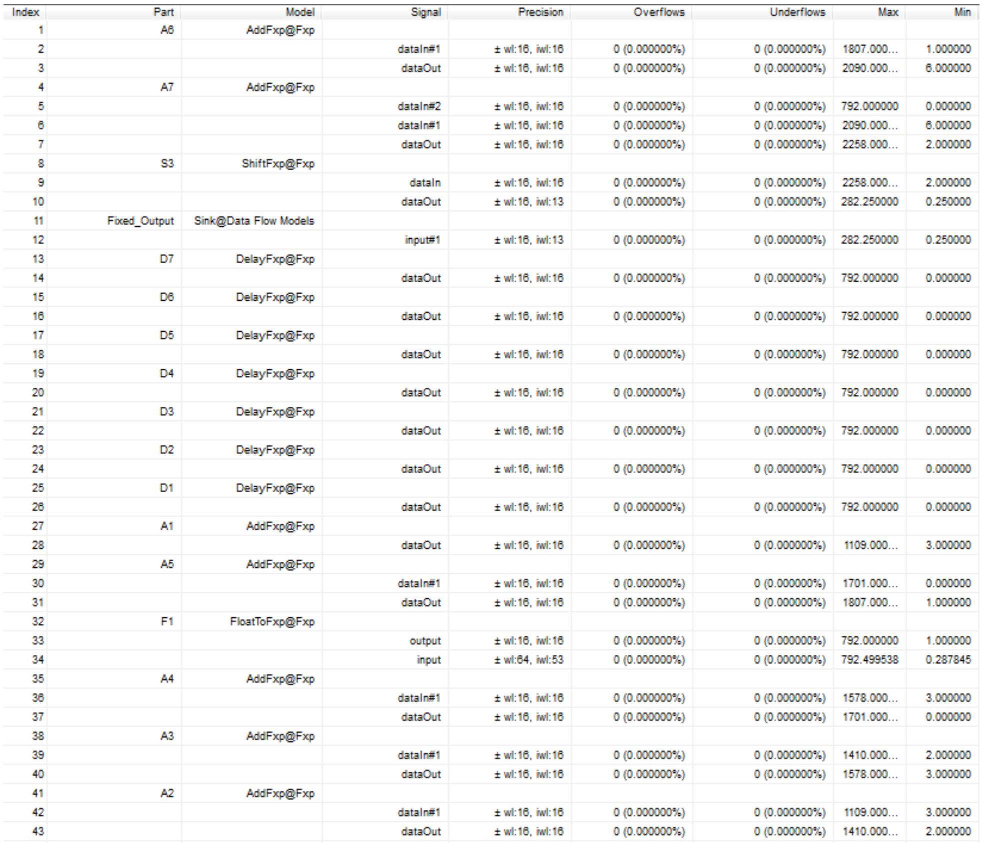
   This formula finds the number of values in Float_Output that are between 0 and 1.

7. Click OK. Note that the calculated value is 2.

8. To resolve this underflow, set the IntegerWordLength at the output of the shift operation to WL-3.

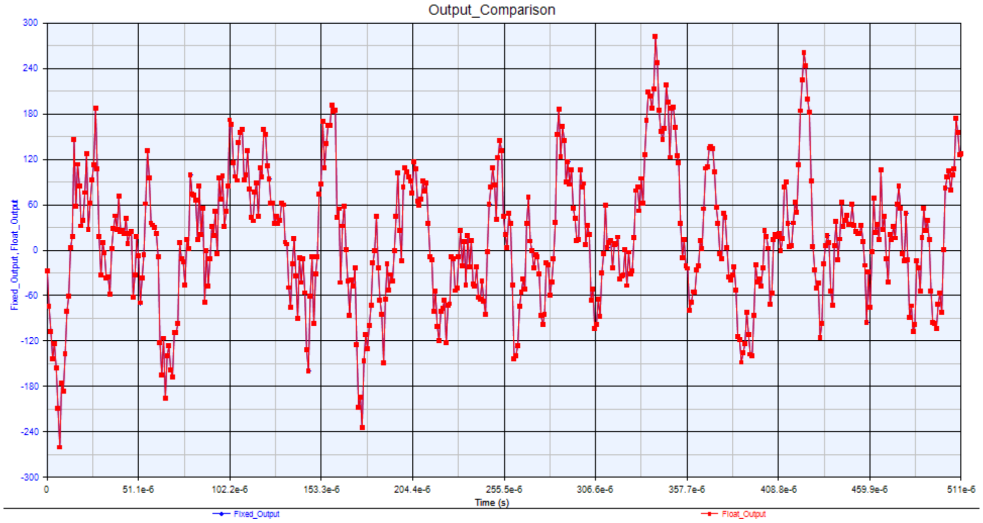
9. Right click on Design1 Analysis in the workspace tree and select Run (Calculate Now) again.

10. Double-click on Design1_Data_FixedPointAnalysis and note that the underflow events are resolved.



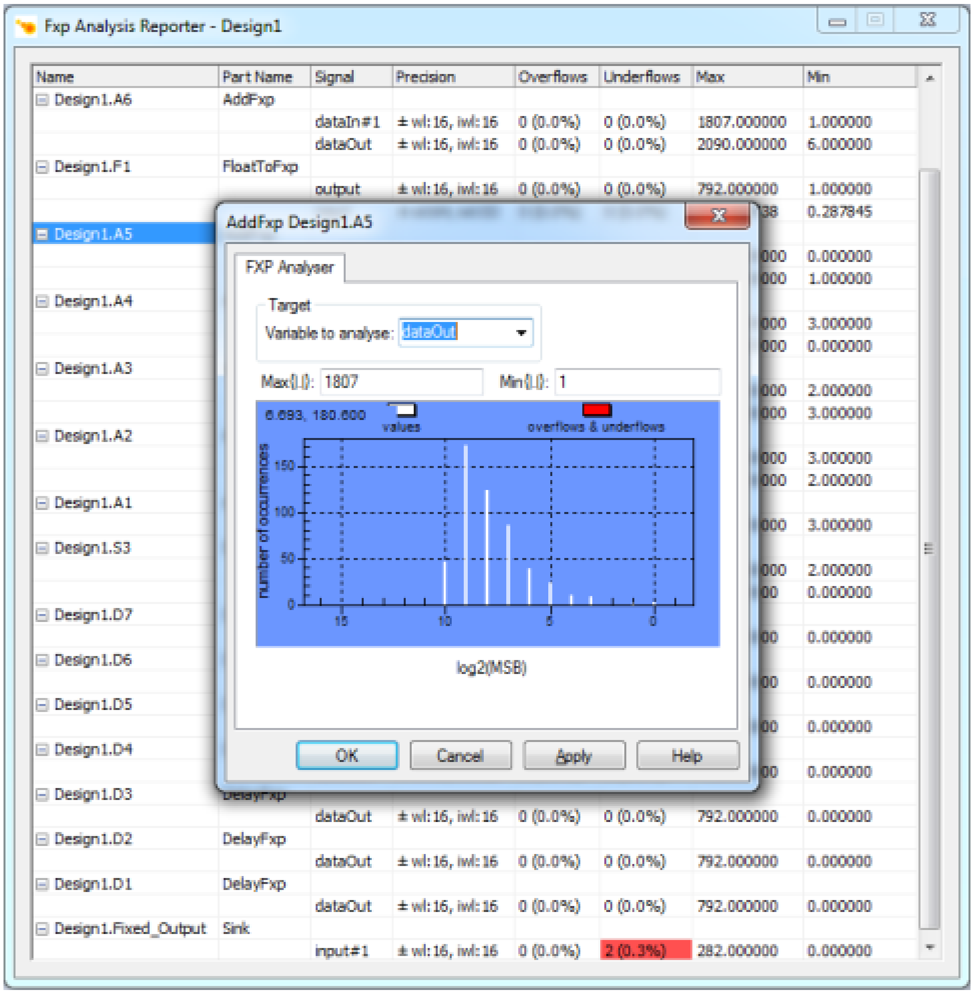
| Index | Part | Model | Signal | Precision | Overflows | Underflows | Max | Min |
|---|---|---|---|---|---|---|---|---|
| 1 | A6 | AddFxp@Fxp | | | | | | |
| 2 | | | dataIn#1 | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 1807.000... | 1.000000 |
| 3 | | | dataOut | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 2090.000... | 6.000000 |
| 4 | A7 | AddFxp@Fxp | | | | | | |
| 5 | | | dataIn#2 | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 792.000000 | 0.000000 |
| 6 | | | dataIn#1 | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 2090.000... | 6.000000 |
| 7 | | | dataOut | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 2258.000... | 2.000000 |
| 8 | S3 | ShiftFxp@Fxp | | | | | | |
| 9 | | | dataIn | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 2258.000... | 2.000000 |
| 10 | | | dataOut | ± wl:16, iwl:13 | 0 (0.000000%) | 0 (0.000000%) | 282.250000 | 0.250000 |
| 11 | Fixed_Output | Sink@Data Flow Models | | | | | | |
| 12 | | | input#1 | ± wl:16, iwl:13 | 0 (0.000000%) | 0 (0.000000%) | 282.250000 | 0.250000 |
| 13 | D7 | DelayFxp@Fxp | | | | | | |
| 14 | | | dataOut | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 792.000000 | 0.000000 |
| 15 | D6 | DelayFxp@Fxp | | | | | | |
| 16 | | | dataOut | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 792.000000 | 0.000000 |
| 17 | D5 | DelayFxp@Fxp | | | | | | |
| 18 | | | dataOut | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 792.000000 | 0.000000 |
| 19 | D4 | DelayFxp@Fxp | | | | | | |
| 20 | | | dataOut | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 792.000000 | 0.000000 |
| 21 | D3 | DelayFxp@Fxp | | | | | | |
| 22 | | | dataOut | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 792.000000 | 0.000000 |
| 23 | D2 | DelayFxp@Fxp | | | | | | |
| 24 | | | dataOut | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 792.000000 | 0.000000 |
| 25 | D1 | DelayFxp@Fxp | | | | | | |
| 26 | | | dataOut | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 792.000000 | 0.000000 |
| 27 | A1 | AddFxp@Fxp | | | | | | |
| 28 | | | dataOut | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 1109.000... | 3.000000 |
| 29 | A5 | AddFxp@Fxp | | | | | | |
| 30 | | | dataIn#1 | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 1701.000... | 0.000000 |
| 31 | | | dataOut | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 1807.000... | 1.000000 |
| 32 | F1 | FloatToFxp@Fxp | | | | | | |
| 33 | | | output | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 792.000000 | 1.000000 |
| 34 | | | input | ± wl:64, iwl:53 | 0 (0.000000%) | 0 (0.000000%) | 792.499538 | 0.287845 |
| 35 | A4 | AddFxp@Fxp | | | | | | |
| 36 | | | dataIn#1 | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 1578.000... | 3.000000 |
| 37 | | | dataOut | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 1701.000... | 0.000000 |
| 38 | A3 | AddFxp@Fxp | | | | | | |
| 39 | | | dataIn#1 | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 1410.000... | 2.000000 |
| 40 | | | dataOut | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 1578.000... | 3.000000 |
| 41 | A2 | AddFxp@Fxp | | | | | | |
| 42 | | | dataIn#1 | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 1109.000... | 3.000000 |
| 43 | | | dataOut | ± wl:16, iwl:16 | 0 (0.000000%) | 0 (0.000000%) | 1410.000... | 2.000000 |

11. Double-click on Output_Comparison graph. Note the perfect match between the output of the floating and fixed point implementation of the averaging filter.



12. At this point, the user can customize the wordlength and integer wordlength settings of the fixed point design based on the recorded Max and Min values at the input and output of each part in the fixed point design. In addition to the Max and Min values, users can examine the distribution of the values at

the input and output ports of each part using the Fxp Analysis Reporter (which is the window that is invoked at the end of the analysis). The figure below shows the histogram of the values at the output of AdderFxp A5. To obtain this histogram, double click on the part name AdderFxp A5 on the Fxp Analysis Reporter window, then set Variable to analyse to dataOut .



Fxp Analysis Reporter - Design1

| Name | Part Name | Signal | Precision | Overflows | Underflows | Max | Min |
|---|---|---|---|---|---|---|---|
| ⊟ Design1.A6 | AddFxp | | | | | | |
| | | dataIn#1 | ± wl:16, iwl:16 | 0 (0.0%) | 0 (0.0%) | 1807.000000 | 1.000000 |
| | | dataOut | ± wl:16, iwl:16 | 0 (0.0%) | 0 (0.0%) | 2090.000000 | 6.000000 |
| ⊟ Design1.F1 | FloatToFxp | | | | | | |
| | | output | ± wl:16, iwl:16 | 0 (0.0%) | 0 (0.0%) | 792.000000 | 1.000000 |
| | | | | | | 38 | 0.287845 |
| ⊟ Design1.A5 | | | | | | | |
| | | | | | | 000 | 0.000000 |
| | | | | | | 000 | 1.000000 |
| ⊟ Design1.A4 | | | | | | | |
| | | | | | | 000 | 3.000000 |
| | | | | | | 000 | 0.000000 |
| ⊟ Design1.A3 | | | | | | | |
| | | | | | | 000 | 2.000000 |
| | | | | | | 000 | 3.000000 |
| ⊟ Design1.A2 | | | | | | | |
| | | | | | | 000 | 3.000000 |
| | | | | | | 000 | 2.000000 |
| ⊟ Design1.A1 | | | | | | | |
| | | | | | | 000 | 3.000000 |
| ⊟ Design1.S3 | | | | | | | |
| | | | | | | 000 | 2.000000 |
| | | | | | | 00 | 0.000000 |
| ⊟ Design1.D7 | | | | | | | |
| | | | | | | 00 | 0.000000 |
| ⊟ Design1.D6 | | | | | | | |
| | | | | | | 00 | 0.000000 |
| ⊟ Design1.D5 | | | | | | | |
| | | | | | | 00 | 0.000000 |
| ⊟ Design1.D4 | | | | | | | |
| | | | | | | 00 | 0.000000 |
| ⊟ Design1.D3 | DelayFxp | | | | | | |
| | | dataOut | ± wl:16, iwl:16 | 0 (0.0%) | 0 (0.0%) | 792.000000 | 0.000000 |
| ⊟ Design1.D2 | DelayFxp | | | | | | |
| | | dataOut | ± wl:16, iwl:16 | 0 (0.0%) | 0 (0.0%) | 792.000000 | 0.000000 |
| ⊟ Design1.D1 | DelayFxp | | | | | | |
| | | dataOut | ± wl:16, iwl:16 | 0 (0.0%) | 0 (0.0%) | 792.000000 | 0.000000 |
| ⊟ Design1.Fixed_Output | Sink | | | | | | |
| | | input#1 | ± wl:16, iwl:16 | 0 (0.0%) | 2 (0.3%) | 282.000000 | 0.000000 |

AddFxp Design1.A5

FXP Analyser

Target

Variable to analyse: dataOut

Max(|.|): 1807

Min(|.|): 1

## Setting Wordlength Based on Analytical Estimation

In Linear Time Invariant (LTI) systems, Schwarz's inequality is used for estimating the bit growth:

$$|y[n]| \le \sqrt{\sum_{-\infty}^{\infty} h^2[n] \sum_{-\infty}^{\infty} x^2[n]}$$

where *h[n]* and *x[n]* are the impulse response and input to the LTI system respectively. A variation of that estimation is implemented in the MATLAB_Script part as shown in the equation below

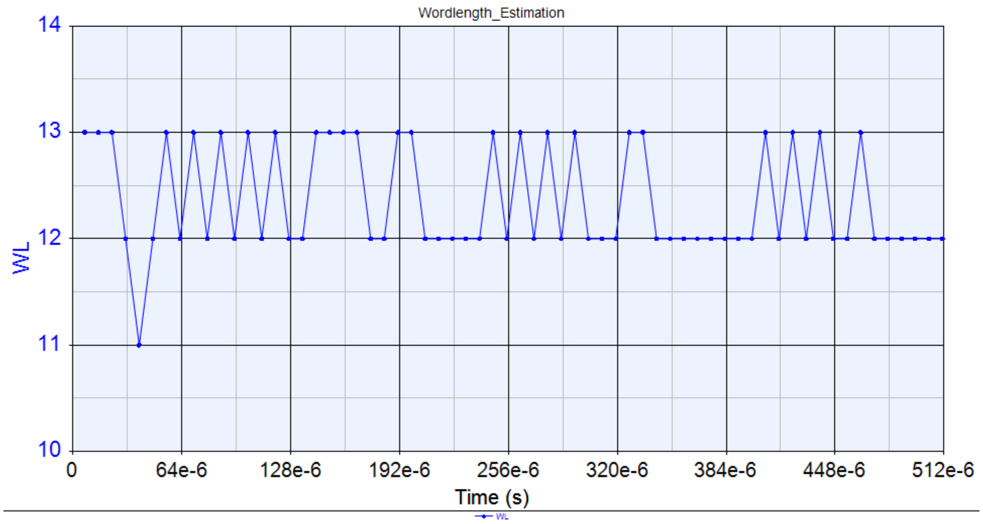
$$|y[n]| = \sqrt{\sum_{-7}^{0} h^2[n] \sum_{-7}^{0} x^2[n]} = \sqrt{8 \times \sum_{-7}^{0} x^2[n]}$$

The corresponding MATLAB Script implementation code is:

```
Sum_Of_x_2 = sum(input.^2);
Sum_Of_h_2 = 8;
Range = sqrt(Sum_Of_x_2 * Sum_Of_h_2);
output=1+ceil(log2( Range ) );
```

Note that the output of the MATLAB_Script part is the estimated wordlength. The addition of 1 at the end is to cover the negative values as well. Follow the steps below to configure the wordlength based on the analysis implemented in this MATLAB_Script part.

13. Double-click on the Wordlength_Estimation graph. This graph shows the output of the MATLAB_Script part which is the analytical estimation of the required wordlength at the output of the fixed point filter. See the figure below:



14. As can be inferred from that figure, WL=13 is the estimated value. Double-click on Equation 1on the workspace tree and change the equation in Line 1 to

```
WL = 13;
```

15. Right click on Design1 Analysis in the workspace tree and select Run (Calculate Now).

16. Verify that there is no overflow or underflow events in Design1_Data_FixedPointAnalysis table.

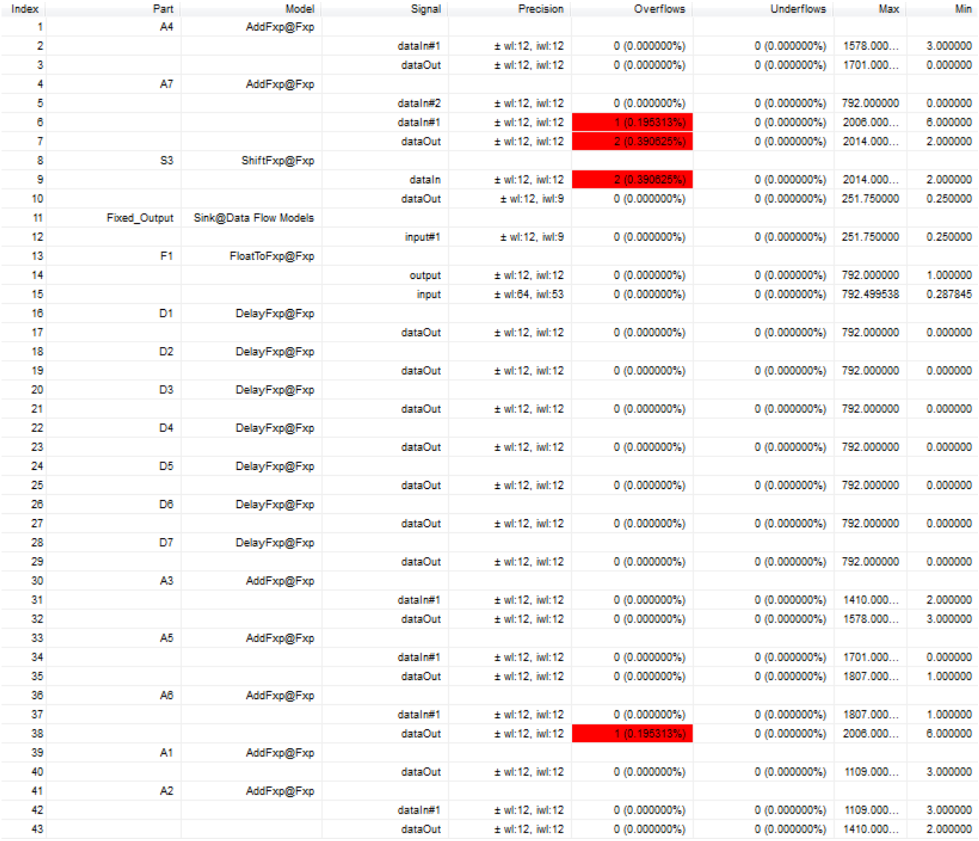
    Overflow Detection

    Fixed Point Analysis table can be used to detect the events of overflow and underflow. The section Wordlength Customization presented previously showed an example of detecting underflow events. An overflow occurs when the data value exceeds the maximum range that can be represented with its fixed point representation. The following steps demonstrate how overflow events are also reported in the Fixed Point Analysis table.

1. Double-click on Equation 1on the workspace tree and change the equation in Line 1 to

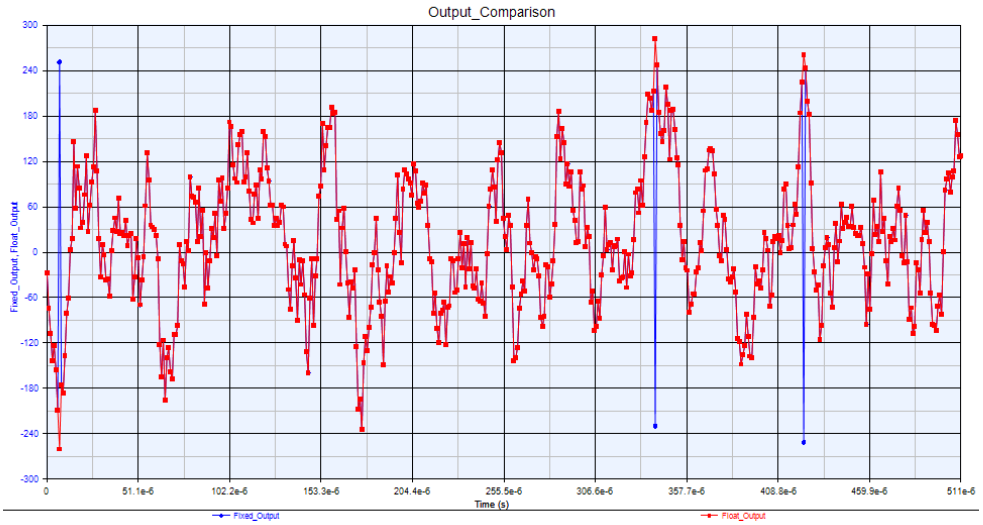
```
WL = 12;
```

2. Right click on Design1 Analysis in the workspace tree and select Run (Calculate Now).
3. Double-click on Design1_Data_FixedPointAnalysis and note that the overflow events that occurred at the output of the sixth and seventh adder A6 and A7 as well as ShiftFxp S1.



| Index | Part | Model | Signal | Precision | Overflows | Underflows | Max | Min |
|---|---|---|---|---|---|---|---|---|
| 1 | A4 | AddFxp@Fxp | | | | | | |
| 2 | | | dataIn#1 | ± wl:12, iwl:12 | 0 (0.000000%) | 0 (0.000000%) | 1578.000... | 3.000000 |
| 3 | | | dataOut | ± wl:12, iwl:12 | 0 (0.000000%) | 0 (0.000000%) | 1701.000... | 0.000000 |
| 4 | A7 | AddFxp@Fxp | | | | | | |
| 5 | | | dataIn#2 | ± wl:12, iwl:12 | 0 (0.000000%) | 0 (0.000000%) | 792.000000 | 0.000000 |
| 6 | | | dataIn#1 | ± wl:12, iwl:12 | 1 (0.195313%) | 0 (0.000000%) | 2006.000... | 6.000000 |
| 7 | | | dataOut | ± wl:12, iwl:12 | 2 (0.390625%) | 0 (0.000000%) | 2014.000... | 2.000000 |
| 8 | S3 | ShiftFxp@Fxp | | | | | | |
| 9 | | | dataIn | ± wl:12, iwl:12 | 2 (0.390625%) | 0 (0.000000%) | 2014.000... | 2.000000 |
| 10 | | | dataOut | ± wl:12, iwl:9 | 0 (0.000000%) | 0 (0.000000%) | 251.750000 | 0.250000 |
| 11 | Fixed_Output | Sink@Data Flow Models | | | | | | |
| 12 | | | input#1 | ± wl:12, iwl:9 | 0 (0.000000%) | 0 (0.000000%) | 251.750000 | 0.250000 |
| 13 | F1 | FloatToFxp@Fxp | | | | | | |
| 14 | | | output | ± wl:12, iwl:12 | 0 (0.000000%) | 0 (0.000000%) | 792.000000 | 1.000000 |
| 15 | | | input | ± wl:64, iwl:53 | 0 (0.000000%) | 0 (0.000000%) | 792.499538 | 0.287845 |
| 16 | D1 | DelayFxp@Fxp | | | | | | |
| 17 | | | dataOut | ± wl:12, iwl:12 | 0 (0.000000%) | 0 (0.000000%) | 792.000000 | 0.000000 |
| 18 | D2 | DelayFxp@Fxp | | | | | | |
| 19 | | | dataOut | ± wl:12, iwl:12 | 0 (0.000000%) | 0 (0.000000%) | 792.000000 | 0.000000 |
| 20 | D3 | DelayFxp@Fxp | | | | | | |
| 21 | | | dataOut | ± wl:12, iwl:12 | 0 (0.000000%) | 0 (0.000000%) | 792.000000 | 0.000000 |
| 22 | D4 | DelayFxp@Fxp | | | | | | |
| 23 | | | dataOut | ± wl:12, iwl:12 | 0 (0.000000%) | 0 (0.000000%) | 792.000000 | 0.000000 |
| 24 | D5 | DelayFxp@Fxp | | | | | | |
| 25 | | | dataOut | ± wl:12, iwl:12 | 0 (0.000000%) | 0 (0.000000%) | 792.000000 | 0.000000 |
| 26 | D6 | DelayFxp@Fxp | | | | | | |
| 27 | | | dataOut | ± wl:12, iwl:12 | 0 (0.000000%) | 0 (0.000000%) | 792.000000 | 0.000000 |
| 28 | D7 | DelayFxp@Fxp | | | | | | |
| 29 | | | dataOut | ± wl:12, iwl:12 | 0 (0.000000%) | 0 (0.000000%) | 792.000000 | 0.000000 |
| 30 | A3 | AddFxp@Fxp | | | | | | |
| 31 | | | dataIn#1 | ± wl:12, iwl:12 | 0 (0.000000%) | 0 (0.000000%) | 1410.000... | 2.000000 |
| 32 | | | dataOut | ± wl:12, iwl:12 | 0 (0.000000%) | 0 (0.000000%) | 1578.000... | 3.000000 |
| 33 | A5 | AddFxp@Fxp | | | | | | |
| 34 | | | dataIn#1 | ± wl:12, iwl:12 | 0 (0.000000%) | 0 (0.000000%) | 1701.000... | 0.000000 |
| 35 | | | dataOut | ± wl:12, iwl:12 | 0 (0.000000%) | 0 (0.000000%) | 1807.000... | 1.000000 |
| 36 | A6 | AddFxp@Fxp | | | | | | |
| 37 | | | dataIn#1 | ± wl:12, iwl:12 | 0 (0.000000%) | 0 (0.000000%) | 1807.000... | 1.000000 |
| 38 | | | dataOut | ± wl:12, iwl:12 | 1 (0.195313%) | 0 (0.000000%) | 2006.000... | 6.000000 |
| 39 | A1 | AddFxp@Fxp | | | | | | |
| 40 | | | dataOut | ± wl:12, iwl:12 | 0 (0.000000%) | 0 (0.000000%) | 1109.000... | 3.000000 |
| 41 | A2 | AddFxp@Fxp | | | | | | |
| 42 | | | dataIn#1 | ± wl:12, iwl:12 | 0 (0.000000%) | 0 (0.000000%) | 1109.000... | 3.000000 |
| 43 | | | dataOut | ± wl:12, iwl:12 | 0 (0.000000%) | 0 (0.000000%) | 1410.000... | 2.000000 |

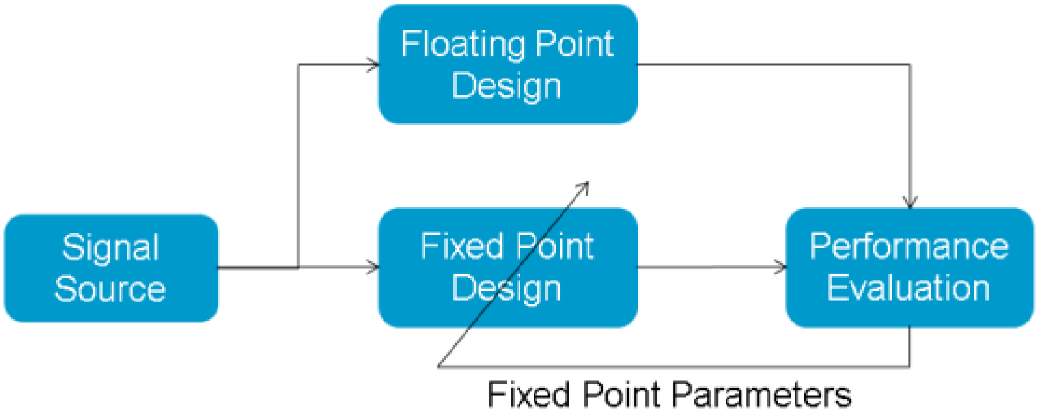
4. Double-click on Output_Comparison graph. Note the impact of overflow events on the output of fixed point filter as shown in the figure below:



## Sweep Analysis

Simulation-based fixed point parameter optimization is usually conducted by evaluating the performance of the digital design at different fixed point parameters. The performance can be measured in comparison to a reference design such as floating point design and then the fixed point analysis can be adjusted accordingly.

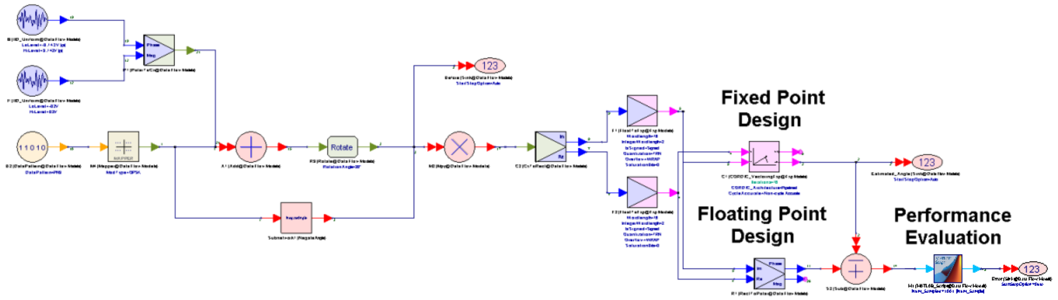
In sweep analysis, users can select several design parameters for tuning, and conduct several simulations at different values of these tunable parameters and observe the corresponding results of these simulations.



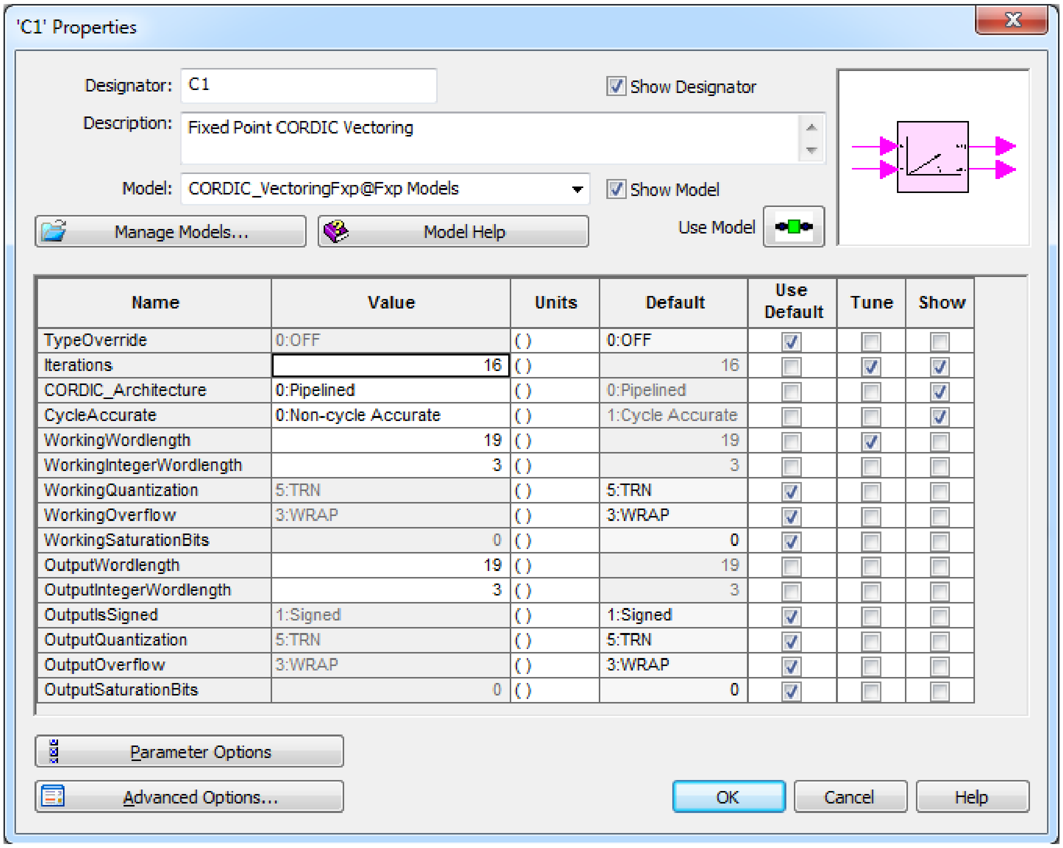
## Example: CORDIC Parameter Optimization

This example demonstrates a heuristic approach to determine the number of iterations and wordlength in CORDIC_VectoringFxp part to fulfill certain performance criteria, e.g. precision. The CORDIC_VectoringFxp implements a vector translation (rectangular to polar conversion) using the COordinate Rotational DIgital Computer (CORDIC) algorithm.

The design below uses CORDIC part to estimate the phase shift of QPSK symbols. The phase shift is simulated by complex rotation of the symbols produced by the Mapper. The output of the CORDIC vectoring part ( angle measured in radians) is compared against the output of a floating point implementation of the CORDIC part functionality (RectToPolar). The performance is measured by the average absolute difference between the angles measured using CORDIC function and the one measured by the RectToPolar part. This difference should be less than 0.01 radian.



The CORDIC part is initially set at its default values as shown in the figure below. The objective of this example is to optimize the WorkingWordlength and the number of Iterations parameters. Hence, they are set for tuning as can be seen in the Tune column next to these parameters.



'C1' Properties

Designator: C1 ☑ Show Designator

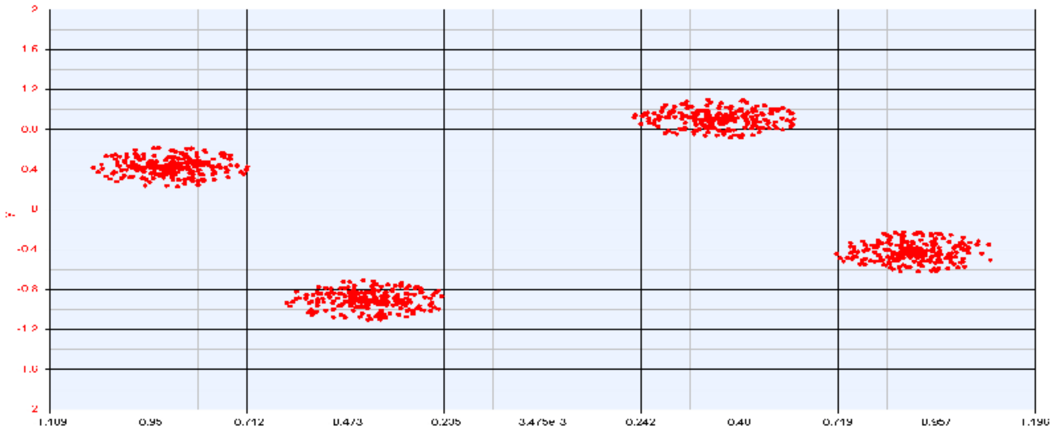
Description: Fixed Point CORDIC Vectoring

Model: CORDIC_VectoringFxp@Fxp Models ☑ Show Model

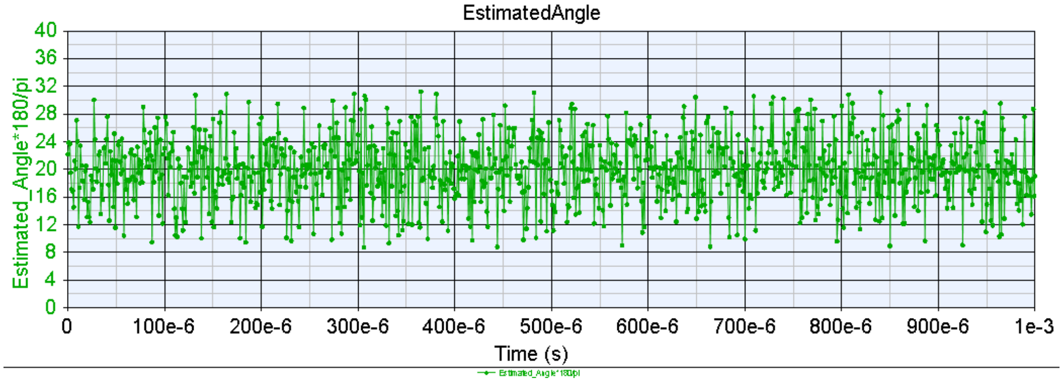
| Name | Value | Units | Default | Use Default | Tune | Show |
|---|---|---|---|---|---|---|
| TypeOverride | 0:OFF | ( ) | 0:OFF | ☑ | □ | □ |
| Iterations | 16 | ( ) | 16 | □ | ☑ | ☑ |
| CORDIC_Architecture | 0:Pipelined | ( ) | 0:Pipelined | □ | □ | ☑ |
| CycleAccurate | 0:Non-cycle Accurate | ( ) | 1:Cycle Accurate | □ | □ | ☑ |
| WorkingWordlength | 19 | ( ) | 19 | □ | ☑ | □ |
| WorkingIntegerWordlength | 3 | ( ) | 3 | □ | □ | □ |
| WorkingQuantization | 5:TRN | ( ) | 5:TRN | ☑ | □ | □ |
| WorkingOverflow | 3:WRAP | ( ) | 3:WRAP | ☑ | □ | □ |
| WorkingSaturationBits | 0 | ( ) | 0 | ☑ | □ | □ |
| OutputWordlength | 19 | ( ) | 19 | □ | □ | □ |
| OutputIntegerWordlength | 3 | ( ) | 3 | □ | □ | □ |
| OutputIsSigned | 1:Signed | ( ) | 1:Signed | ☑ | □ | □ |
| OutputQuantization | 5:TRN | ( ) | 5:TRN | ☑ | □ | □ |
| OutputOverflow | 3:WRAP | ( ) | 3:WRAP | ☑ | □ | □ |
| OutputSaturationBits | 0 | ( ) | 0 | ☑ | □ | □ |

CAUTION To simplify the comparison between the floating and fixed point designs, the **CycleAccurate** parameter is set to *Non-cycle Accurate* so that the fixed point model will have zero latency and no need to use a re-timing delay to align the outputs of the floating and fixed point models.

Run Design 1 Analysis. Note that the initial default settings have good precision since the Error variable obtained at the end of analysis is around Error is around 33e-6. Check the input signal in the Constellation_Before_Correction graph shown below. This graph shows several QPSK symbols with phase shift of 20 degrees.



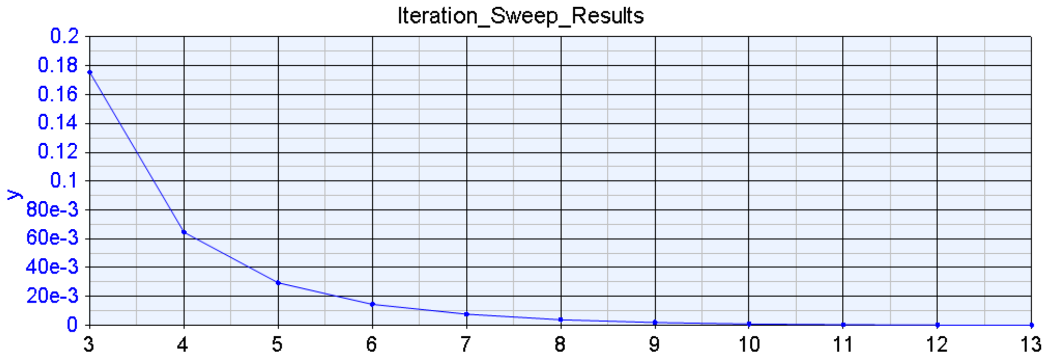
Note the EstimatedAngle graph as shown below. The estimated phase oscillates around 20 degrees.



In the next steps, we are going to find the minimal values for the WorkingWordlength and the number of Iterations parameters in CORDIC part so that the difference between the Error value should not exceed 0.01 radian.

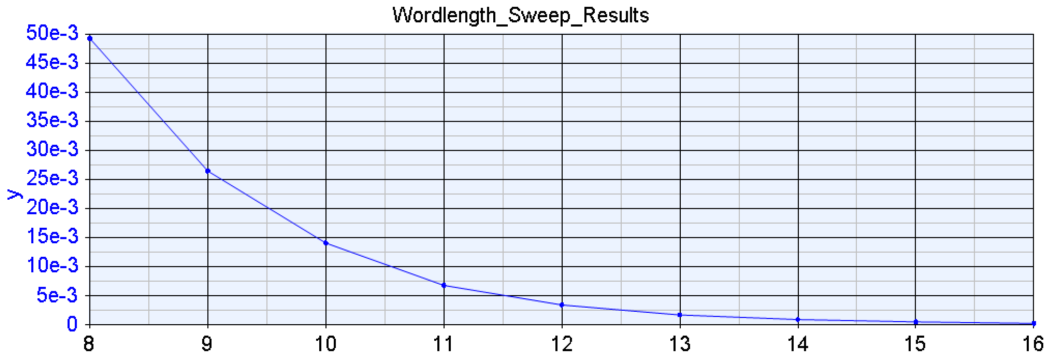
Number of Iterations optimization

Double-click on the Iteration_Sweep in the workspace tree. Note that we are going to examine the results of simulation for a number of iterations that varies from 3 to 13. The WorkingWordlength parameter will be set to its default value during this sweep. By examining the results in the Iteration_Sweep_Results graph, it seems that 7 iterations is the minimum number of iterations that results in Error value less than 0.01.



Wordlength_Sweep

Double-click on the Wordlength_Sweep in the workspace tree. This sweep analysis will generate the results of simulation for WorkingWordlength parameter that varies from 8 to 16. The number of iterations parameter will be set to its default value during this sweep. By examining the results in the Wordlength_Sweep_Results graph, it shows that WorkingWordlength of 11 bits is the minimum wordlength that results in Error value less than 0.01.



Now verify that the Error remains below 0.01 when applying the findings of these two sweeps in the CORDIC part in Design 1 :

- WorkingWordlength = 11. (Since the precision of the internal calculations is set at this wordlength, set the OutputWordlength to 9 as well).
- Iterations = 7.

CAUTION Optimizing fixed point parameters is based on the assumption that the data input values are within specific dynamic range. Otherwise, in order to support the full range of input, the full internal precision should be set to ( InputWordlength+ OutputWordlength + log2( OutputWordlength ) ) .

## HDL Co-simulation

### Prerequisite

- ModelSim SE/Questa from Mentor Graphics
- Riviera PRO from Aldec

NOTE For software version compatibility, refer to System Requirements

Associated workspaces

- *CustomHDL_workspace.wsv* in Examples\Tutorials\Hardware_Design\CoSim\Custom_HDL
- *NCO_Cosim.wsv* in Examples\Tutorials\Hardware_Design\CoSim\Generated_HDL
- *MultiRate_Filtering.wsv* in Examples\Tutorials\Hardware_Design\CoSim\MultiRate

CAUTION We recommend you to copy the complete directory of **Examples\Tutorials\Hardware_Design** and **Examples\Hardware Design\HDLCodeGeneration\NCO** to your local directory to avoid file writing permission issues on default installation directory.

## Introduction

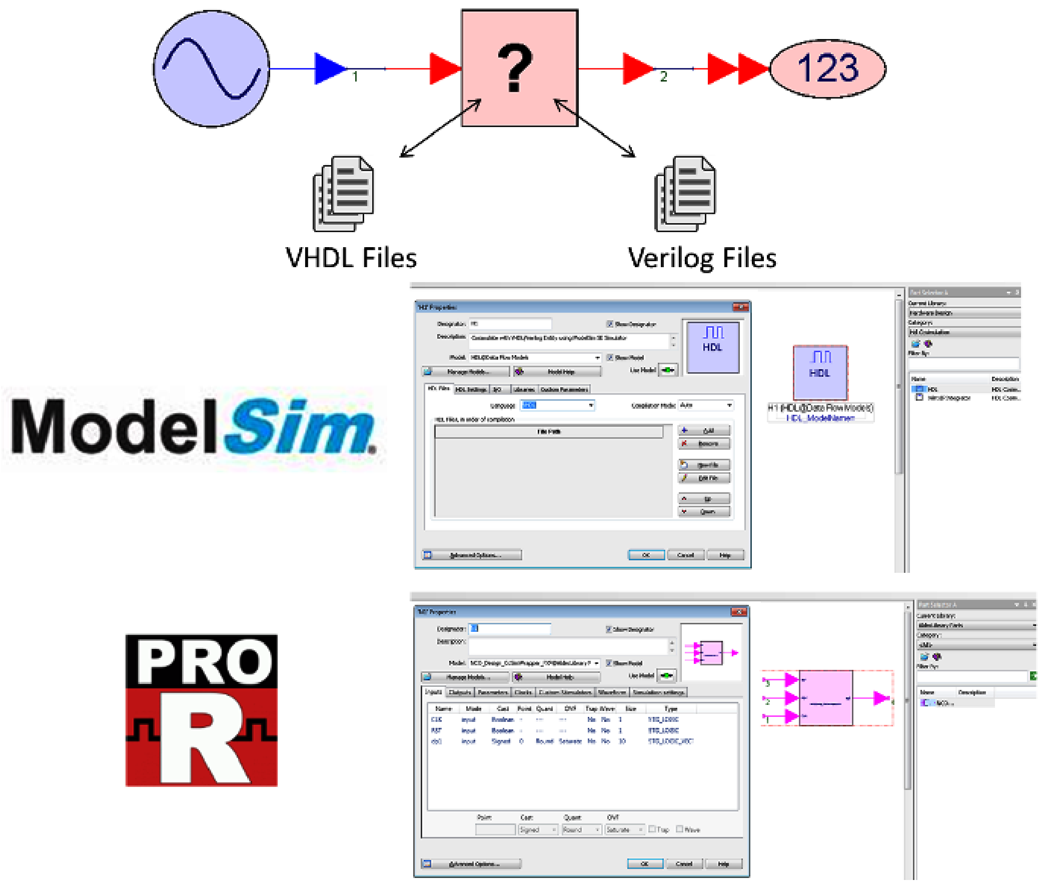
In this tutorial, we demonstrate how to co-simulate HDL designs within SystemVue environment. HDL designs are described in VHDL files (.vhd) or Verilog files (.v). These files can be obtained from:

- Generated HDL Files: These are HDL files that are generated automatically using the HDL code generation capability of SystemVue for a design developed using the Hardware Design Library.
- Custom HDL files: These are HDL files that are obtained from other sources.
- MultiRate design: This is a special case of the first case (Generated HDL Files ) in which the HDL design is multirate (the sampling rates of the input and output ports are different).

HDL cosimulation requires one of the two HDL simulators to be available:

- Modelsim SE/Questa from Mentor Graphics
- Riviera Pro from Aldec

The figure below summarizes the HDL-cosimulation flow in SystemVue



### Co-simulating existing HDL code

In this tutorial, we are going to use the following designs:

- Combinational logic design: n-bit Adder
- Sequential logic design: n-bit counter

For this tutorial, VHDL examples are used. However, the same steps can be applied for Verilog based designs.

### Simple Combinational logic design

In combinational logic designs, the output signals depend only on the current status of input signals. A number of combinational logic designs can be found under CoSim\Custom_HDL\HDL_Source\VHDL\Combinational folder. In this example, we are going to use the n-bit adder design described in Adder.vhd. The contents of this file is shown below:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
---------------------------------------------------------
entity ADDER is
generic(n: natural :=4);
```

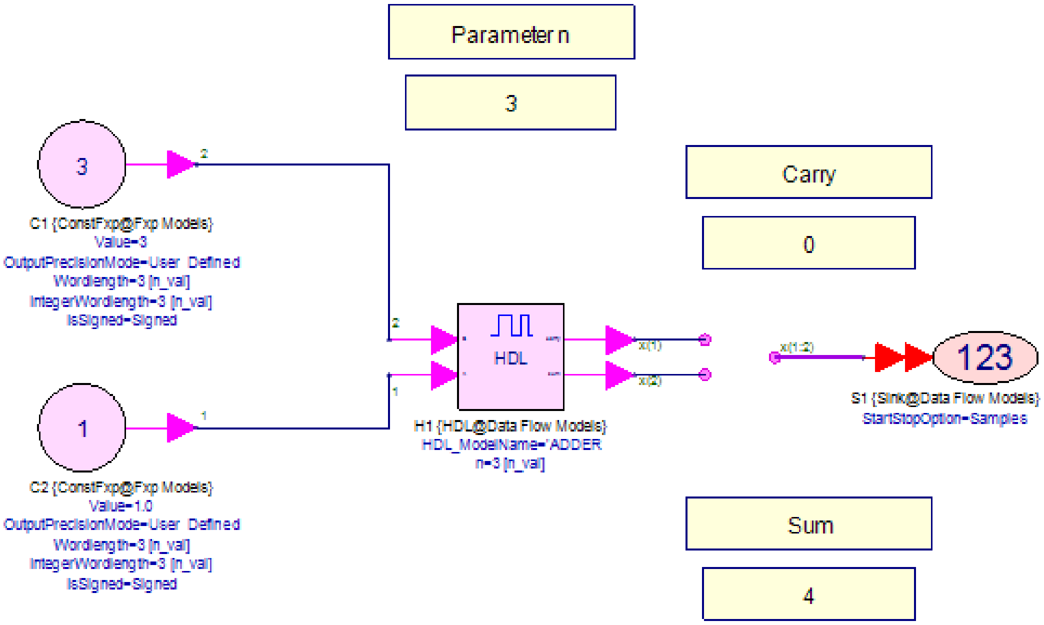
```
port(    A:    in std_logic_vector(n-1 downto 0);
    B:    in std_logic_vector(n-1 downto 0);
    carry:    out std_logic;
    sum:    out std_logic_vector(n-1 downto 0)
);
end ADDER;
--------------------------------------------------------
architecture behv of ADDER is
-- define a temparary signal to store the result
signal result: std_logic_vector(n downto 0);
begin
    -- the 3rd bit should be carry
    result <= ('0' & A)+('0' & B);
    sum <= result(n-1 downto 0);
    carry <= result(n);
end behv;
--------------------------------------------------------
```

The following subsections provide will guide you on how to cosimulate this HDL code using ModelSim/Questa and Riviera PRO inside SystemVue environment.

### Co-simulation using ModelSim and Questa

**NOTE** General flow: Co-simulating HDL files with ModelSim/Questa can be done using the HDL cosim block.

Open CustomHDL_workspace.wsv. On the workspace tree, open the schematic design in CoSim-Combinational > Comb_Design_Modelsim. The design is shown in figure below:



- Double click on the HDL cosim block to bring up the Properties window. Verify the following:

- HDL Files page: The location of the HDL files are added. In this tutorial, there is a single VHDL file, which is Adder.vhd.

  NOTE The path of the file can be defined as the absolute or relative path. Depending on how the user will be moving the workspaces and HDL files around, one choice might be better than the other. If the HDL source files will remain in a fixed location, the absolute path might be more practical. However, if the user is moving SystemVue workspace along with the HDL source while maintaining the folder and file structure, then the relative path is more practical.

- HDL settings page: The top level entry is the design is added. In this tutorial, we can infer from the Adder.vhd file that the top entry is ADDER. In addition, the Iteration Time determines the duration of each clock cycle in the HDL simulation.

  CAUTION The iteration time will be used inside the HDL simulator [ModelSim/Questa] only. SystemVue does not use the time information of the HDL simulation. The value should be an integer larger than *one*.

- I/O page: The input and output ports are added. The names in this listing should match the name of the ports in the HDL file that describes the top-level entity. Note that you only need to specify the port data type specifications (wordlength, integer wordlength, and Is signed) for the output ports. These setting are based on the following blocks that are used in the design and the desired representation format of the output data.
- Custom Parameters page: Any parameters used in the design are added here. In this tutorial, the parameter used is n and the value of that parameter is n_val which is defined by an equation.

  NOTE Tip: The reason for defining a variable **n_val** is to be able to control the value of the parameter **n** (which is the adder wordlength) in the **I/O** page and in the **Custom Parameters** page at once.

- Change the input, output and parameter values to examine the behavior of the cosimulation.
- Develop similar HDL cosim blocks for the *comb.vhd* and *multiplier.vhd* in Cosim\Custom_HDL\HDL_Source\VHDL\Combinational\VHDL folder.

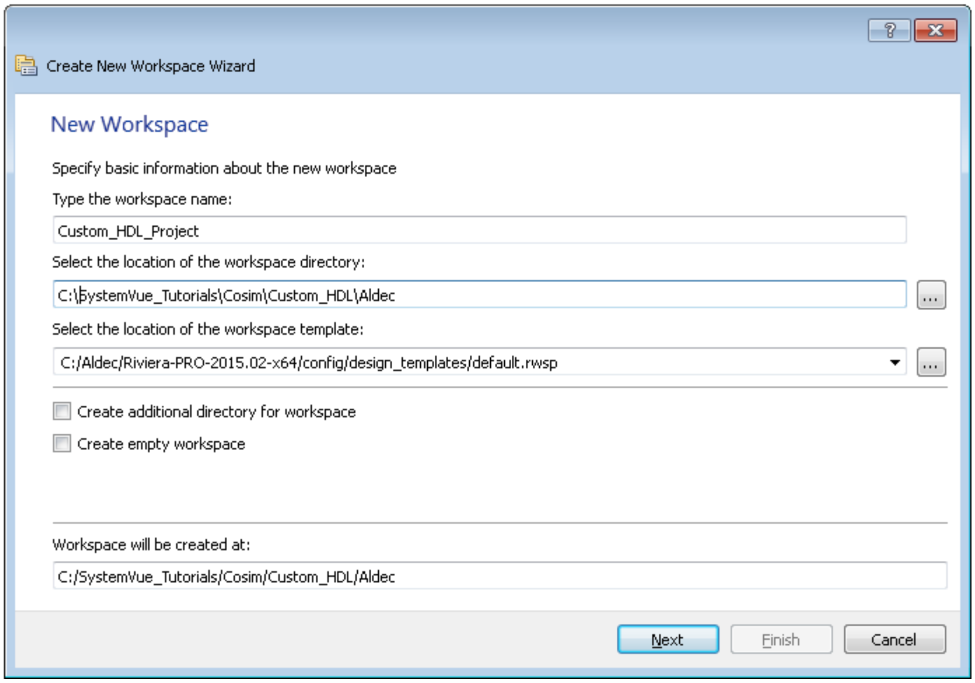
NOTE General flow: Co-simulating HDL files with Riviera Pro is done in two stages:

1. In Riviera Pro: Compile the design HDL files and generate the design XML library file for SystemVue

2. In SystemVue: Load the generated XML library and instantiate the generated Fixed point/Floating Point SystemVue parts of the HDL design.

For this example, we are going to perform the two stages above to cosimulate the Adder design using Riviera Pro:
Stage 1 - in Riviera PRO

- Start Riviera PRO
- Go to File > New > Workspace... to create a new workspace named *Custom_HDL_Project*.
- Apply the settings as shown in the figure below



Create New Workspace Wizard

New Workspace

Specify basic information about the new workspace

Type the workspace name:

Custom_HDL_Project

Select the location of the workspace directory:

C:\SystemVue_Tutorials\Cosim\Custom_HDL\Aldec

Select the location of the workspace template:

C:/Aldec/Riviera-PRO-2015.02-x64/config/design_templates/default.rwsp

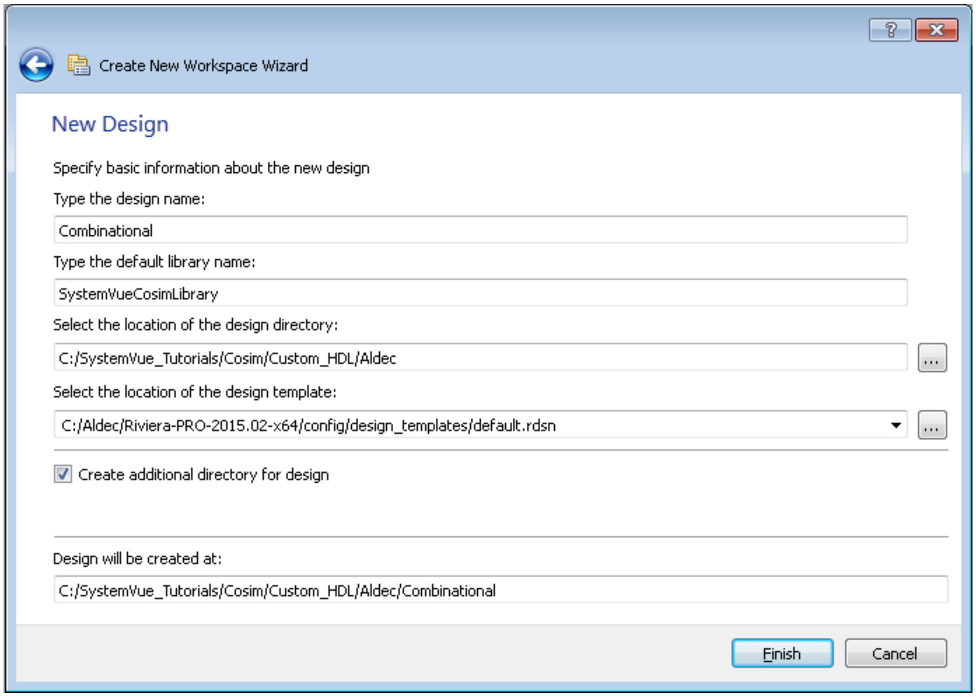
□ Create additional directory for workspace

□ Create empty workspace

Workspace will be created at:

C:/SystemVue_Tutorials/Cosim/Custom_HDL/Aldec

- Click Next
- Apply the settings as shown in the figure below



Create New Workspace Wizard

New Design

Specify basic information about the new design

Type the design name:

Combinational

Type the default library name:

SystemVueCosimLibrary

Select the location of the design directory:

C:/SystemVue_Tutorials/Cosim/Custom_HDL/Aldec

Select the location of the design template:

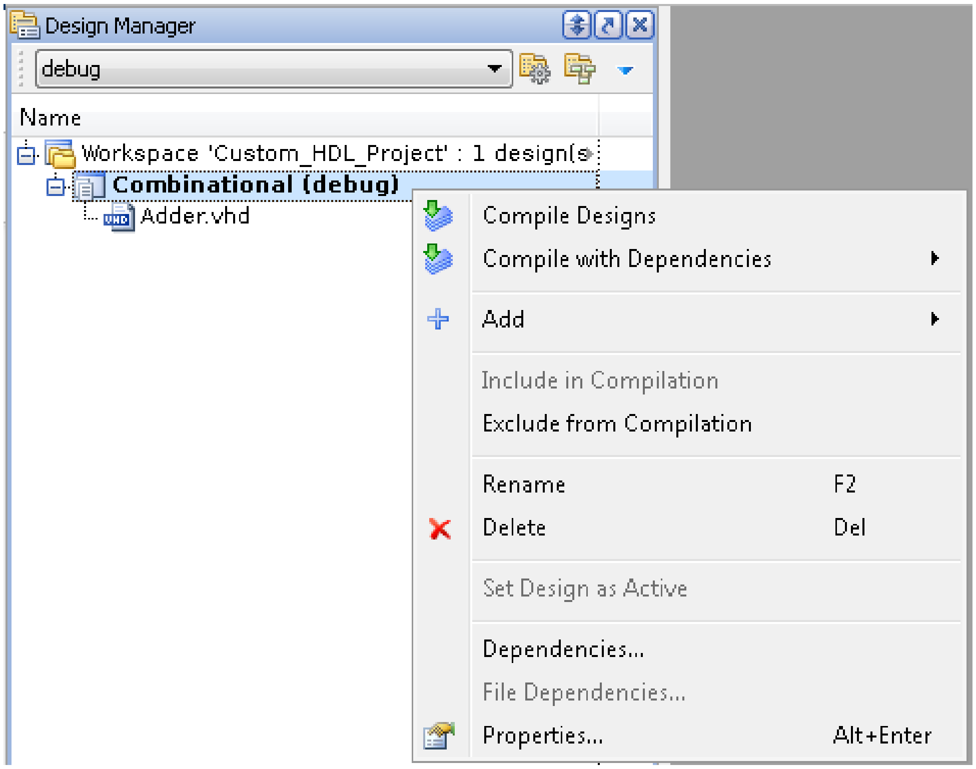
C:/Aldec/Riviera-PRO-2015.02-x64/config/design_templates/default.rdsn

☑ Create additional directory for design

Design will be created at:

C:/SystemVue_Tutorials/Cosim/Custom_HDL/Aldec/Combinational

- Click Finish
- In the Design Manager window, select Combinational design, right click and select Add -> Existing File ... to add the Adder.vhd to it from CoSim\Custom_HDL\HDL_Source\VHDL\Combinational folder.
- Right click on Combinational design and select "Compile Designs"



Design Manager

debug

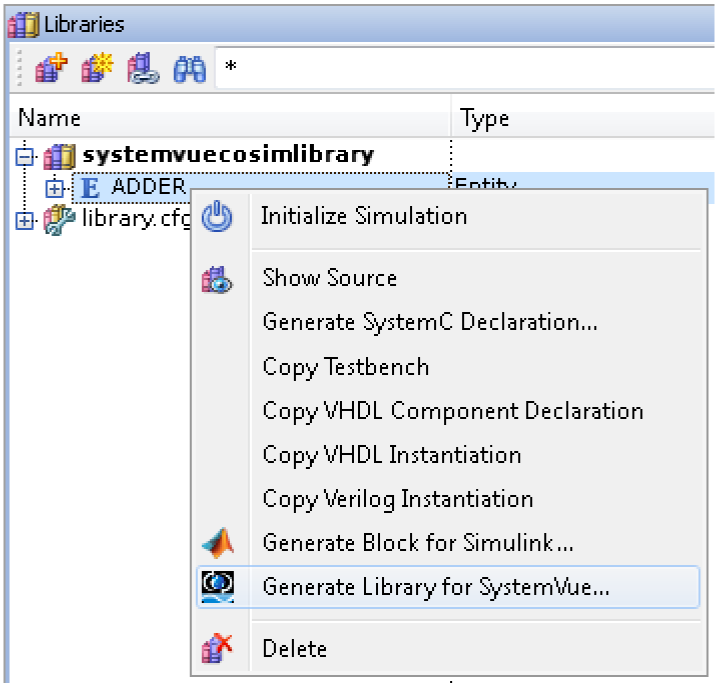
Name

Workspace 'Custom_HDL_Project' : 1 design(s

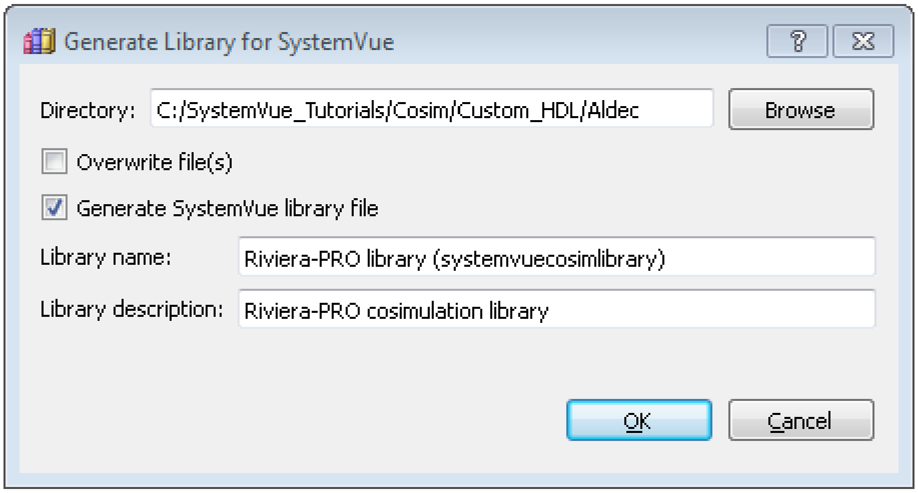
Combinational (debug)

Adder.vhd

- Go to the Library manager window, right click on the ADDER in the systemvuecosimlibrary library and select Generate Library for SystemVue


Libraries
*
Name
Type
systemvuecosimlibrary
ADDER
Entity
library.cfg
Initialize Simulation
Show Source
Generate SystemC Declaration...
Copy Testbench
Copy VHDL Component Declaration
Copy VHDL Instantiation
Copy Verilog Instantiation
Generate Block for Simulink...
Generate Library for SystemVue...
Delete

- Make sure that the Library name is *Riviera-PRO library (systemvuecosimlibrary)* as shown in the figure below and click OK.



Directory: C:/SystemVue_Tutorials/Cosim/Custom_HDL/Aldec

☐ Overwrite file(s)

☑ Generate SystemVue library file

Library name: Riviera-PRO library (systemvuecosimlibrary)

Library description: Riviera-PRO cosimulation library

Note in the console window that Riviera PRO will run the systemvuegenmod command to generate the RivieraProModelFactory.XML library that is going to be loaded in SystemVue to be used for HDL cosimulation.

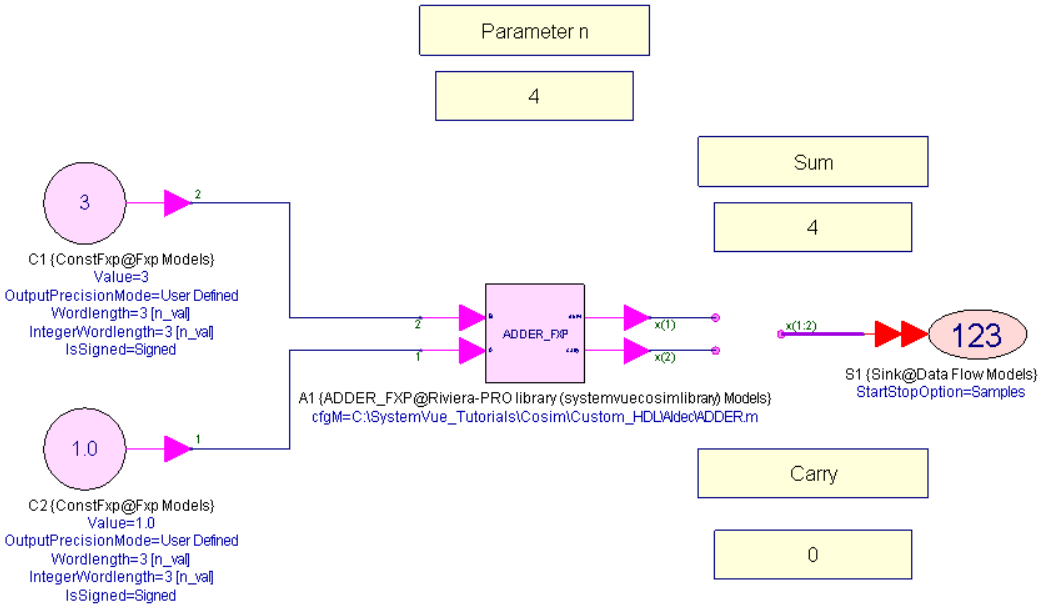
**NOTE** Type **systemvuegenmod -help** in Riviera PRO console to check the possible parameters of this command. For more information, refer to HDL Cosimulation With Riviera PRO

Stage 2 - in SystemVue

- Open CustomHDL_workspace.wsv.

- Load the generated XML library ( RivieraProModelFactory.xml ) using the Library manager (Tools > Library Manager... and then click Add From File...).
- Find the library: Riviera-PRO library in the current library list.
- On the workspace tree, open the schematic design in CoSim-Combinational > Comb_Design_RPRO.
- Double click on ADDER_FXP.
- Verify the following setting in ADDER_FXP properties window:
    - Inputs page: The input ports A and B are added. Set the Integer wordlength to be the same as the specified size in the parameter page ( which is 4 ).
    - Outputs page: Set the Integer wordlength to be the same as the specified size in the parameter page ( which is 4 ), and change the cast to Unsigned.
    - Parameters page: note that the parameter n is specified here.
    - Clocks page: In this combinational logic design, no clock signals are used. This page remains empty
    - Custom stimulators page: In this combinational logic design, no stimulators signals such as RST are used. This page remains empty
    - Waveform page: In this page, you can add the signals that you would like to be plotted in the Waveform window if you chose to invoke Riviera Pro in the Simulation Settings page. However, in this tutorial, we are going to run the cosimulation within SystemVue environment, so no signals will be added to this page.
    - Simulation Settings page: To run the cosimulation without invoking Riviera PRO, select Enable Batch Mode and Quit simulator after cosimulation options

The design is shown in figure below:


Parameter n
4
Sum
4
3
2
C1 {ConstFxp@Fxp Models}
Value=3
OutputPrecisionMode=User Defined
Wordlength=3 [n_val]
IntegerWordlength=3 [n_val]
IsSigned=Signed
2
1
ADDER_FXP
x(1)
x(2)
x(1:2)
123
S1 {Sink@Data Flow Models}
StartStopOption=Samples
A1 {ADDER_FXP@Riviera-PRO library (systemvuecosimlibrary) Models}
cfgM=C:\SystemVue_Tutorials\Cosim\Custom_HDL\Aldec\ADDER.m
1.0
1
C2 {ConstFxp@Fxp Models}
Value=1.0
OutputPrecisionMode=User Defined
Wordlength=3 [n_val]
IntegerWordlength=3 [n_val]
IsSigned=Signed
Carry
0

- Run the simulation and verify the results.

### Simple Sequential Design (Counter)

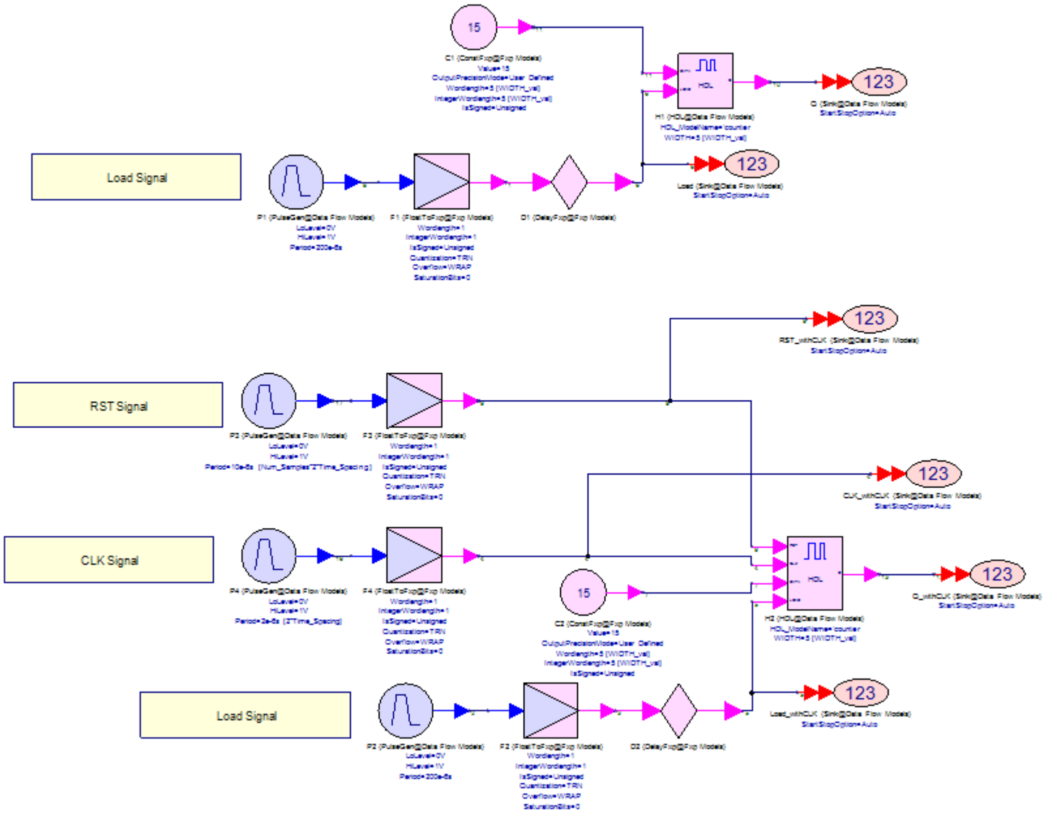
In sequential logic design, the output signals depend on the current status of input signals and the history of input signals.

```
----------------------------------------------------
-- VHDL code for n-bit counter
-- this is the behavior description of n-bit counter
----------------------------------------------------
library ieee ;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
----------------------------------------------------
entity counter is
generic(n: natural :=2);
port(    clock:    in std_logic;
    clear:    in std_logic;
    count:    in std_logic;
    Q:    out std_logic_vector(n-1 downto 0)
);
end counter;
----------------------------------------------------
architecture behv of counter is
    signal Pre_Q: std_logic_vector(n-1 downto 0);
begin
    -- behavior describe the counter
    process(clock, count, clear)
    begin
    if clear = '1' then
         Pre_Q <= Pre_Q - Pre_Q;
    elsif (clock='1' and clock'event) then
        if count = '1' then
        Pre_Q <= Pre_Q + 1;
        end if;
    end if;
    end process;
    -- concurrent assignment statement
    Q <= Pre_Q;
end behv;
-----------------------------------------------------
```

The following subsections provide will guide you on how to cosimulate this HDL code using ModelSim/Questa and Riviera PRO inside SystemVue environment. The main difference between cosimulating a sequential logic circuit such as the counter above compared to a combinational logic circuit such as the adder in the previous section is in handling the following input signals: clocks and Stimulator signals such as RST.

NOTE General flow: Co-simulating HDL files with ModelSim/Questa can be done using the HDL cosim block.

Open CustomHDL_workspace.wsv. On the workspace tree, open the schematic design in CoSim-Sequential > Seq_Design_Modelsim. The design is shown in figure below:



Note that there are two instantiations of the counter in this design. One is without Clock (CLK) and RST inputs (the simulator generates them internally)>. The other one is with CLK and RST inputs and in this case, these input signals have to be generated inside SystemVue environment.

CAUTION When using ModelSim/Questa for cosimulation, the clock signal has to be generated internally if the user would like to have a complete clock cycle for each input sample.

- Double click on the HDL cosim block of the top counter block (which has the CLK and RSTsignals generated by internally using ModelSim/Questa) to bring up the Properties window. Verify the following:
  - HDL Files page: The location of the HDL files are added which is, in this case, a single VHDL file: *counter.vhd*.

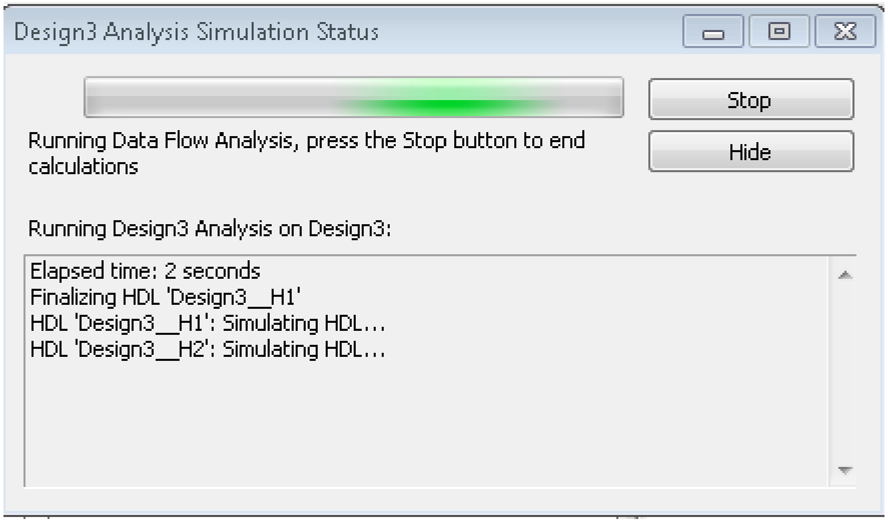
- HDL settings page: The top level entry is the design is added (counter). Note also that the CLK and RSTsignals are added to the Optional settings.

  CAUTION If **CLK** or **RST** signals are added to the optional settings in the **HDL settings** page, they should NOT be added to the **I /O** page.

- I/O page: The input and output ports are added. The names in this listing should match the name of the ports in the HDL file that describes the top level entity. Note that as CLK and RST signals are not listed as they are added in the HDL settings page.
- Custom Parameters page: Any parameters used in the design are added here. In this tutorial, the parameter used is WIDTH and the value of that parameter is WIDTH_val which is defined by an equation.

  NOTE Tip: The reason for defining a variable **WIDTH_val** is to be able to control the value of the parameter **WIDTH** (which is the Counter wordlength) in the **I/O** page and in the **Custom Parameters** page at once.

- Run the simulation. Since there are two HDL cosim blocks in the schematic, two instances of ModelSim/Questa processes will be invoked as shown in the figure below:

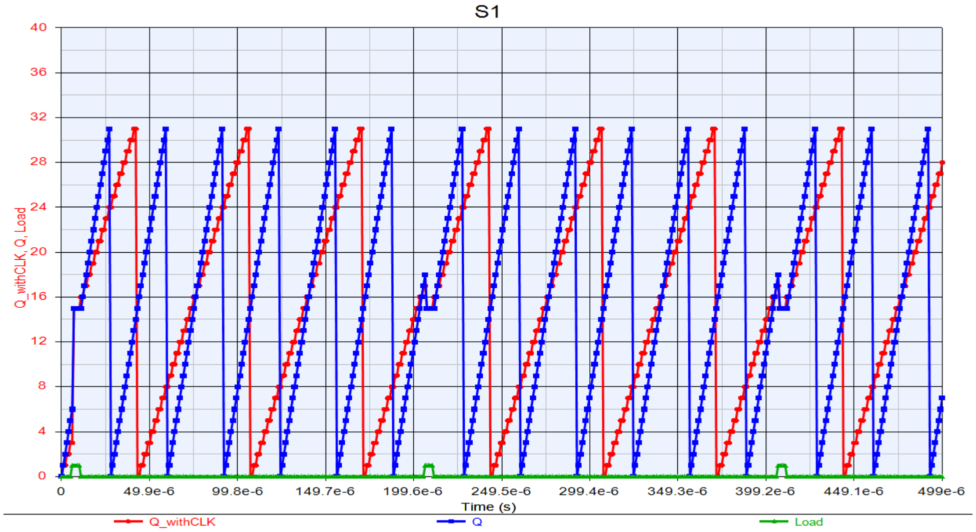


Running Data Flow Analysis, press the Stop button to end calculations

Running Design3 Analysis on Design3:

```
Elapsed time: 2 seconds
Finalizing HDL 'Design3__H1'
HDL 'Design3__H1': Simulating HDL...
HDL 'Design3__H2': Simulating HDL...
```

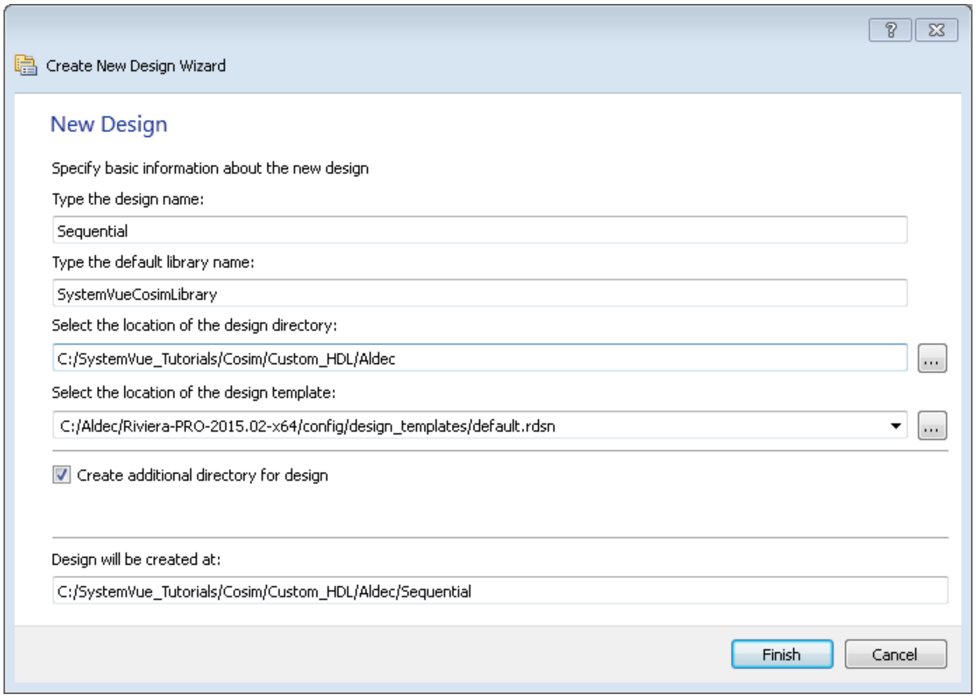
- Double click on the ModelSim_Results graph. Note the difference in output rate between the two counters. The counter which has the clock generated internally can simulate a complete clock cycle for each HDL cosim sample at its input. However, if the clock is generated within SystemVue, the fastest rate that can be simulated is one clock cycle for two SystemVue sample periods, where the first sample holds the 'high' state of the clock and the second sample holds the 'low' state of the clock.



## Co-simulation using Riviera Pro

For this example, we are going to perform the two stages above to cosimulate the counter design using Riviera Pro:
Stage 1 - in Riviera PRO

- Open the *Custom_HDL_Porject* workspace using Riviera PRO that was created for the ADDER combinational logic circuit.
- In the Design Manager window, right click on the workspace *Custom_HDL_Project* and select Add > New Design ... and name the design *Sequential*.
- Apply the settings as shown in the figure below



New Design

Specify basic information about the new design

Type the design name:
Sequential

Type the default library name:
SystemVueCosimLibrary

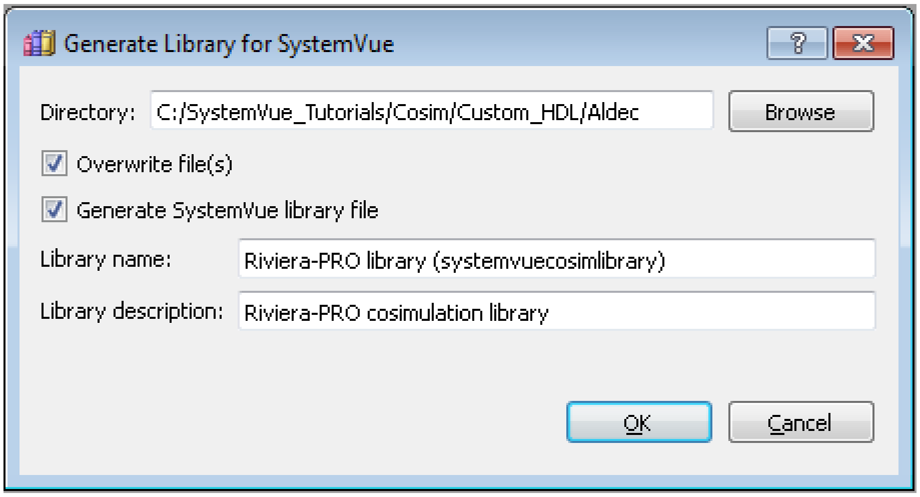
Select the location of the design directory:
C:/SystemVue_Tutorials/Cosim/Custom_HDL/Aldec

Select the location of the design template:
C:/Aldec/Riviera-PRO-2015.02-x64/config/design_templates/default.rdsn

☑ Create additional directory for design

Design will be created at:
C:/SystemVue_Tutorials/Cosim/Custom_HDL/Aldec/Sequential

- In the Design Manager window, select Sequential design, right click and select Add > Existing File ... to add the counter.vhd to it from CoSim\Custom_HDL\HDL_Source\VHDL\Sequential folder.
- Right click on Sequential design and select "Compile Designs"
- Now right click on the Sequential design and select Set Design as Active to bring up the sequential library in the Library window manager
- Go to the Library manager window, right click on the COUNTER in the sequential library and select Generate Library for SystemVue
- Enable the Overwrite file(s) and Generate SystemVue library file options and Name the Library name as *Riviera-PRO library (systemvuecosimlibrary)* and click OK as shown in the figure below:



Generate Library for SystemVue

Directory: C:/SystemVue_Tutorials/Cosim/Custom_HDL/Aldec

☑ Overwrite file(s)

☑ Generate SystemVue library file

Library name: Riviera-PRO library (systemvuecosimlibrary)

Library description: Riviera-PRO cosimulation library

CAUTION This will overwrite the XML library generated previously. However, when loading the newly generated library, both the combinational and sequential designs are listed in the library. If you would like to have the sequential design in a different library, you have to Generate SystemVue library in another folder that does not contain the XML and m files generated for another design.

Stage 2 - in SystemVue

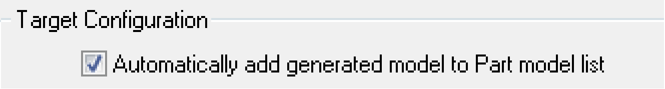
- Open CustomHDL_workspace.wsv.
- Go to Tools > Library Manager... and then unload the previously added RivieraProModelFactory XML library by selecting it and click on Remove Library. Then load the newly generated XML library.
- Find the library: Riviera-PRO library in the current library list.
- On the workspace tree, open the schematic design in CoSim-Sequential > Seq_Design_RPRO. Replace the counter in the design with COUNTER_FXP from Riviera-PRO library.

NOTE Note that the library now contains the parts for both combinational and sequential logic designs.

- Verify the following setting in COUNTER_FXPproperties window:
    - Inputs page: The input ports are added. Set the Integer wordlength to be the same as the specified size in the parameter page ( which is 5 ) and change the cast to Unsigned.
    - Outputs page: Set the Integer wordlength to be the same as the specified size in the parameter page ( which is 5 ), and change the cast to Unsigned.
    - Parameters page: note that the parameter WIDTH is specified here.
    - Clocks page: In this sequential logic design, clock signals are used. Add the CLK signal to the clock window.
    - Custom stimulators page: In this sequential logic design, RST is used as a stimulator signal. Add it the list and configure it as follows ('1 0us, 0 1us').
    - Waveform page: Add all signals on this page
    - Simulation Settings page: To run the cosimulation and invoke Riviera PRO, make sure that Run simulator in batch mode option is not selected
- Run the simulation and verify the results.

## Co-simulating generated HDL code

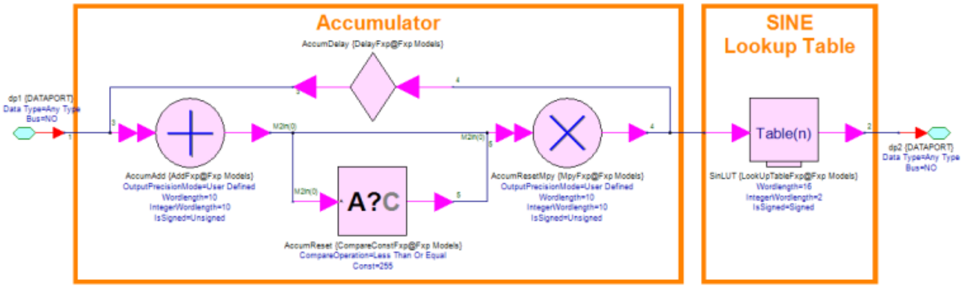
Co-simulating generated HDL code using ModelSim/Questa OR Riviera Pro can be done easily by enabling the check box for “Automatically add generated model to Part model list” in the HDL Code Generation Options dialog as shown below:



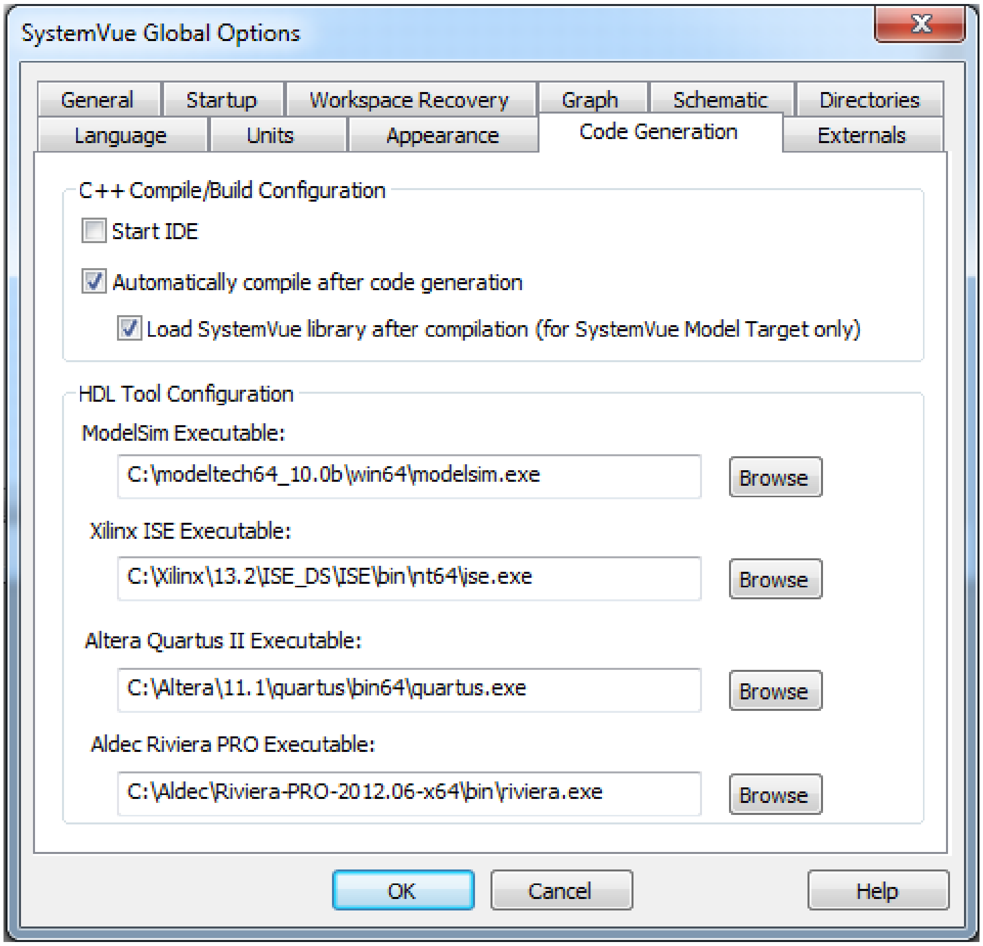
Target Configuration

☑ Automatically add generated model to Part model list

1. Open *NCO_cosim.wsv* located at CoSim\Generated_HDL
2. Double click on the NCO design and take time to explore the components of the NCO design; namely, adder, comparator, multiplier, delay and look-up table as shown in the figure below:



CAUTION Before generating HDL code, verify that the paths to ModelSim/Questa and Riviera Pro Executables are set in the Code Generation page of Global Options as shown below



C++ Compile/Build Configuration

□ Start IDE

☑ Automatically compile after code generation

☑ Load SystemVue library after compilation (for SystemVue Model Target only)

HDL Tool Configuration

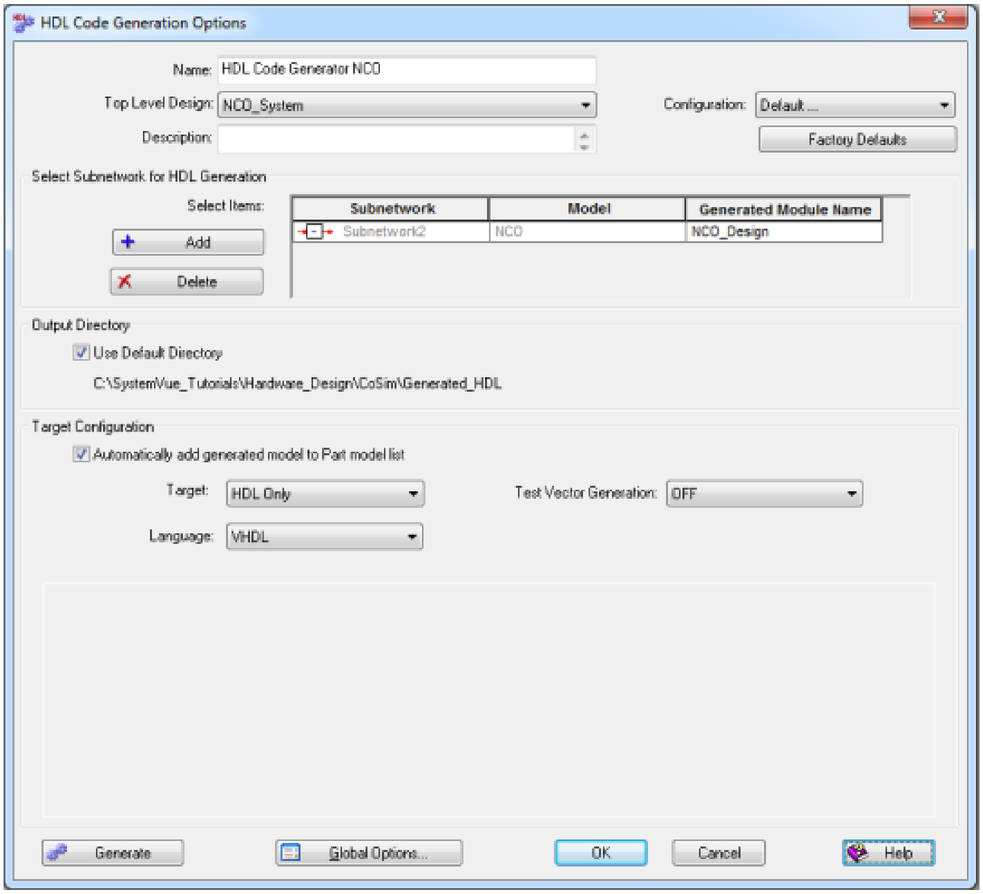
ModelSim Executable: C:\modeltech64_10.0b\win64\modelsim.exe

Xilinx ISE Executable: C:\Xilinx\13.2\ISE_DS\ISE\bin\nt64\ise.exe

Altera Quartus II Executable: C:\Altera\11.1\quartus\bin64\quartus.exe

Aldec Riviera PRO Executable: C:\Aldec\Riviera-PRO-2012.06-x64\bin\riviera.exe

3. Double click on the HDL code generator NCO, verify that the settings are similar to the ones are shown in the figure below:



HDL Code Generation Options

Name: HDL Code Generator NCO

Top Level Design: NCO_System

Configuration: Default ...

Description:

Factory Defaults

Select Subnetwork for HDL Generation

Select Items:

| Subnetwork | Model | Generated Module Name |
|---|---|---|
| Subnetwork2 | NCO | NCO_Design |

Add

Delete

Output Directory

☑ Use Default Directory

C:\SystemVue_Tutorials\Hardware_Design\CoSim\Generated_HDL

Target Configuration

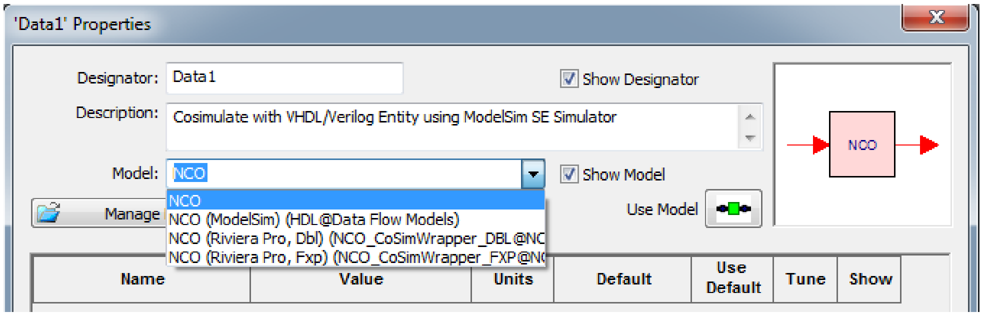
☑ Automatically add generated model to Part model list

Target: HDL Only

Test Vector Generation: OFF

Language: VHDL

4. Click Generate.
5. Double click on NCO sub-network and check the managed model list as shown in the figure below



'Data1' Properties

Designator: Data1

☑ Show Designator

Description: Cosimulate with VHDL/Verilog Entity using ModelSim SE Simulator

Model: NCO

☑ Show Model

Use Model

Manage
NCO
NCO (ModelSim) (HDL@Data Flow Models)
NCO (Riviera Pro, Dbl) (NCO_CoSimWrapper_DBL@NC
NCO (Riviera Pro, Fxp) (NCO_CoSimWrapper_FXP@N(
NCO

| Name | Value | Units | Default | Use Default | Tune | Show |
|---|---|---|---|---|---|---|

As can be seen from the figure, three models are generated

- NCO [ModelSim]: This model is the HDL cosim block configured automatically for cosimulation using ModelSim/Questa
- NCO [Riviera Pro, Dbl]: This model is obtained from the NCO_Library which is loaded automatically after HDL code generation for Riviera Pro cosimulation. The Model is a fixed point model with floating point interface, so the conversion to fixed point is done automatically inside the model.

- NCO [Riviera Pro, Fxp]: This model is obtained from the NCO_Library which is loaded automatically after HDL code generation for Riviera Pro cosimulation. The Model is a fixed point model with fixed point interface, so fixed point converters may be required to connect this model with other non fixed point parts.

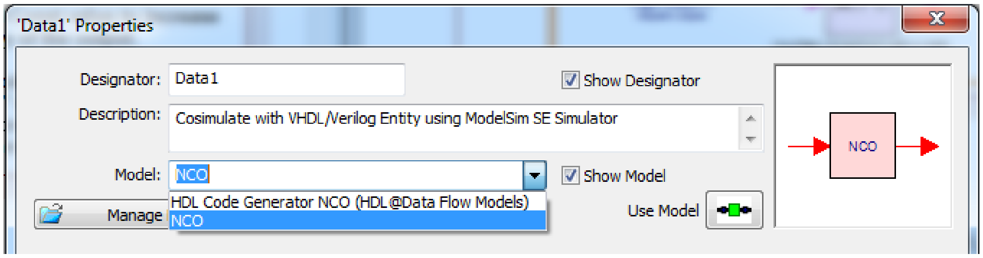
### Co-simulating generated HDL code using ModelSim and Questa

There are two modes of HDL Co-Simulation using ModelSim/Questa, the mode is configured by checking the option Display HDL simulator Graphical User Interface in HDL Settings tab:

- Native Mode: In this mode, ModelSim/Questa GUI will not be invoked and co-simulation using ModelSim/Questa will be executed behind the scene and when finished, results will be updated automatically in SystemVue. This mode is active when the option Display HDL simulator Graphical User Interface in HDL Settings tab is NOT selected.
- Debugging Mode: In this mode, ModelSim/Questa GUI will be invoked and co-simulation can be done step by step using ModelSim/Questa, results will be updated automatically in SystemVue when the whole simulation duration finished in ModelSim/Questa. This mode is active when the option Display HDL simulator Graphical User Interface in HDL Settings tab is selected.

To use ModelSim/Questa for HDL cosimulation, follow the steps below:

1. Since the option “Automatically add generated model to Part model list” is checked in the HDL Code Generation Options dialog in the previous stage, an HDL model is added to the NCO subsystem as shown in the figure below:



'Data1' Properties

Designator: Data1 ☑ Show Designator

Description: Cosimulate with VHDL/Verilog Entity using ModelSim SE Simulator

Model: NCO ☑ Show Model

HDL Code Generator NCO (HDL@Data Flow Models)
NCO

2. Switch the “HDL code generator NCO” model.
3. Take the time to explore the settings of the NCO [ModelSim]model
   a. In HDL files tab: Note the added HDL files in. These HDL files can be found at “C:\SystemVue_Tutorials\HDL_Cosimulation\NCO_HDL\hdl”
   b. Change the compilation Mode to Always
   c. In HDL settings tab: Note the top level entity “NCO_CoSimWrapper” and the optional setting CLK and RST.
   d. Note that the iteration time is automatically set to the time spacing parameter in the analysis.
   e. Enable the option: “Display HDL simulator Graphical User Interface.

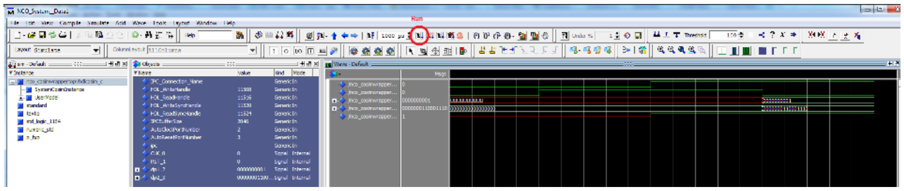
4. Go to Tools>Options... and select the “Code Generation” tab. Set the ModelSim/Questa Executable path. Note that for SystemVue 32 (64) bits, ModelSim/Questa 32 (64) bits should be used.
5. Run the analysis

At this point, ModelSim SE/Questa will be invoked. Using ModelSim/Questa Graphic Interface, you can debug the generated HDL code. SystemVue is halted until the ModelSim/Questa session is terminated.

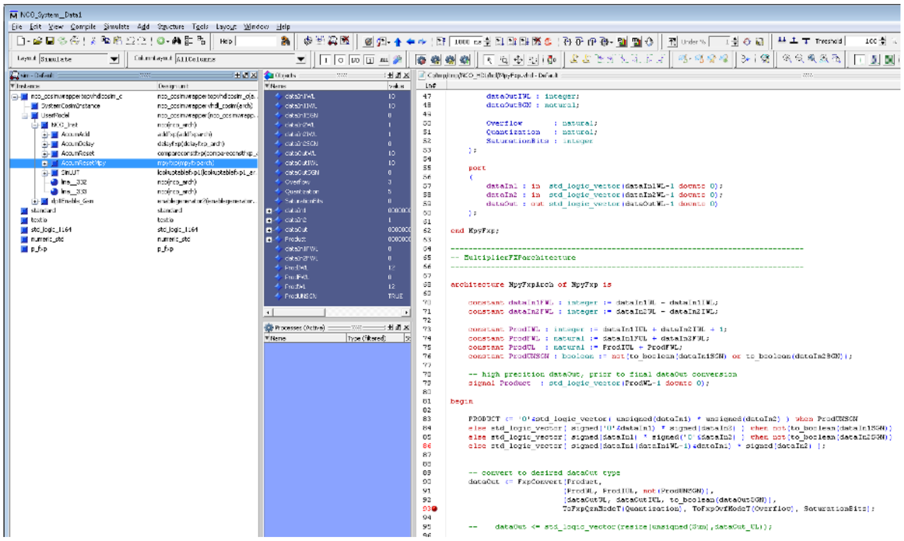
### Debugging in ModelSim SE/Questa

In the next steps, we will take a brief look at one interactive debugging session using the recently invoked ModelSim/Questa session. The debugging will start at time 0. The simulation step is set to the sampling period 1000 ns.

1. Stepping One Clock Cycle
    a. click run , this will run the simulation for one clock cycle, and the reset signal can be observed as in the figure below:



2. Setting Break-Points
    a. Expand UserModel -> NCO_Inst -> AccumResetMpy in the The Structure (sim) window and double click on AccumResetMpy to open MpyFxp.vhd file
    b. Set a breakpoint at the output of the multiplier by clicking at the side of line 93 as shown in the figure below:



```
47          dataOutIWL : integer;
48          dataOutSGN : natural;
49
50          Overflow        : natural;
51          Quantization    : natural;
52          SaturationBits  : integer
53      );
54
55      port
56      (
57          dataIn1 : in  std_logic_vector(dataIn1WL-1 downto 0);
58          dataIn2 : in  std_logic_vector(dataIn2WL-1 downto 0);
59          dataOut : out std_logic_vector(dataOutWL-1 downto 0)
60      );
61
62  end MpyFxp;
63
64  ------------------------------------------------------------------------------
65  -- MultiplierFXParchitecture
66  ------------------------------------------------------------------------------
67
68  architecture MpyFxpArch of MpyFxp is
69
70      constant dataIn1FWL : integer := dataIn1WL - dataIn1IWL;
71      constant dataIn2FWL : integer := dataIn2WL - dataIn2IWL;
72
73      constant ProdIWL : integer := dataIn1IWL + dataIn2IWL + 1;
74      constant ProdFWL : natural := dataIn1FWL + dataIn2FWL;
75      constant ProdWL  : natural := ProdIWL + ProdFWL;
76      constant ProdUNSGN : boolean := not(to_boolean(dataIn1SGN) or to_boolean(dataIn2SGN));
77
78      -- high precition dataOut, prior to final dataOut conversion
79      signal Product  : std_logic_vector(ProdWL-1 downto 0);
80
81  begin
82
83      PRODUCT <= '0'&std_logic_vector( unsigned(dataIn1) * unsigned(dataIn2) ) when ProdUNSGN
84      else std_logic_vector( signed('0'&dataIn1) * signed(dataIn2) ) when not(to_boolean(dataIn1SGN))
85      else std_logic_vector( signed(dataIn1) * signed('0'&dataIn2) ) when not(to_boolean(dataIn2SGN))
86      else std_logic_vector( signed(dataIn1(dataIn1WL-1)&dataIn1) * signed(dataIn2) );
87
88
89      -- convert to desired dataOut type
90      dataOut <= FxpConvert(Product,
91                          (ProdWL, ProdIWL, not(ProdUNSGN)),
92                          (dataOutWL, dataOutIWL, to_boolean(dataOutSGN)),
93                          ToFxpQznModeT(Quantization), ToFxpOvfModeT(Overflow), SaturationBits);
94
95      --    dataOut <= std_logic_vector(resize(unsigned(Sum),dataOut_WL));
96
```

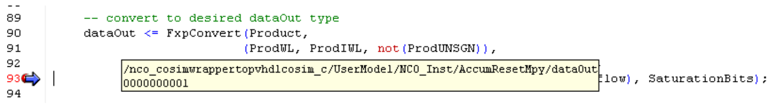
    c. Click on run -all

d. The simulation will stop at line 93 with a blue arrow at the side of that line.

3. Reading Signal Values

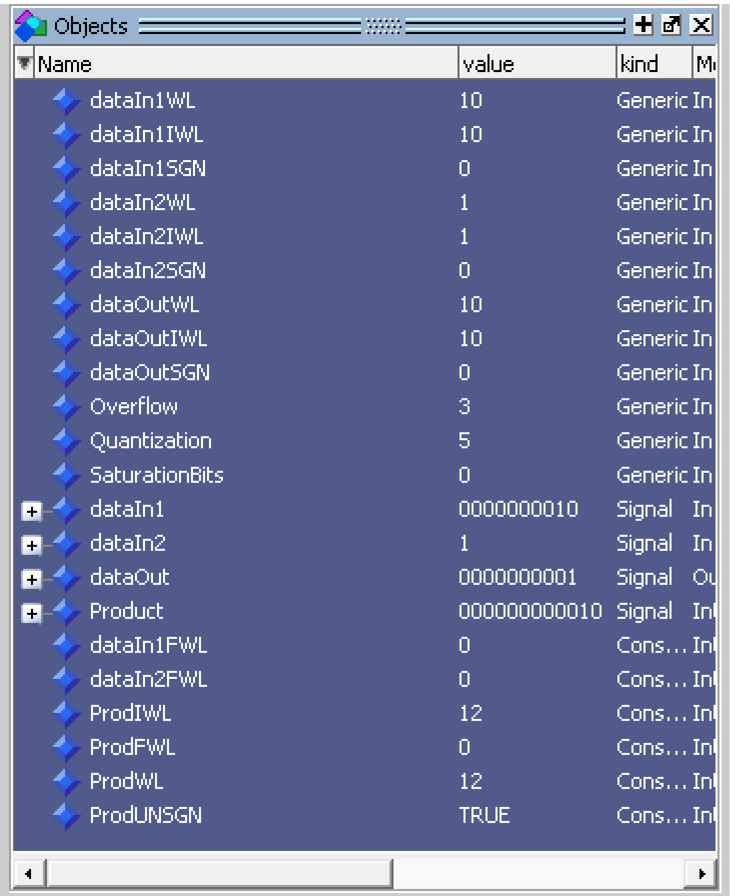
a. When a breakpoint is reached, typically users want to know one or more signal values. There are several options for checking these values:

i. Hover the mouse pointer over dataOut variable at line 90 and a small box with information about the variable will pop up as shown in the figure below:



```
--
89          -- convert to desired dataOut type
90          dataOut <= FxpConvert(Product,
91                               (ProdWL, ProdIWL, not(ProdUNSGN)),
92
93  |          /nco_cosimwrappertopvhdlcosim_c/UserModel/NCO_Inst/AccumResetMpy/dataOut  flow), SaturationBits);
               0000000001
94
```

ii. Also, you can examine the variables of interests in the objects window as shown in the figure below



Objects

| Name | value | kind | M |
|---|---|---|---|
| dataIn1WL | 10 | Generic | In |
| dataIn1IWL | 10 | Generic | In |
| dataIn1SGN | 0 | Generic | In |
| dataIn2WL | 1 | Generic | In |
| dataIn2IWL | 1 | Generic | In |
| dataIn2SGN | 0 | Generic | In |
| dataOutWL | 10 | Generic | In |
| dataOutIWL | 10 | Generic | In |
| dataOutSGN | 0 | Generic | In |
| Overflow | 3 | Generic | In |
| Quantization | 5 | Generic | In |
| SaturationBits | 0 | Generic | In |
| dataIn1 | 0000000010 | Signal | In |
| dataIn2 | 1 | Signal | In |
| dataOut | 0000000001 | Signal | Ou |
| Product | 000000000010 | Signal | In |
| dataIn1FWL | 0 | Cons... | In |
| dataIn2FWL | 0 | Cons... | In |
| ProdIWL | 12 | Cons... | In |
| ProdFWL | 0 | Cons... | In |
| ProdWL | 12 | Cons... | In |
| ProdUNSGN | TRUE | Cons... | In |

b. Step In

i. When Step In is used on the following statement on line 90-93

```
dataOut <= FxpConvert(Product,
                              (ProdWL, ProdIWL,
not(ProdUNSGN)),
                          (dataOutWL,
dataOutIWL, to_boolean(dataOutSGN)),
                          ToFxpQznModeT
(Quantization), ToFxpOvfModeT(Overflow),
SaturationBits);
```

The debugger will first step into function to_boolean (dataOutSGN), then ToFxpQznModeT(Quantization), then ToFxpOvfModeT(Overflow), and then the function call FxpConvert.

ii. Stepping in is performed by clicking on Step {↑} .

c. Run All

i. It is possible to advance the simulation until no simulation events are scheduled or until the simulation is stopped by a code or a signal breakpoint.

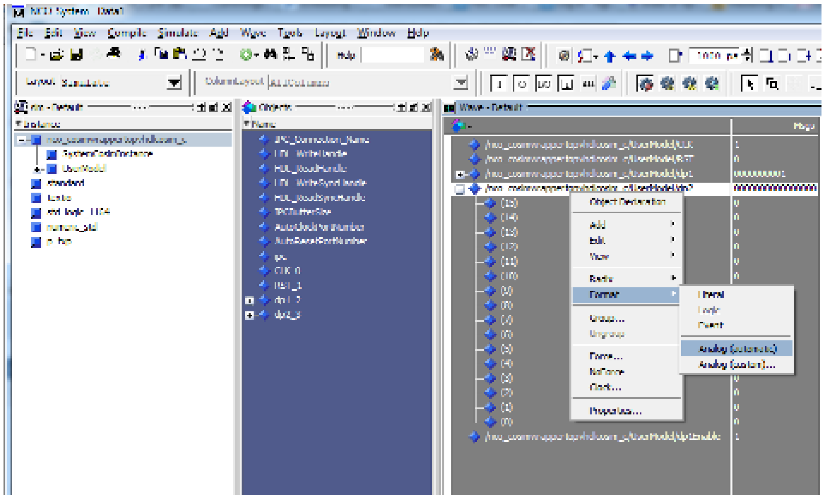
ii. Disable the breakpoint placed on line 93 in MpyFxp.vhd

iii. Run All is performed by clicking on run -all,

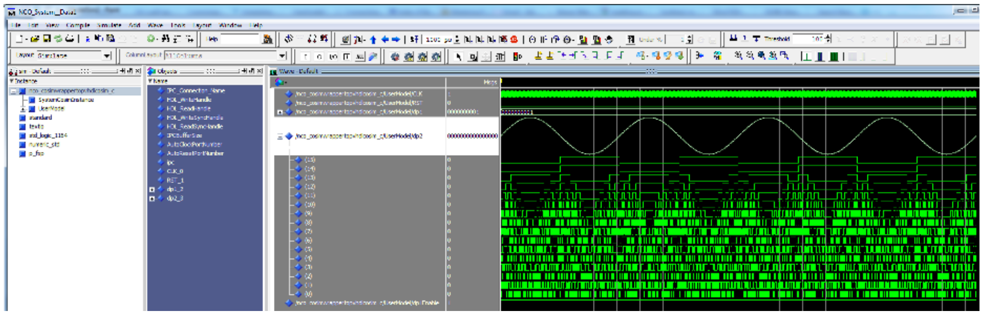
iv. Return to the automatically generated waveform window Wave - Default and click on Wave > Zoom > zoom full (or press F). The final results of the simulation will be displayed as shown in the figure below:

v. click this will run the simulation for 1024 clock cycles as specified by the number of samples inside SystemVue.

vi. To display the Sine wave in analog format, right click on /nco_cosimwrappertopvhdlcosim_c/UserModel/dp2 wave as then select Format > Analog (Automatic) as shown in the figure below:



4. click . The whole Sine wave is displayed as shown in the figure below:



Note that 4 periods of the Sin wave is generated which matches the simulation results in SystemVue.

If Test Vector Generation option is ON, the following files are also generated:

- dp1.txt: test vector input for ModelSim/Questa simulation
- dp2_SVU.txt: the test vector output generated by SystemVue
  As a result, the following files are generated inside NCO_HDL folder:
- HDLCodeGenerator.txt: is a log of the HDL code generation process
- NCO_mtisim.do: this is .do file for ModelSim/Questa that can be executed using the generated batch file NCO_SimTB
- NCO_mtisim: contains the list of generated HDL files in hdl folder.
- NCO_SimTB: This batch file executes ModelSim/Questa and run the sequence of commands specified in the .do file.

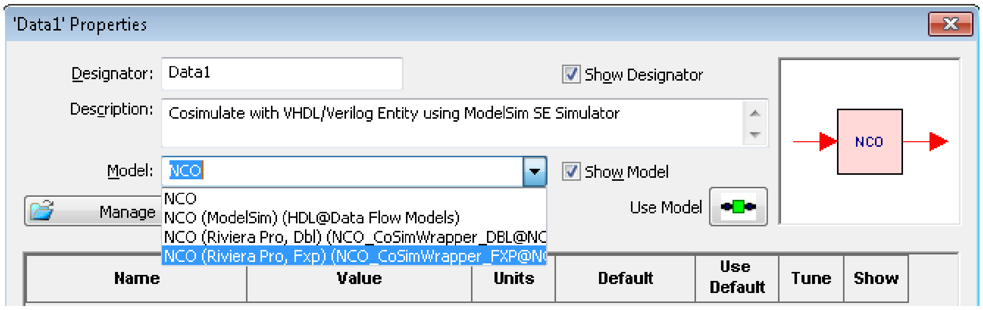
### Co-simulating generated HDL code using Riviera Pro

There are three modes of HDL Co-Simulation using Riviera Pro, the modes are configured in the Simulation Setting tab:

- Native Mode: In this mode, Riviera Pro GUI will not be invoked and co-simulation using Riviera Pro will be executed behind the scene, and when finished, results will be updated automatically in SystemVue. This mode is active when the following options are selected:
  - Run simulator in batch mode
  - Quit simulator after cosimulation
- Batch Mode: In this mode, the user will be able to see the details of co-simulation operations but will not be able to enter commands interactively. This mode is active when Run simulator in batch mode is selected and Quit simulator after cosimulation is not selected.
- Debug Mode: In this mode, Riviera Pro will be invoked in interactive debugging mode, in which users can set breakpoints, step through the design under test and examine the signals and variables inside the design. This mode is active when option Enable debug mode is selected.

To use Riviera Pro for HDL cosimulation, follow the steps below:

1. Double click on NCO sub-network and switch to NCO [Riviera Pro, Fxp] model as shown in the figure below:



'Data1' Properties

Designator: Data1 ☑ Show Designator

Description: Cosimulate with VHDL/Verilog Entity using ModelSim SE Simulator

Model: NCO ☑ Show Model

Manage

NCO
NCO (ModelSim) (HDL@Data Flow Models)
NCO (Riviera Pro, Dbl) (NCO_CoSimWrapper_DBL@NC
NCO (Riviera Pro, Fxp) (NCO_CoSimWrapper_FXP@N(

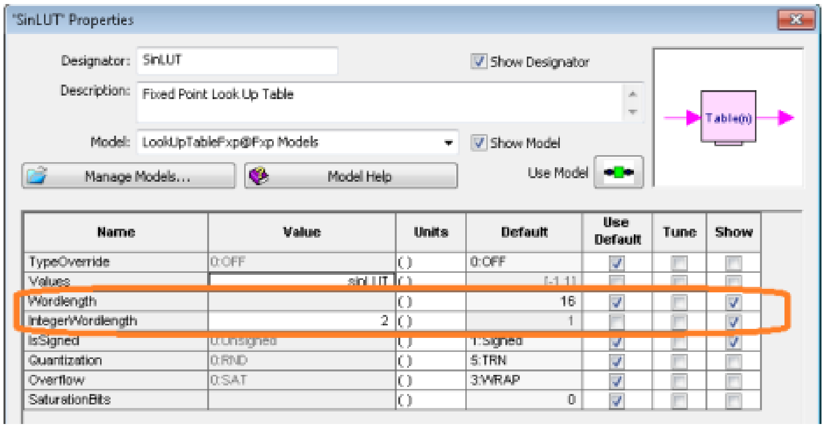
Use Model

| Name | Value | Units | Default | Use Default | Tune | Show |
|---|---|---|---|---|---|---|

**CAUTION** Riviera Pro models are automatically generated when:

- the option “ Automatically add generated model to Part model list” is asserted in the HDL code generation interface and,
- the path to Riviera Pro executable is defined in the "Code Generation Page" of Global Options.,

2. Take the time to explore the settings of NCO [Riviera Pro, Fxp]model
   a. Inputs and Outputs: For ports with wordlength larger than one, the decimal point is set automatically to zero. Users have to manually adjust this parameter based on the design:
      i. For dp1 input port, Point parameter can be left at zero since the input signal coming from Constant part has its output integer wordlength = wordlength = 10 (i.e., no fractional portion).
      ii. For dp2 output port, adjust Point parameter to 14 to match the presentation of the signal coming out of the LookUpTable in the NCO design as shown in the figure below



'SinLUT' Properties

Designator: SinLUT ☑ Show Designator

Description: Fixed Point Look Up Table

Model: LookUpTableFxp@Fxp Models ☑ Show Model

Manage Models... Model Help Use Model

| Name | Value | Units | Default | Use Default | Tune | Show |
|---|---|---|---|---|---|---|
| TypeOverride | 0:OFF | ( ) | 0:OFF | ☑ | □ | □ |
| Values | sinLUT | ( ) | [-1 1] | □ | □ | □ |
| Wordlength |  | ( ) | 16 | ☑ | □ | ☑ |
| IntegerWordlength | 2 | ( ) | 1 | □ | □ | ☑ |
| IsSigned | 0:Unsigned | ( ) | 1:Signed | ☑ | □ | ☑ |
| Quantization | 0:RND | ( ) | 5:TRN | ☑ | □ | □ |
| Overflow | 0:SAT | ( ) | 3:WRAP | ☑ | □ | □ |
| SaturationBits |  | ( ) | 0 | ☑ | □ | □ |

   b. Clocks: Note that the clock port clk is added automatically to the Clocks list so that it can be generated internally by Riviera Pro simulator.
   c. Custom Stimulators: Note that the reset signal RST and clock-enable signal CE are added automatically in the Custom Stimulators list. The RST signal is formulated so that during the first iteration is a logic low

from time 0 to 1/4 times the clock sampling period, a logic high from time 1/4 to 3/4 times of the clock sampling period, a logic low for remaining simulation time. This means that Reset signal is high during first rising edge of the Clock signal that is generated automatically by Riviera PRO simulator. The CE signal is logic high during all the cosimulation time.

d. Waveforms Add all input and output ports to the waveforms list.

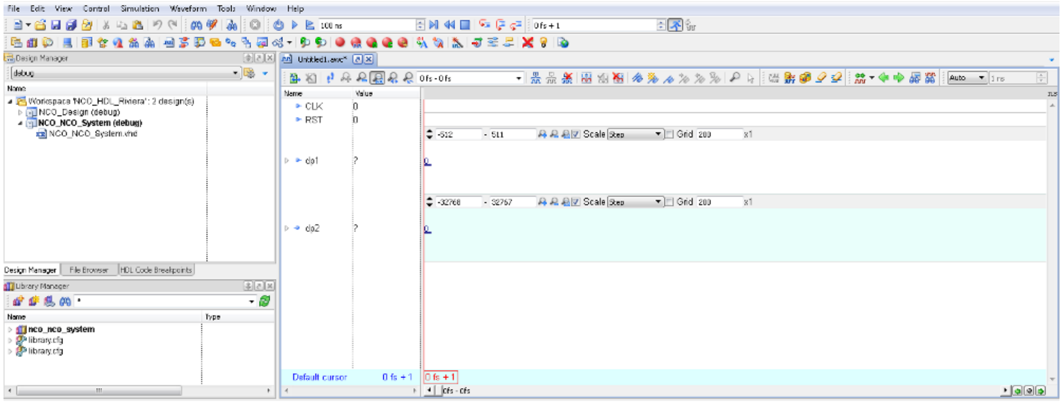
e. Simulation settings: Select Enable debug mode

3. Run the analysis

At this point, Riviera Pro simulator will be invoked to debug the design interactively. SystemVue is halted until the Riviera Pro session is terminated.

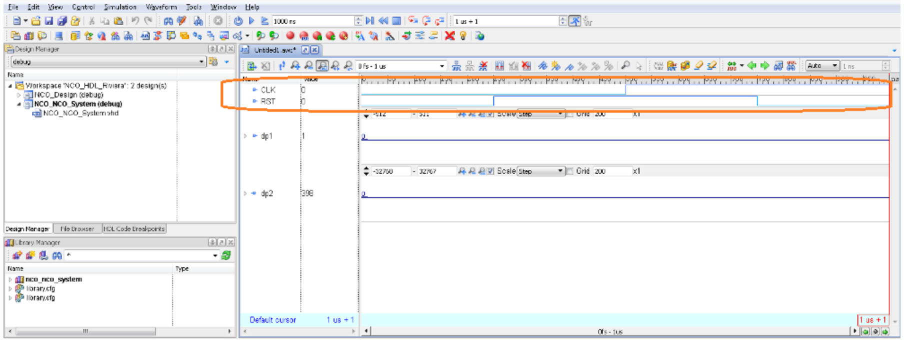
### Debugging in Riviera Pro

In the next steps, we will take a brief look at one interactive debugging session using the recently invoked Riviera Pro session. The debugging will start at time 0 as shown in the figure below



1. Stepping One Clock Cycle

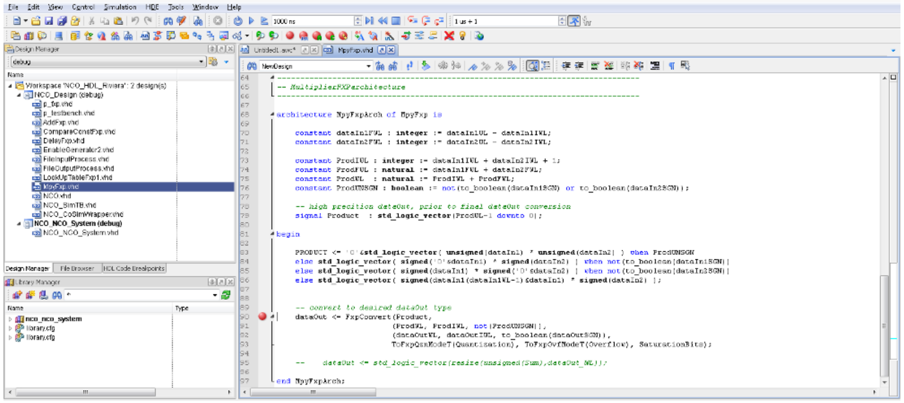
a. The simulation session starts with a pre-configured time steps at 100 ns. Change the time step to 1000 ns and then click on Run For or press (F5). One clock cycle is simulated, and the reset signal can be observed as in the figure below:



2. Setting Break-Points

a. Expand the NCO_Design design in the Design Manager Window and double click on MpyFxp.vhd file

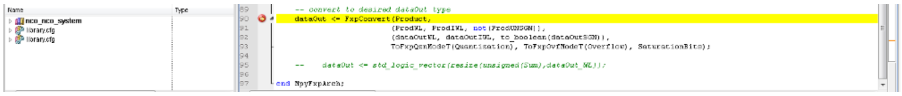
b. Set a breakpoint at the output of the multiplier by double clicking at the side of line 90 as shown in the figure below:



```
64  ----------------------------------------------------------------------------
65  -- MultiplierFXParchitecture
66  ----------------------------------------------------------------------------
67
68  architecture MpyFxpArch of MpyFxp is
69
70      constant dataIn1FWL : integer := dataIn1UL - dataIn1IWL;
71      constant dataIn2FWL : integer := dataIn2UL - dataIn2IWL;
72
73      constant ProdIUL : integer := dataIn1IWL + dataIn2IWL + 1;
74      constant ProdFUL : natural := dataIn1FWL + dataIn2FWL;
75      constant ProdWL  : natural := ProdIWL + ProdFWL;
76      constant ProdUNSGN : boolean := not(to_boolean(dataIn1SGN) or to_boolean(dataIn2SGN));
77
78      -- high precition dateOut, prior to final dateOut conversion
79      signal Product  : std_logic_vector(ProdWL-1 downto 0);
80
81  begin
82
83      PRODUCT <= '0'&std_logic_vector( unsigned(dataIn1) * unsigned(dataIn2) ) when ProdUNSGN
84      else std_logic_vector( signed('0'&dataIn1) * signed(dataIn2) ) when not(to_boolean(dataIn1SGN))
85      else std_logic_vector( signed(dataIn1) * signed('0'&dataIn2) ) when not(to_boolean(dataIn2SGN))
86      else std_logic_vector( signed(dataIn1(dataIn1WL-1)&dataIn1) * signed(dataIn2) );
87
88
89      -- convert to desired dataOut type
90      dataOut <= FxpConvert(Product,
91                            (ProdWL, ProdIWL, not(ProdUNSGN)),
92                            (dataOutWL, dataOutIWL, to_boolean(dataOutSGN)),
93                            ToFxpQsnModeT(Quantization), ToFxpOvfModeT(Overflow), SaturationBits);
94
95      --    dataOut <= std_logic_vector(resize(unsigned(Sum),dataOut_WL));
96
97  end MpyFxpArch;
```

c. Click on Run All or press Shift+(F5)

d. The simulation will stop at line 90 with a yellow arrow at the side of that line as shown in the figure below

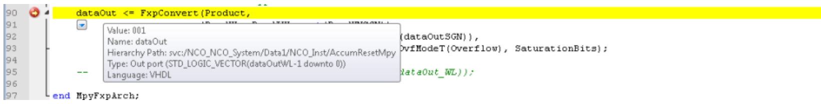


```
89      -- convert to desired dataOut type
90      dataOut <= FxpConvert(Product,
91                            (ProdWL, ProdIWL, not(ProdUNSGN)),
92                            (dataOutWL, dataOutIWL, to_boolean(dataOutSGN)),
93                            ToFxpQsnModeT(Quantization), ToFxpOvfModeT(Overflow), SaturationBits);
94
95      --    dataOut <= std_logic_vector(resize(unsigned(Sum),dataOut_WL));
96
97  end MpyFxpArch;
```

3. Reading Signal Values

a. When a breakpoint is reached, typically users want to know one or more signal values. There are several options for checking these values:

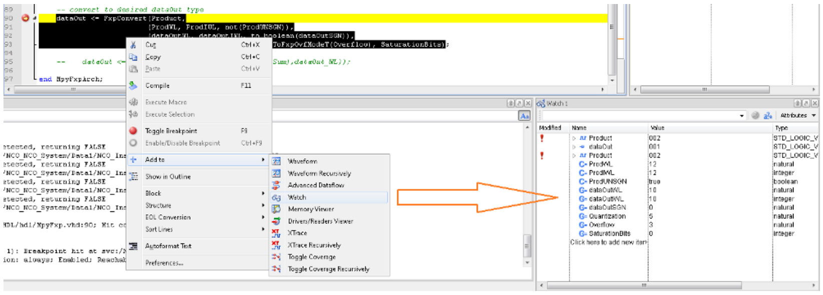
i. Hover the mouse pointer over dataOut variable at line 90 and a small box with information about the variable will pop up as shown in the figure below:



```
90      dataOut <= FxpConvert(Product,
97  end MpyFxpArch;
```

Value: 001
Name: dataOut
Hierarchy Path: svc:/NCO_NCO_System/Data1/NCO_Inst/AccumResetMpy
Type: Out port (STD_LOGIC_VECTOR(dataOutWL-1 downto 0))
Language: VHDL

ii. Select all variables of interests (select lines 90-93) and Right click on the selected area and then select Add to -> Watch and all the variables in the selected area will be added to the watch window as displayed in the figure below:



```
etected, returning FALSE
/NCO_NCO_System/Data1/NCO_Ins
etected, returning FALSE
/NCO_NCO_System/Data1/NCO_Ins
etected, returning FALSE
/NCO_NCO_System/Data1/NCO_Ins
etected, returning FALSE
/NCO_NCO_System/Data1/NCO_Ins

HDL/hdl/MpyFxp.vhd:90: Hit co

 1): Breakpoint hit at svc:/N
ion: always; Enabled; Reachab
```

| Modified | Name | Value | Type |
|---|---|---|---|
| ! | Product | 002 | STD_LOGIC_V |
|  | dataOut | 001 | STD_LOGIC_V |
| ! | Product | 002 | STD_LOGIC_V |
|  | ProdWL | 12 | natural |
|  | ProdIWL | 12 | integer |
|  | ProdUNSGN | true | boolean |
|  | dataOutWL | 10 | natural |
|  | dataOutIWL | 10 | integer |
|  | dataOutSGN | 0 | natural |
|  | Quantization | 5 | natural |
|  | Overflow | 3 | natural |
|  | SaturationBits | 0 | integer |
|  | Click here to add new item |  |  |

b. Step In

   i. When Step In is used on the following statement on line 90

```
dataOut <= FxpConvert(Product,
                           (ProdWL, ProdIWL,
not(ProdUNSGN)),
                           (dataOutWL,
dataOutIWL, to_boolean(dataOutSGN)),
                           ToFxpQznModeT
(Quantization), ToFxpOvfModeT(Overflow),
SaturationBits);
```

   The debugger will first step into function to_boolean (dataOutSGN), then ToFxpQznModeT(Quantization), then ToFxpOvfModeT(Overflow), and then the function call FxpConvert.

   ii. Stepping in is performed by clicking on Step In or press F7.

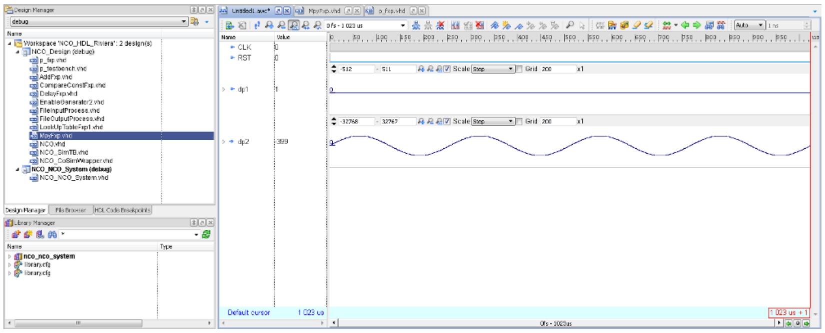
c. Run All

   i. It is possible to advance the simulation until no simulation events are scheduled or until the simulation is stopped by a code or a signal breakpoint.

   ii. Disable the breakpoint placed on line 90 in MpyFxp.vhd

   iii. Run All is performed by clicking on Run All or press Shift+F5

   iv. Return to the automatically generated waveform document Untitled1.awc and click on Zoom To Fit in View or click Ctrl+ *. The final results of the simulation will be displayed as shown in the figure below:



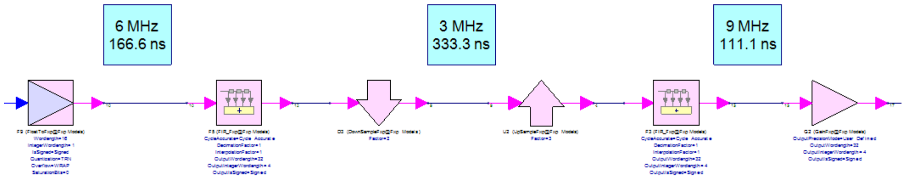
d. Quit Simulator

   i. To end the debugging session and return to SystemVue, click on Stop Simulation or click Shift+F6

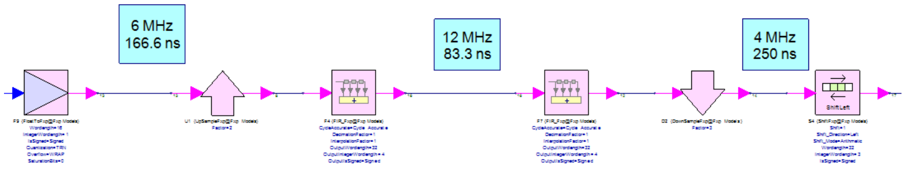
## Co-simulating HDL code of a MultiRate Design

A MultiRate design has input and output ports with different sampling rates. However, the HDL co-simulation blocks are unirate (all ports has the same sampling rate). If the HDL co-simulation blocks are instantiated automatically as part of the HDL code generation, the sampling rate of the co-simulation block is set to the minimum common multiple sampling rates used in the design. The *MultiRate_Filtering.wsv* example demonstrates the HDL cosimulation of two Multirate filters 3 re-sampler and 3:2 resampler. The multirate filters are implemented by cascading interpolating and decimating fixed point filters.

1. Open *MultiRate_Filtering.wsv* located at CoSim\MultiRate
2. Check the two multirate filter designs in D2U3_Design folder and U2D3_Design folder.
   a. In the D2U3_Design case, the input signal is at 6 MHz, then it is downsampled to 3 MHz then upsampled to 9 MHz. Note that the minimum common multiple of 6,3 and 9 is 18. Therefore, the HDL cosimulation block will run at 18 MHz.

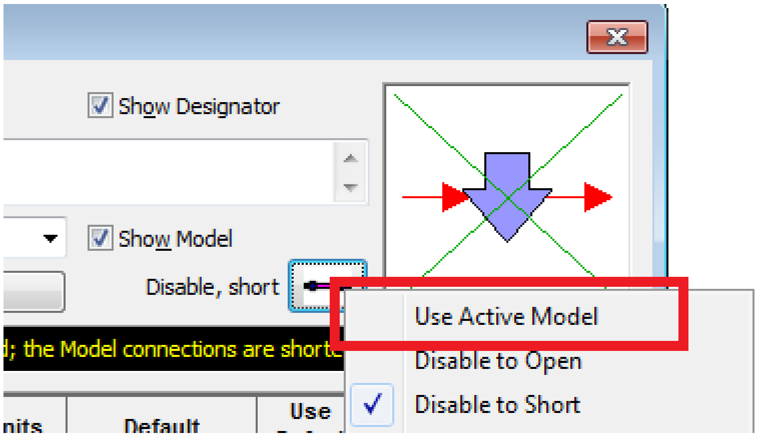


   b. In the U2D3_Design case, the input signal is at 6 MHz, then it is upsampled to 12 MHz then downsampled to 4 MHz. Note that the minimum common multiple of 6,12 and 4 is 12. Therefore, the HDL cosimulation block will run at 12 MHz.



**CAUTION** Note the shorted Upsampler and Downsampler before and after **U2D3** and **D2U3** designs in **Design1** and **Design2** schematics respectively. Those parts are needed when running the HDL cosimulation as will be seen in the following steps.

3. Generate the HDL code using HDL Code Generator1 and HDL Code Generator2 workspace items.
4. Unshort the upsampler and downsampler before and after U2D3 and D2U3 designs in Design1 and Design2 in order to run the cosimulation correctly. The HDL cosimulation blocks are running at different rate than the input signal 6 MHz. To unshort a part, double click on it and then select Use Active Model as shown in the figure below:



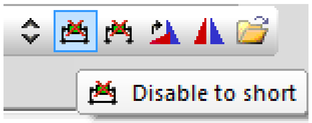
☑ Show Designator

☑ Show Model

Disable, short

d; the Model connections are short

or use the **disable to short** button in the toolbar (see figure below) to toggle between the short status of the part when selected.



**NOTE** The HDL cosim blocks inserted to the designs after generating the HDL code are unirate blocks (the input and output sampling rates are identical). If the input and output sample rates of the original design are different, the generated cosim block will run at a clock rate which is least common multiple of sampling rates used in the design. Therefore, those upsamplers and downsamplers before and after D2U3 and U2D3 should be active during the HDL cosim.

5. Go to managed model list in U2D3 and D2U3 models in Design1 and Design2 schematics, switch to an HDL cosimulation block using Modelsim /Questa(r) or Riviera Pro(r), and run the simulation. Notice the results of the HDL cosimulation in CompareWaveform1 and CompareWaveform2. The figure below is for CompareWaveform1. Note that the graph contains the following traces:

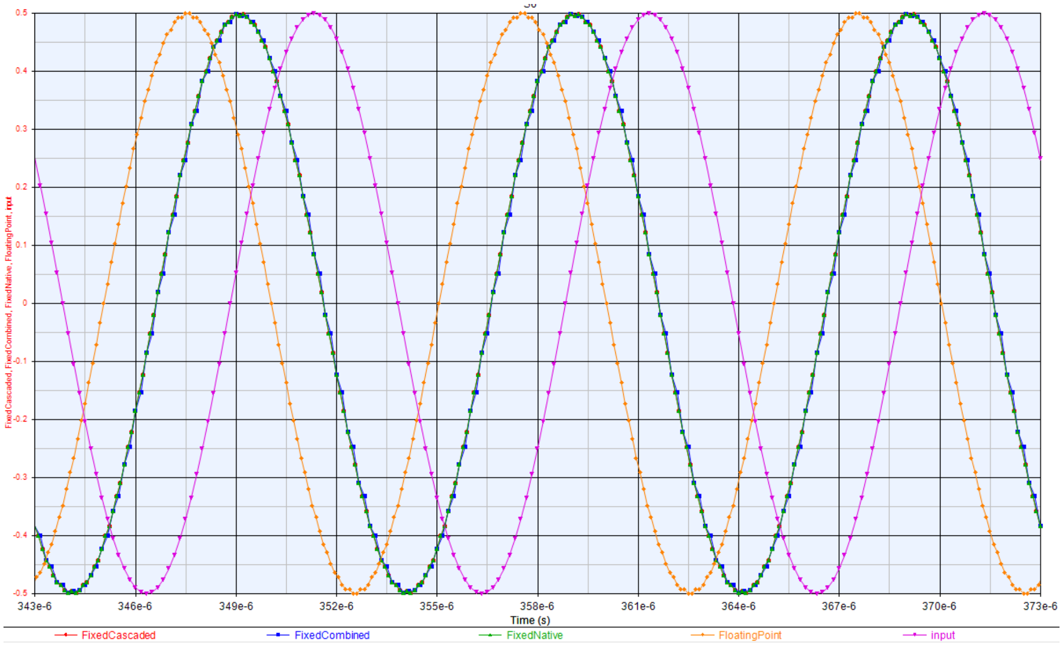
    a. input: This is the trace of the signal generated at 6MHz

    b. FloatingPoint: this is the signal at the output of the interpolation and decimation floating point filters shown in Design1 and Design2 schematics.

    c. FixedNative: this signal is produced by the fixed point filter implementation using Hardware Design library.

    d. FixedCombined: this is the signal that is produced by D2U3 or U2D3 designs or their corresponding HDL cosimulation blocks.

e. FixedCascaded: this is the signal that is produced by the implementation where the decimation and interpolation filters are in two separate subnetworks. This allows the user to test the HDL code generation for each stage independently before generating the code for the whole design using the D2U3 or U2D3 designs.



## Differences of Co-simulation using ModelSim/Questa and Riviera Pro

| | **ModelSim/Questa** | **Riviera Pro** |
|---|---|---|
| Input/Output Data Type setting | Automatic | Manual |
| Cosim across different clock domains | Not Supported | Supported |
| Support IP core gen | Yes | No |
| Bitness compatibility limitation | Limited | Not limited |
| Custom library | Yes | No |
| Automatic Floating/Fixed point conversion | No | Yes |
| Configurable Clock Generation | No | Yes |
| Configurable Reset Generation | No | Yes |
| Configurable Clock-Enable Generation | No | Yes |

| | ModelSim/Questa | Riviera Pro |
|---|---|---|
| Simulator command-line control | Yes | No |

## FPGA Implementation

### Prerequisite

- SystemVue 2012.06 or higher with Hardware Design Library
- Xilinx ISE
- ModelSim SE / Questa

NOTE For software version compatibility, refer to System Requirements

### Associated workspaces

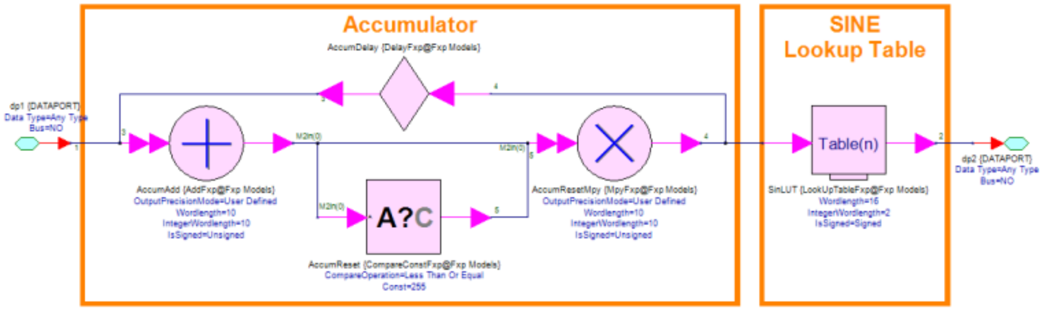
- NCO_Cosim.wsv in Examples\Tutorials\Hardware_Design\CoSim\Generated_HDL

  CAUTION We recommend you to copy the complete directory of **Examples\Tutorials** to your local directory to avoid file writing permission issues on default installation directory.

The FPGA implementation flow is composed of several stages. The following sections detail the steps for each stage:

### Stage 1: Develop the digital design

Open NCO_Cosim.wsv workspace and go to the design in the HDL_Code_Generation folder. Verify the functionality of the design shown in the figure below.

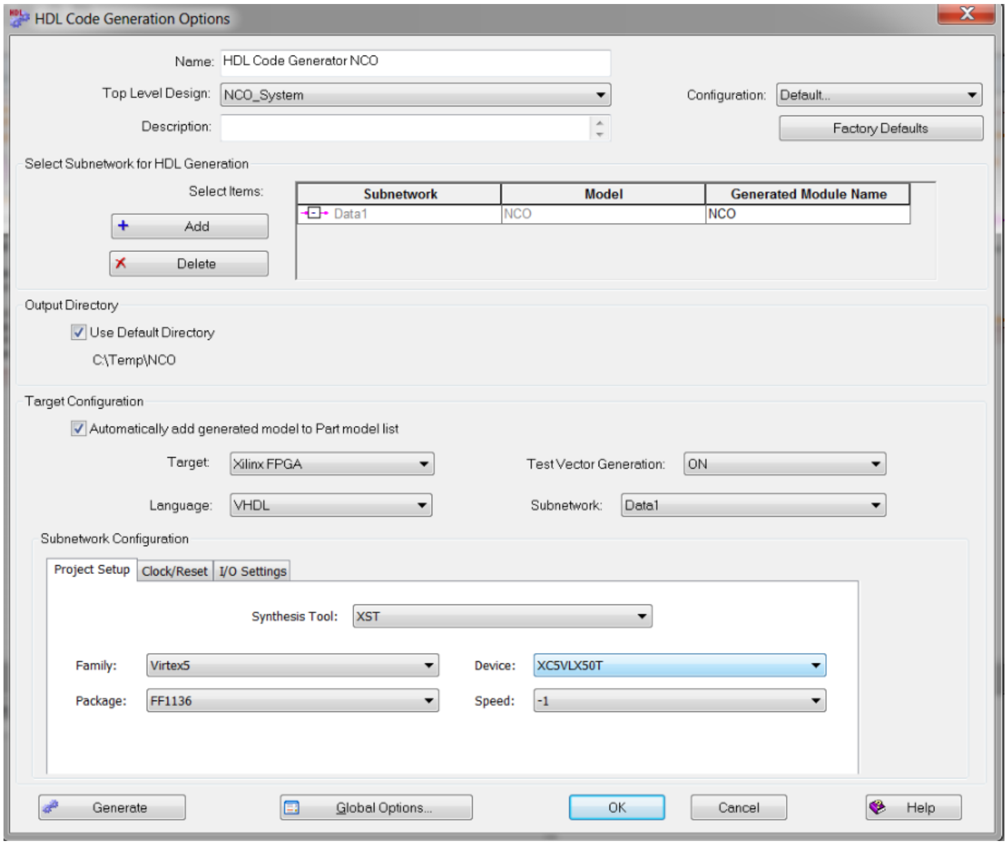


Note that the table in the design stores 256 values of a complete sinusoidal cycle. The logic implemented before the table is used to generate the address for accessing these values. In this context, the maximum number of frequencies that can be supported by this design is 128 different frequencies.

The bit width of input (dp1) in the NCO example is 10 bits. However, we are going to use only the first 4 bits. This can be done by changing the wordlength (and IntegerWordLength) property of the constant input from “10” to “4” in the schematic design of the NCO example.

## Stage 2: Determine the targeted FPGA

After verifying the functionality of the design, double-click on the HDL code generation icon in the workspace tree to open the HDL Code Generation Options dialog as shown in the following figure:



Name: HDL Code Generator NCO

Top Level Design: NCO_System

Configuration: Default...

Description:

Select Subnetwork for HDL Generation

Select Items:

| Subnetwork | Model | Generated Module Name |
|---|---|---|
| Data1 | NCO | NCO |

Output Directory

☑ Use Default Directory

C:\Temp\NCO

Target Configuration

☑ Automatically add generated model to Part model list

Target: Xilinx FPGA

Test Vector Generation: ON

Language: VHDL

Subnetwork: Data1

Subnetwork Configuration

Synthesis Tool: XST

Family: Virtex5

Device: XC5VLX50T

Package: FF1136

Speed: -1

Verify that the NCO design is added to the HDL generation table. Select the desired output directory for generating the HDL code and necessary development files. In the target configuration area, set the parameters as shown in the HDL Code Generation Options dialog above. You can check the “automatically add generated model to part model list” to perform hardware co-simulation. This particular setting is not used within the scope of this document.

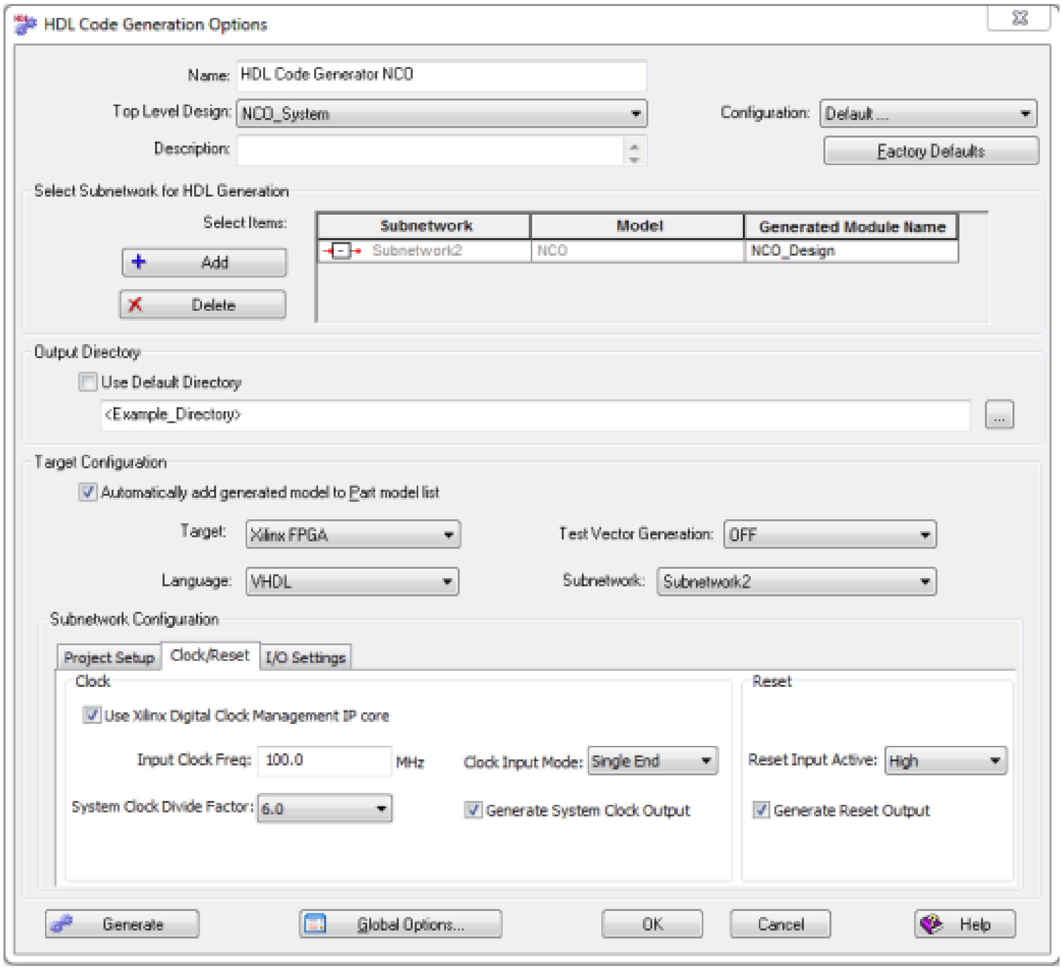
The next steps in this stage involve setting the appropriate values for the design parameters of the Project Setup, Clock/Reset, and I/O settings tabs inside the Subnetwork Configuration area.

- Project Setup:
  Set the Generation Mode to ISE project, and the Synthesis tool to XST. The FPGA settings on the right side are based on the FPGA chip used in the

development board. In this example, we use the ML 506 Xilinx FPGA development board. The settings shown in Project tab of the HDL code generation options window reflect the specifications of the FPGA chip in that board.

- Clock/Reset:
  Set the FPGA system clock to 100 MHz, which is the clock rate of the main oscillator of the ML-506 development board. For demonstration purposes, make sure that the option “Use Xilinx clock management IP core to de-skew clock” is checked. This will result in instantiating a Digital Clock Manager (DCM) Xilinx IP core within the design. Select the System Clock Divide Factor to 6.0.

The push buttons used in the development board are active high. In this example, we are going to use one of these push buttons for reset signal. Therefore, set the “Reset Input Active” to “High”. Also, for demonstration purposes, make sure the “Generate System Clock Output” and “Generate Reset Output” options are checked. Verify that the settings of the Clock/Reset tab match the settings in the figure below:



HDL Code Generation Options

Name: HDL Code Generator NCO

Top Level Design: NCO_System

Configuration: Default ...

Description:

Select Subnetwork for HDL Generation

Select Items:

| Subnetwork | Model | Generated Module Name |
|---|---|---|
| Subnetwork2 | NCO | NCO_Design |

Output Directory

□ Use Default Directory

<Example_Directory>

Target Configuration

☑ Automatically add generated model to Part model list

Target: Xilinx FPGA

Test Vector Generation: OFF

Language: VHDL

Subnetwork: Subnetwork2

Subnetwork Configuration

Clock

☑ Use Xilinx Digital Clock Management IP core

Input Clock Freq: 100.0 MHz

Clock Input Mode: Single End

System Clock Divide Factor: 6.0

☑ Generate System Clock Output

Reset

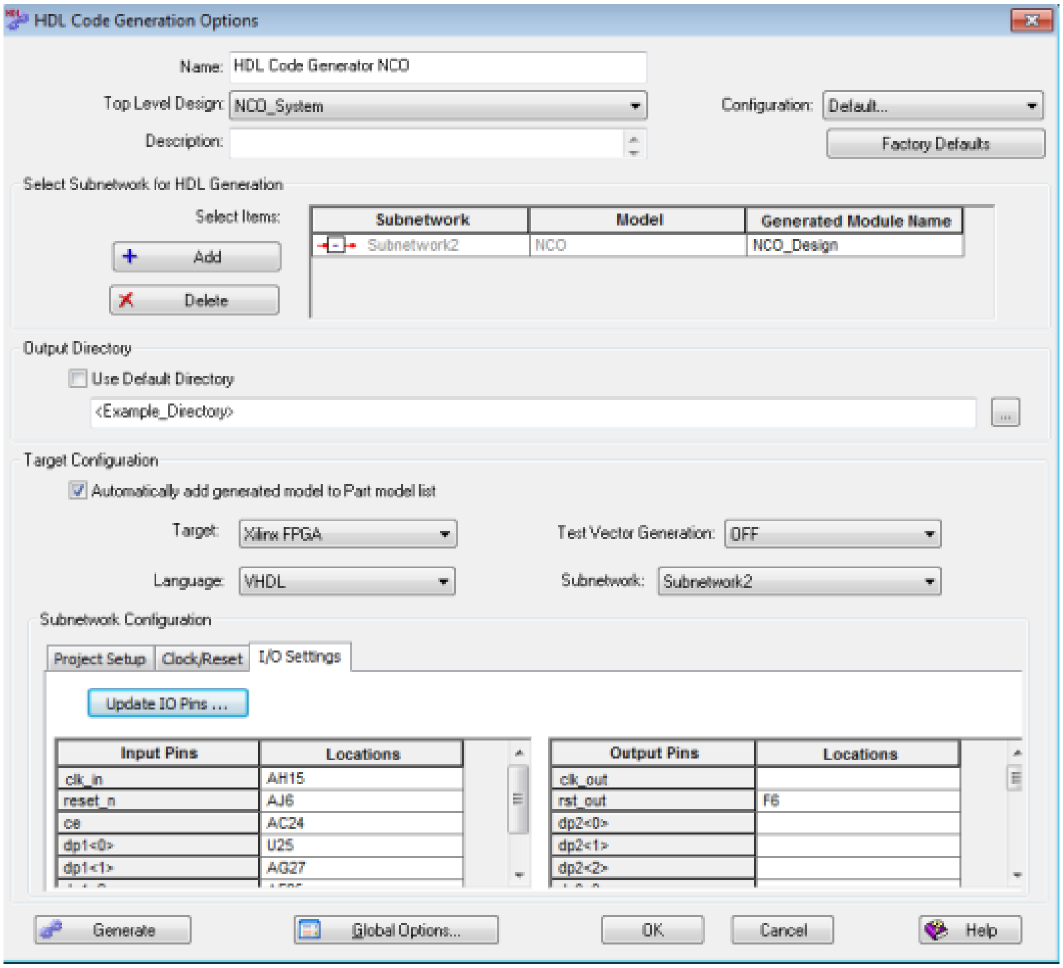
Reset Input Active: High

☑ Generate Reset Output

- I/O Setting:
  In this tab, the I/O pins of the design are mapped to specific FPGA I/O pins. (Click the Update IO Pins... if you don't see any pins.) This setting will be reflected in the constraint files (.ucf) that will be generated as a result of the next stage. The following I/O settings are entered in this tab:

– clk_in: AH15
– reset_in: AJ6
– ce: AC24
– dp<0-3>: U25, AG27, AF25, and AF26
– rst_out: F6

The rest of the I/O signals (including the output of the NCO design) will be mapped to memory within the FPGA using ChipScope debugging tool in Stage 4.



HDL Code Generation Options

Name: HDL Code Generator NCO
Top Level Design: NCO_System
Configuration: Default...
Description:

Select Subnetwork for HDL Generation

Select Items:

| Subnetwork | Model | Generated Module Name |
|---|---|---|
| Subnetwork2 | NCO | NCO_Design |

Output Directory
□ Use Default Directory
<Example_Directory>

Target Configuration
☑ Automatically add generated model to Part model list
Target: Xilinx FPGA
Test Vector Generation: OFF
Language: VHDL
Subnetwork: Subnetwork2

Subnetwork Configuration

| Input Pins | Locations |
|---|---|
| clk_in | AH15 |
| reset_n | AJ6 |
| ce | AC24 |
| dp1<0> | U25 |
| dp1<1> | AG27 |

| Output Pins | Locations |
|---|---|
| clk_out | |
| rst_out | F6 |
| dp2<0> | |
| dp2<1> | |
| dp2<2> | |

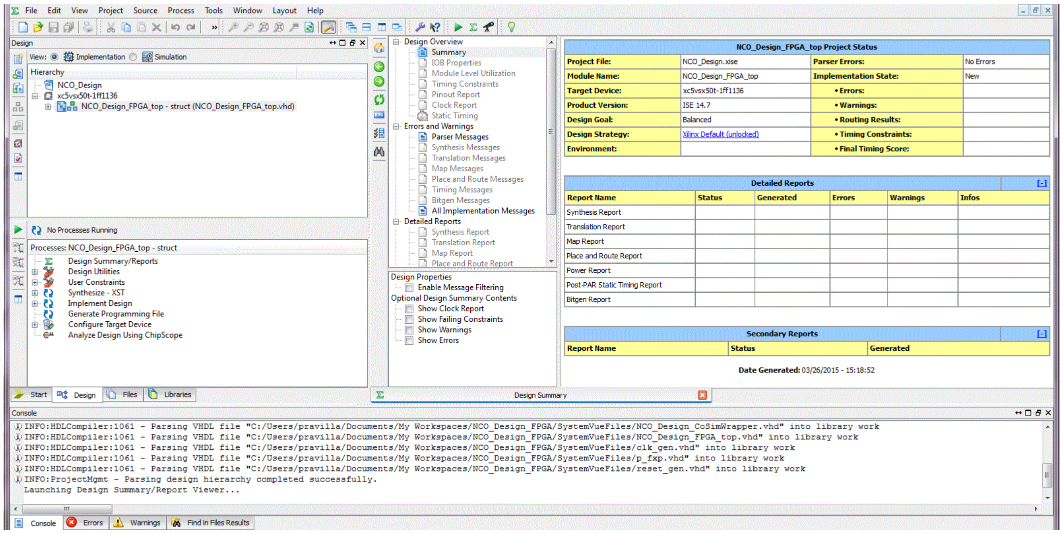
## Stage 3: Generate the HDL code

After including all the necessary settings in the Hardware Code Generation Options Window, make sure that path for the ISE® tool and ModelSim/Questa® is correctly listed in the Global options of Code Generation.

After verifying the path of Xilinx ISE and ModelSim/Questa executables, click on "Generate" to generate the HDL code. The following files will be generated:
NCO_HDL folder: contains the generic HDL code of each element in the design.
NCO_FPGA folder: contains the necessary HDL, constraints and project files used by the Synthesis tool (Xilinx ISE® in this example).

## Stage 4: Resume the conventional FPGA design development in the Synthesis tool

The Xilinx ISE® Synthesis and development tool will be opened automatically as a result of the HDL code generation stage as described in the previous stage. See the figure below:



| NCO_Design_FPGA_top Project Status | | | |
|---|---|---|---|
| **Project File:** | NCO_Design.xise | **Parser Errors:** | No Errors |
| **Module Name:** | NCO_Design_FPGA_top | **Implementation State:** | New |
| **Target Device:** | xc5vsx50t-1ff1136 | **• Errors:** | |
| **Product Version:** | ISE 14.7 | **• Warnings:** | |
| **Design Goal:** | Balanced | **• Routing Results:** | |
| **Design Strategy:** | Xilinx Default (unlocked) | **• Timing Constraints:** | |
| **Environment:** | | **• Final Timing Score:** | |

Detailed Reports [-]

| Report Name | Status | Generated | Errors | Warnings | Infos |
|---|---|---|---|---|---|
| Synthesis Report | | | | | |
| Translation Report | | | | | |
| Map Report | | | | | |
| Place and Route Report | | | | | |
| Power Report | | | | | |
| Post-PAR Static Timing Report | | | | | |
| Bitgen Report | | | | | |

Secondary Reports [-]

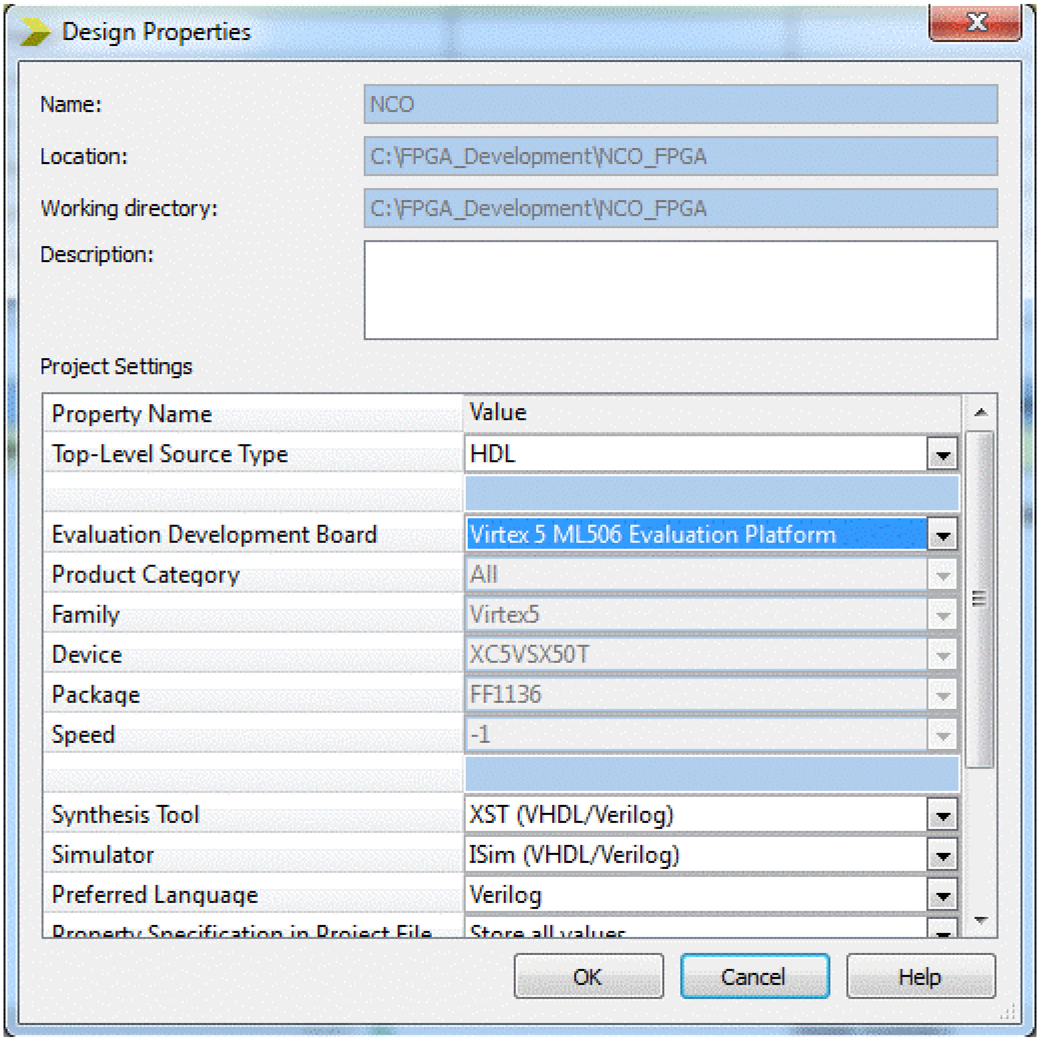
| Report Name | Status | Generated |
|---|---|---|

**Date Generated:** 03/26/2015 - 15:18:52

```
INFO:HDLCompiler:1061 - Parsing VHDL file "C:/Users/pravilla/Documents/My Workspaces/NCO_Design_FPGA/SystemVueFiles/NCO_Design_CoSimWrapper.vhd" into library work
INFO:HDLCompiler:1061 - Parsing VHDL file "C:/Users/pravilla/Documents/My Workspaces/NCO_Design_FPGA/SystemVueFiles/NCO_Design_FPGA_top.vhd" into library work
INFO:HDLCompiler:1061 - Parsing VHDL file "C:/Users/pravilla/Documents/My Workspaces/NCO_Design_FPGA/SystemVueFiles/clk_gen.vhd" into library work
INFO:HDLCompiler:1061 - Parsing VHDL file "C:/Users/pravilla/Documents/My Workspaces/NCO_Design_FPGA/SystemVueFiles/p_fxp.vhd" into library work
INFO:HDLCompiler:1061 - Parsing VHDL file "C:/Users/pravilla/Documents/My Workspaces/NCO_Design_FPGA/SystemVueFiles/reset_gen.vhd" into library work
INFO:ProjectMgmt - Parsing design hierarchy completed successfully.
Launching Design Summary/Report Viewer...
```

**CAUTION** This section assumes that the reader is familiar with the Xilinx ISE development environment. For further details on the use of this tool. Refer to Xilinx ISE and ChipScope documentations.

Take a moment to verify the code generation process. This includes the Design Properties:



Name: NCO

Location: C:\FPGA_Development\NCO_FPGA

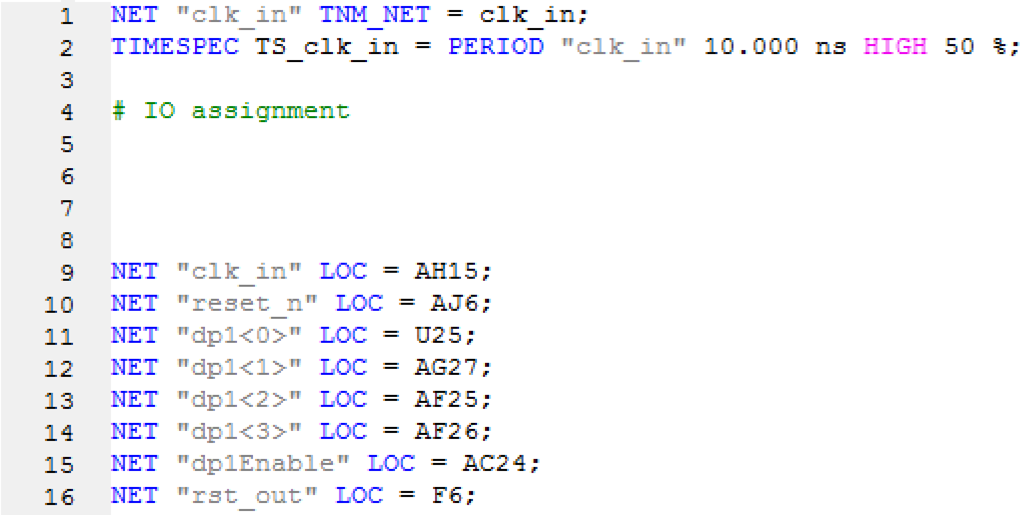
Working directory: C:\FPGA_Development\NCO_FPGA

Description:

Project Settings

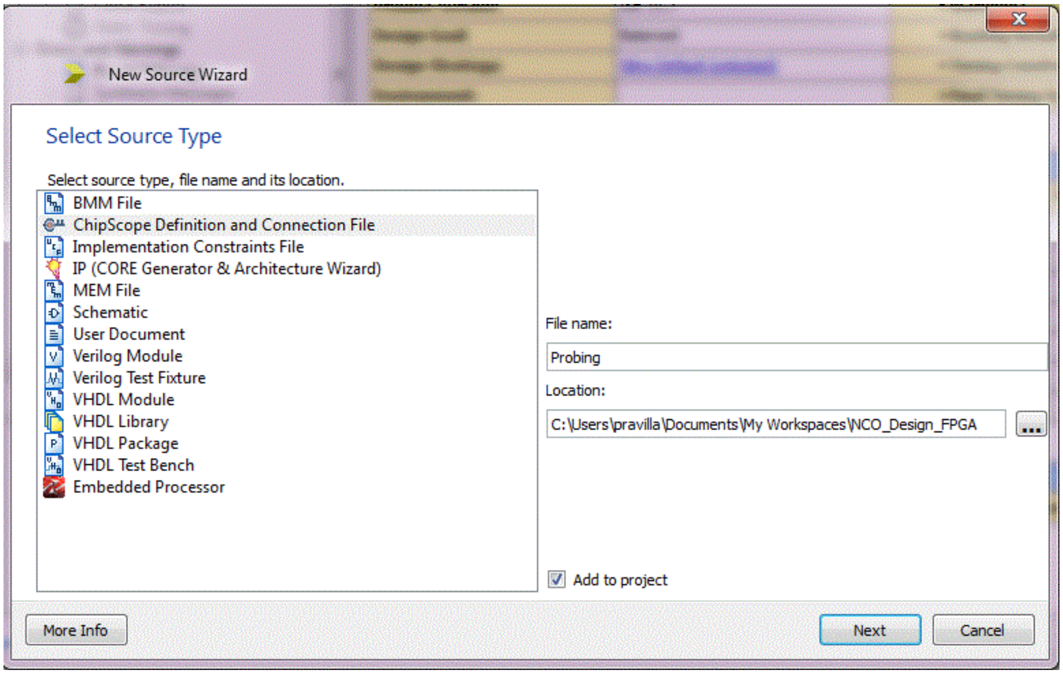
| Property Name | Value |
|---|---|
| Top-Level Source Type | HDL |
| | |
| Evaluation Development Board | Virtex 5 ML506 Evaluation Platform |
| Product Category | All |
| Family | Virtex5 |
| Device | XC5VSX50T |
| Package | FF1136 |
| Speed | -1 |
| | |
| Synthesis Tool | XST (VHDL/Verilog) |
| Simulator | ISim (VHDL/Verilog) |
| Preferred Language | Verilog |
| Property Specification in Project File | Store all values |

and the NCO design constraints:



```
NET "clk_in" TNM_NET = clk_in;
TIMESPEC TS_clk_in = PERIOD "clk_in" 10.000 ns HIGH 50 %;

# IO assignment




NET "clk_in" LOC = AH15;
NET "reset_n" LOC = AJ6;
NET "dp1<0>" LOC = U25;
NET "dp1<1>" LOC = AG27;
NET "dp1<2>" LOC = AF25;
NET "dp1<3>" LOC = AF26;
NET "dp1Enable" LOC = AC24;
NET "rst_out" LOC = F6;
```

In this design project, we are going to add a ChipScope® Definition and Connection File to capture the output of the NCO design as well as other control signals in the design as shown below:



Select Source Type

Select source type, file name and its location.

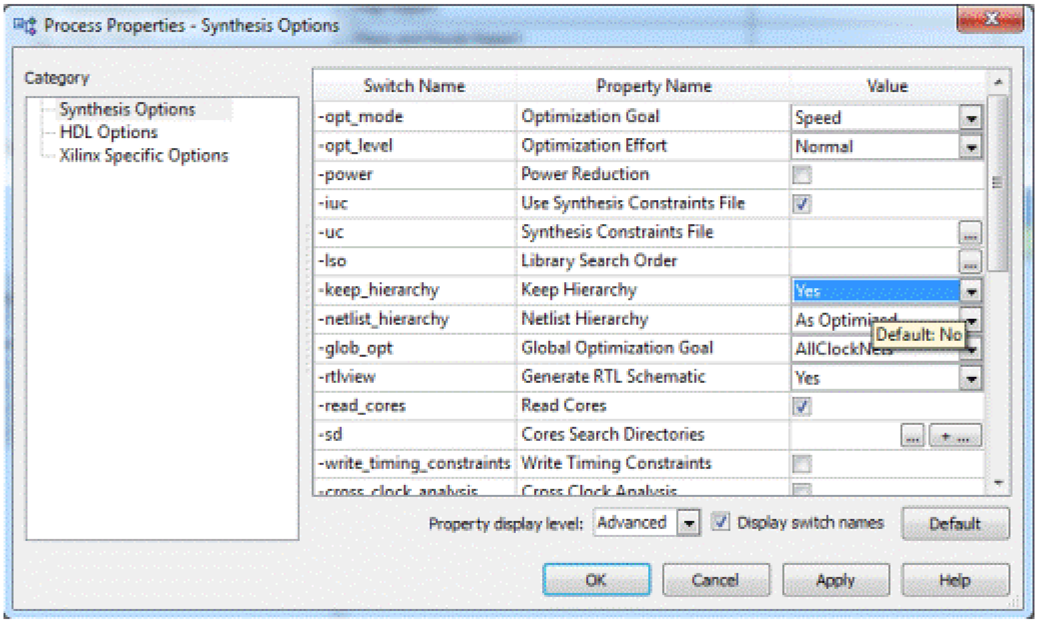
- BMM File
- ChipScope Definition and Connection File
- Implementation Constraints File
- IP (CORE Generator & Architecture Wizard)
- MEM File
- Schematic
- User Document
- Verilog Module
- Verilog Test Fixture
- VHDL Module
- VHDL Library
- VHDL Package
- VHDL Test Bench
- Embedded Processor

File name: Probing

Location: C:\Users\pravilla\Documents\My Workspaces\NCO_Design_FPGA

☑ Add to project

To maintain the hierarchy of the design, make sure to set the corresponding property to “Yes” in the Synthesis process properties as shown below:

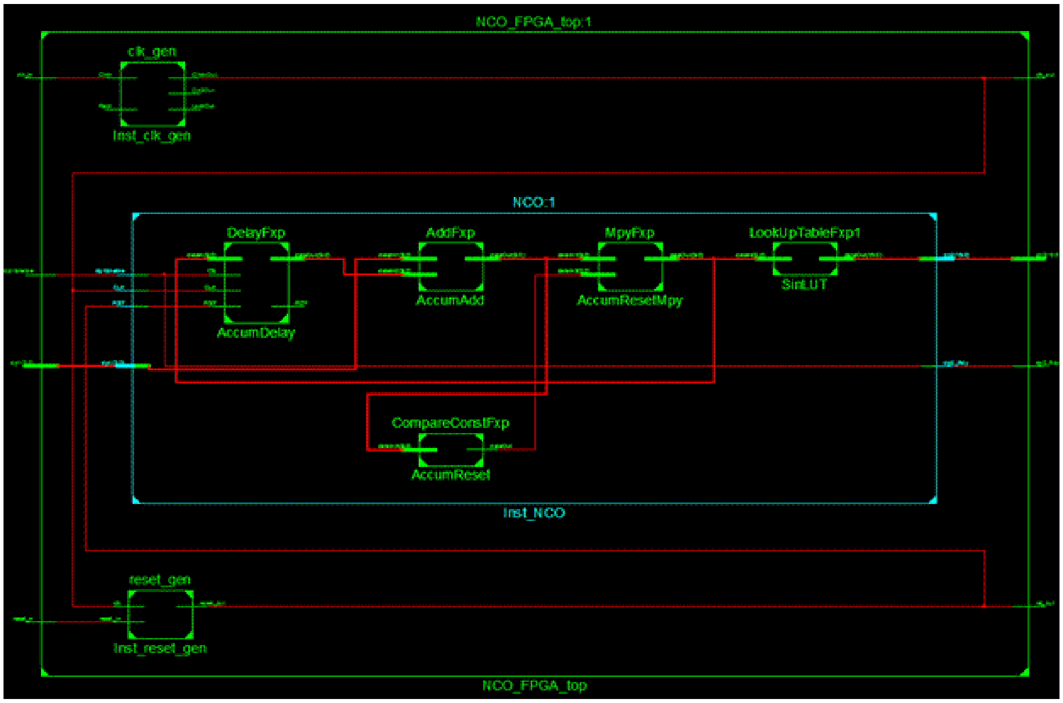


| Switch Name | Property Name | Value |
|---|---|---|
| -opt_mode | Optimization Goal | Speed |
| -opt_level | Optimization Effort | Normal |
| -power | Power Reduction | □ |
| -iuc | Use Synthesis Constraints File | ☑ |
| -uc | Synthesis Constraints File | |
| -lso | Library Search Order | |
| -keep_hierarchy | Keep Hierarchy | Yes |
| -netlist_hierarchy | Netlist Hierarchy | As Optimized |
| -glob_opt | Global Optimization Goal | AllClockNets |
| -rtlview | Generate RTL Schematic | Yes |
| -read_cores | Read Cores | ☑ |
| -sd | Cores Search Directories | |
| -write_timing_constraints | Write Timing Constraints | □ |
| -cross_clock_analysis | Cross Clock Analysis | □ |

Default: No

Property display level: Advanced ☑ Display switch names

This prevents the optimization process from modifying the hierarchy of the design, and simplifies the process of instantiating the signals to the ChipScope ILA element. The results of synthesizing the design are shown in the representation of RTL schematic below. Note the instantiation of the DCM core, the reset generation block, and the digital elements of the NCO design.

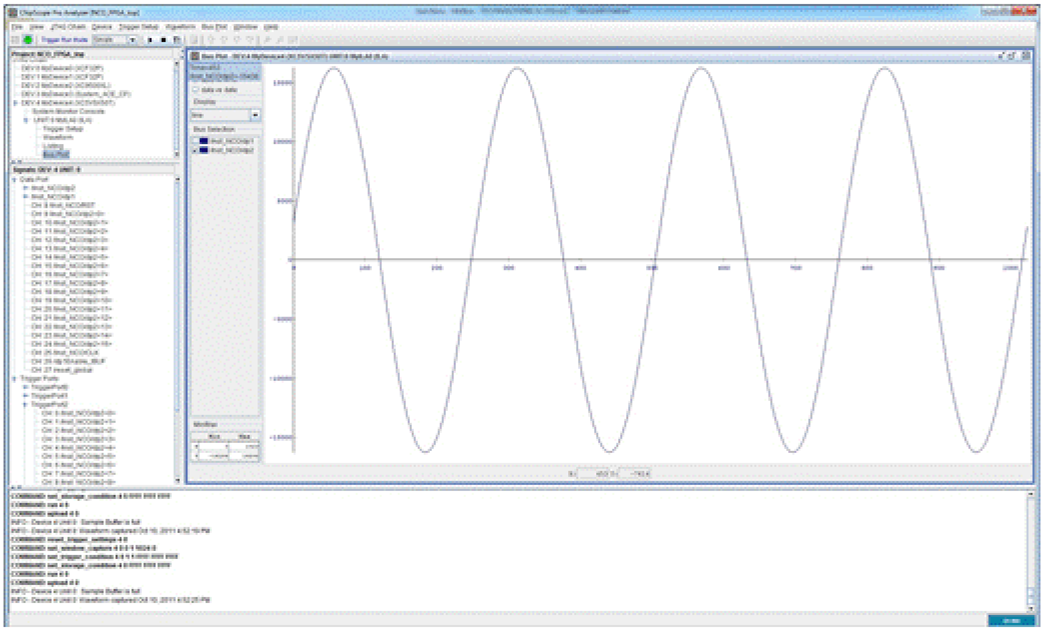


Double-click on NCO_FPGA_top.cfc that was added to the design to open the ChipScope Prop Core Inserter. Add one ILA with 4 trigger ports as follows:

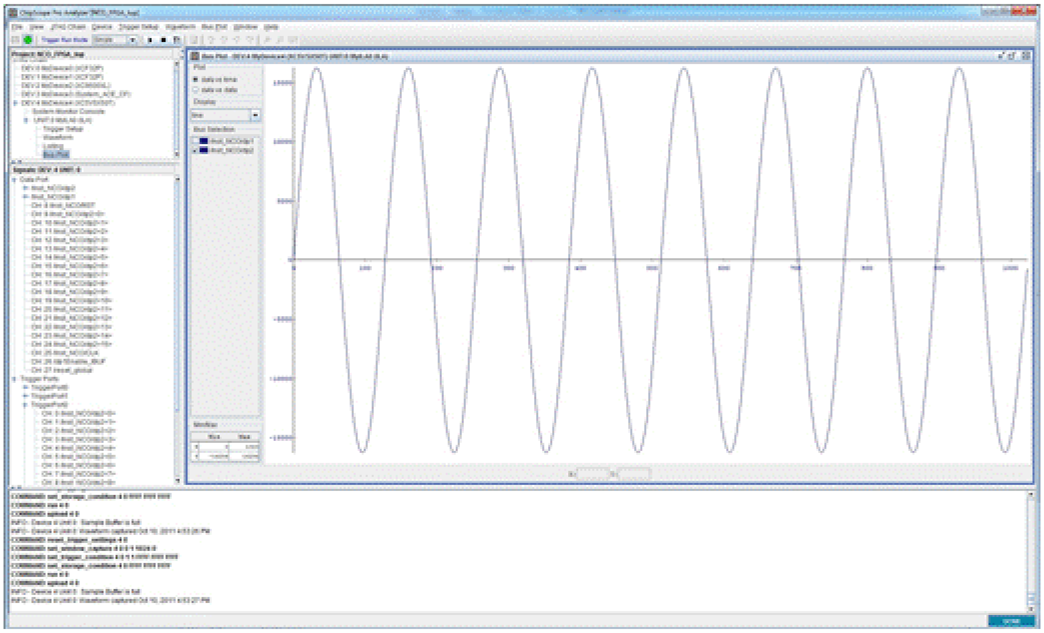
- Trig0: width 1 – Basic with Edges: connect to dp1_Enable_IBUF
- Trig1: width 4 – Basic with Edges: connect to dp1<0-3>
- Trig2: width 16 – Basic with Edges: connect to dp2<0-15>
- Trig3: width 1 – Basic with Edges: connect to reset_in

Set the Data to be as Triggers with Data Depth of 1024, and connect the Trigger Clock to ClkDvOut.
Based on the setting above, generate the bit file using the ISE tool and open chipscope to trigger the design.
Make sure that the associated switch with the dp1_Enable signal is asserted to 1. The clock on the reset push button in the design to reset the NCO. Try different values for dp1 input. The following figures show different sinusoidal signals for the values 1, 2 and 3 in dp1.

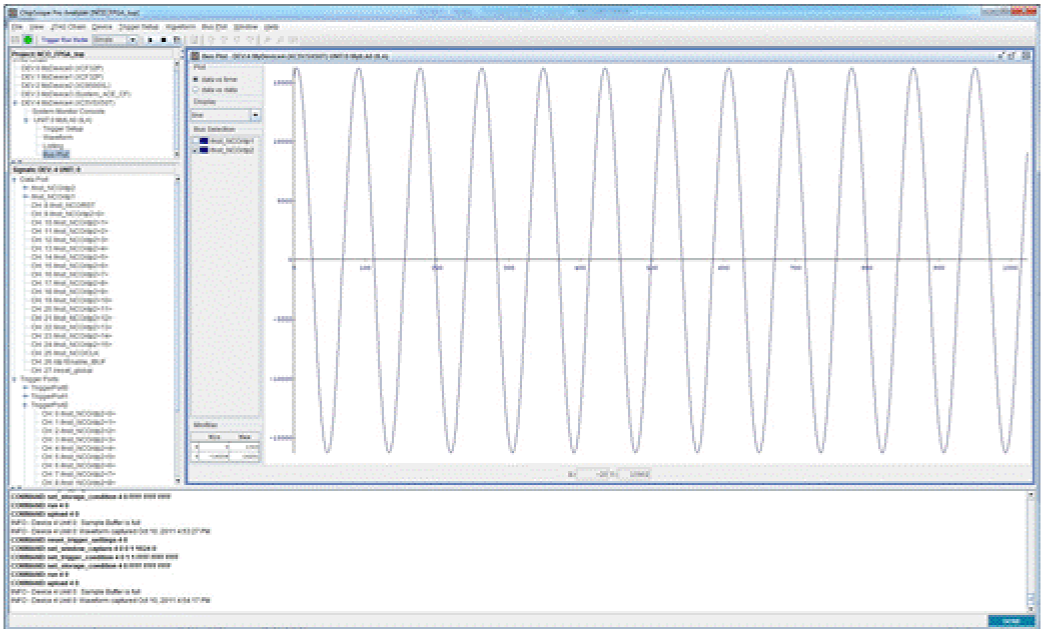
NCO output when dp1 =1:



NCO output when dp1 =2:



NCO output when dp1 =3:



## Using Xilinx IP Cores

### Prerequisite

- SystemVue 2012.06 or higher with Hardware Design Library
- Xilinx ISE
- ModelSim SE / Questa

NOTE For software version compatibility, refer to System Requirements.

### Associated workspaces

- QPSK_FixedP.wsv in Examples\Tutorials\QPSK_Transceiver_Design\Fixed_Point

CAUTION We recommend you to copy the complete directory of **Examples\Tutorials\QPSK_Transceiver_Design\Fixed_Point** to your local directory to avoid file writing permission issues on default installation directory.

### Introduction

This tutorial describes in details how to incorporate Xilinx IP Core Gen in your digital design using XilinxIPIntegrator. The tutorial is composed of two sections: 1) Co-simulating Xilinx IP Core, 2) HDL Code Generation of sub-systems with Xilinx IP Cores.

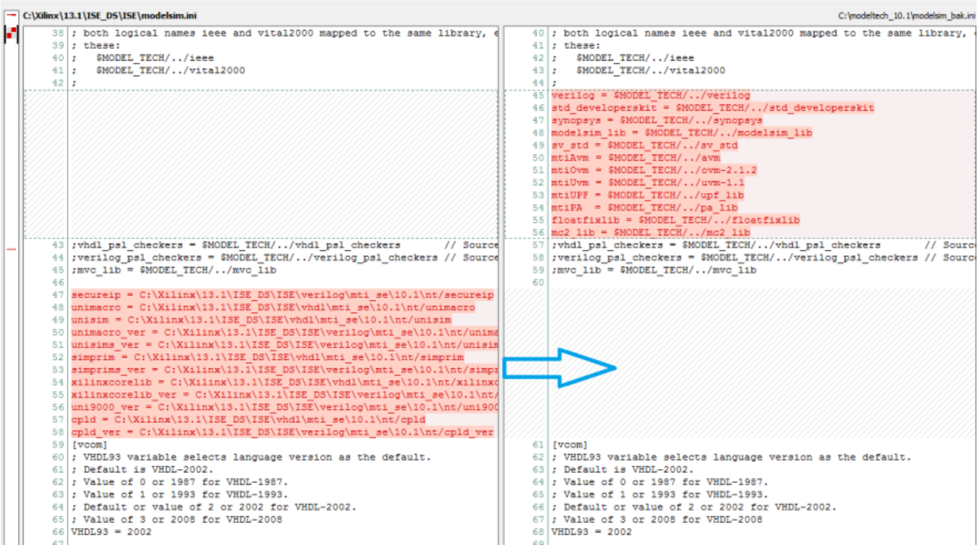
In this tutorial, we demonstrate how to import the Cordic IP Core to use it for phase recovery and phase correction in QPSK transceiver in QPSK_FixedP.wsv workspace. The phase correction uses the ATAN function, while the phase correction is done using the Rotate function. This IP core will replace the Cordic block from the Hardware Design library that is already used in the QPSK_FixedP.wsv workspace.

## Compiling Xilinx IP core simulation libraries

1. Open Windows cmd window and type in: <ISE installation path>\compxlib.exe -arch all -l all -lib all -s mti_se, where <ISE installation path> is the path of ISE installation (it is the same as the path of ise.exe). For example: C:\Xilinx\13.1\ISE_DS\ISE\bin\nt\compxlib.exe -arch all -l all -lib all -s mti_se

   **NOTE** **The above command execution may take a while, so be patient.**

2. Then a new modelsim.ini file will be generated in the current directory of cmd.
3. In ModelSim installation folder, remove the “read-only” priority of ModelSim original modelsim.ini file and open it. (It is recommended to keep a copy of the original ini file here.)
4. Open modelsim.ini file generated by compxlib command, find and copy all path settings of Xilinx libs to original modelsim.ini under ModelSim installation directory (that all uncommented content except “others = $MODEL_TECH/../modelsim.ini”)



C:\Xilinx\13.1\ISE_DS\ISE\modelsim.ini

```
; both logical names ieee and vital2000 mapped to the same library, e
; these:
;   $MODEL_TECH/../ieee
;   $MODEL_TECH/../vital2000
;
;vhdl_psl_checkers = $MODEL_TECH/../vhdl_psl_checkers       // Source
;verilog_psl_checkers = $MODEL_TECH/../verilog_psl_checkers // Source
;mvc_lib = $MODEL_TECH/../mvc_lib

secureip = C:\Xilinx\13.1\ISE_DS\ISE\verilog\mti_se\10.1\nt/secureip
unimacro = C:\Xilinx\13.1\ISE_DS\ISE\vhdl\mti_se\10.1\nt/unimacro
unisim = C:\Xilinx\13.1\ISE_DS\ISE\vhdl\mti_se\10.1\nt/unisim
unimacro_ver = C:\Xilinx\13.1\ISE_DS\ISE\verilog\mti_se\10.1\nt/unima
unisims_ver = C:\Xilinx\13.1\ISE_DS\ISE\verilog\mti_se\10.1\nt/unisim
simprim = C:\Xilinx\13.1\ISE_DS\ISE\vhdl\mti_se\10.1\nt/simprim
simprims_ver = C:\Xilinx\13.1\ISE_DS\ISE\verilog\mti_se\10.1\nt/simpr
xilinxcorelib = C:\Xilinx\13.1\ISE_DS\ISE\vhdl\mti_se\10.1\nt/xilinxc
xilinxcorelib_ver = C:\Xilinx\13.1\ISE_DS\ISE\verilog\mti_se\10.1\nt/
uni9000_ver = C:\Xilinx\13.1\ISE_DS\ISE\verilog\mti_se\10.1\nt/uni900
cpld = C:\Xilinx\13.1\ISE_DS\ISE\vhdl\mti_se\10.1\nt/cpld
cpld_ver = C:\Xilinx\13.1\ISE_DS\ISE\verilog\mti_se\10.1\nt/cpld_ver
[vcom]
; VHDL93 variable selects language version as the default.
; Default is VHDL-2002.
; Value of 0 or 1987 for VHDL-1987.
; Value of 1 or 1993 for VHDL-1993.
; Default or value of 2 or 2002 for VHDL-2002.
; Value of 3 or 2008 for VHDL-2008
VHDL93 = 2002
```

C:\modeltech_10.1\modelsim_bak.ini

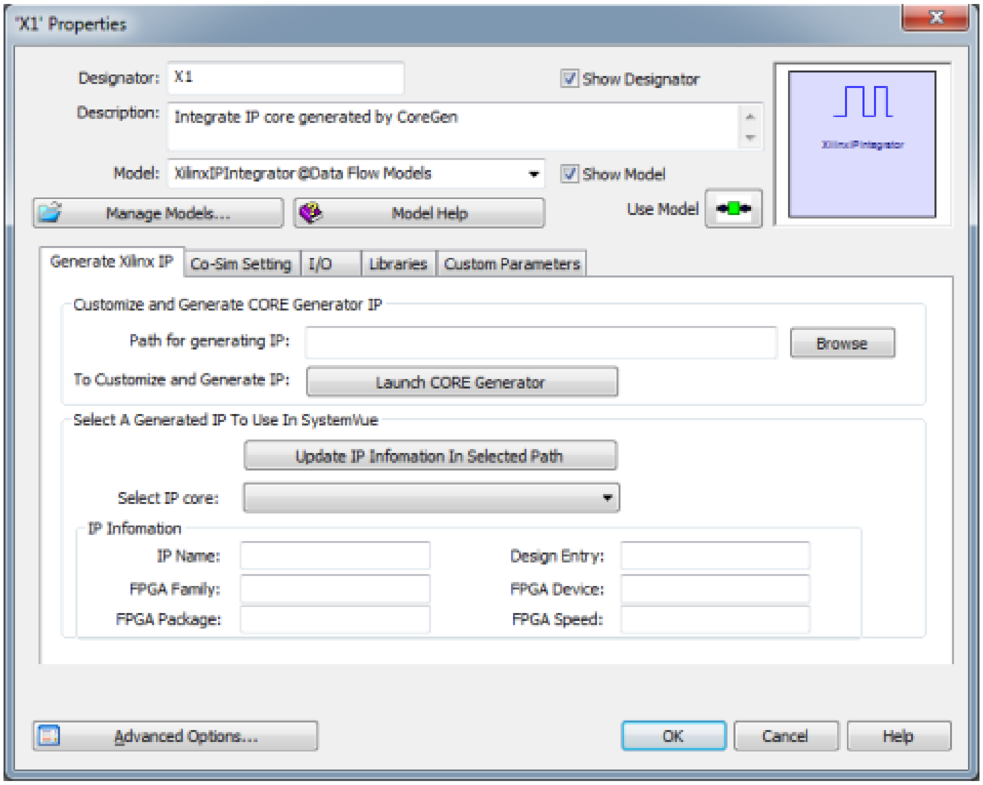
```
; both logical names ieee and vital2000 mapped to the same library, e
; these:
;   $MODEL_TECH/../ieee
;   $MODEL_TECH/../vital2000
;
verilog = $MODEL_TECH/../verilog
std_developerskit = $MODEL_TECH/../std_developerskit
synopsys = $MODEL_TECH/../synopsys
modelsim_lib = $MODEL_TECH/../modelsim_lib
sv_std = $MODEL_TECH/../sv_std
mtiAvm = $MODEL_TECH/../avm
mtiOvm = $MODEL_TECH/../ovm-2.1.2
mtiUvm = $MODEL_TECH/../uvm-1.1
mtiUPF = $MODEL_TECH/../upf_lib
mtiPA  = $MODEL_TECH/../pa_lib
floatfixlib = $MODEL_TECH/../floatfixlib
mc2_lib = $MODEL_TECH/../mc2_lib
;vhdl_psl_checkers = $MODEL_TECH/../vhdl_psl_checkers       // Sourc
;verilog_psl_checkers = $MODEL_TECH/../verilog_psl_checkers // Sourc
;mvc_lib = $MODEL_TECH/../mvc_lib

[vcom]
; VHDL93 variable selects language version as the default.
; Default is VHDL-2002.
; Value of 0 or 1987 for VHDL-1987.
; Value of 1 or 1993 for VHDL-1993.
; Default or value of 2 or 2002 for VHDL-2002.
; Value of 3 or 2008 for VHDL-2008
VHDL93 = 2002
```

5. Save the updated modelsim.ini.

## Co-simulating Xilinx IP Core

1. Open the QPSK_FixedP.wsv

2. Save the workspace in a write-permissible area if you opened it from the installation examples directory. This tutorial will refer to the full path of that area as <Example_Local_Directory>.
3. Go to Rx folder in the workspace tree and open qpsk_demod model, and find the CORDIC blocks in the qpsk_demod model.
4. Drag and drop XilinxIPIntegrator part from the Hardware Design library. Double click on this part to open the part's Properties dialog:



'X1' Properties

Designator: X1

☑ Show Designator

Description: Integrate IP core generated by CoreGen

Model: XilinxIPIntegrator@Data Flow Models

☑ Show Model

Customize and Generate CORE Generator IP

Path for generating IP:

Select A Generated IP To Use In SystemVue

Select IP core:

IP Infomation

IP Name:

Design Entry:

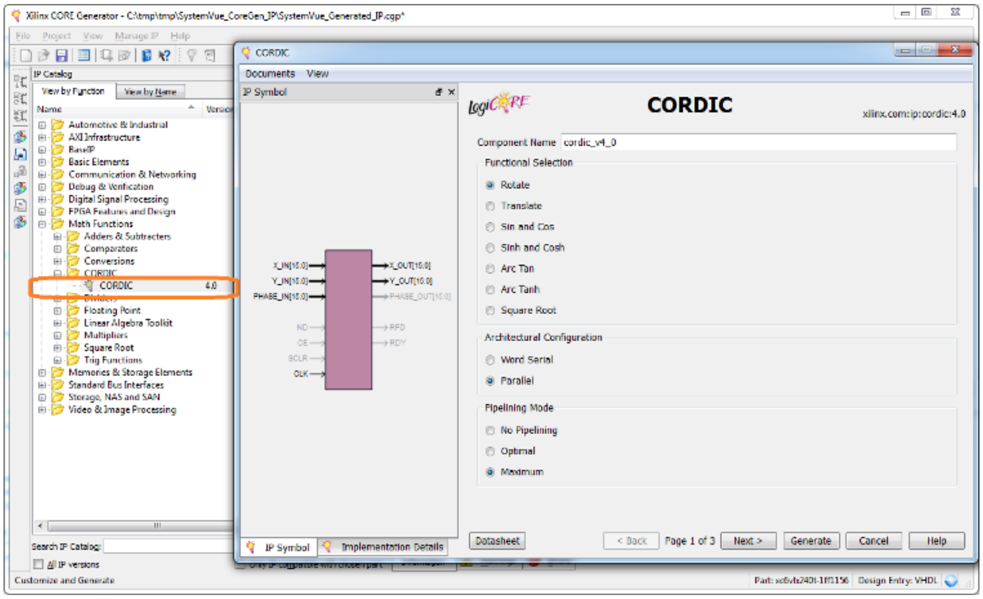
FPGA Family:

FPGA Device:

FPGA Package:

FPGA Speed:

5. Define the Path for generating IP as <Example_Local_Directory>.
6. Click on Launch CORE Generator
7. Go to Project -> Project Options to configure the targeted FPGA, type of the generated HDL language and other simulation and code generation settings: (For the tutorial, the default settings are used).
8. Find CORDIC IP core in Math Functions -> CORDIC and double click on it. The CORDIC IP core user interface will be invoked as shown below



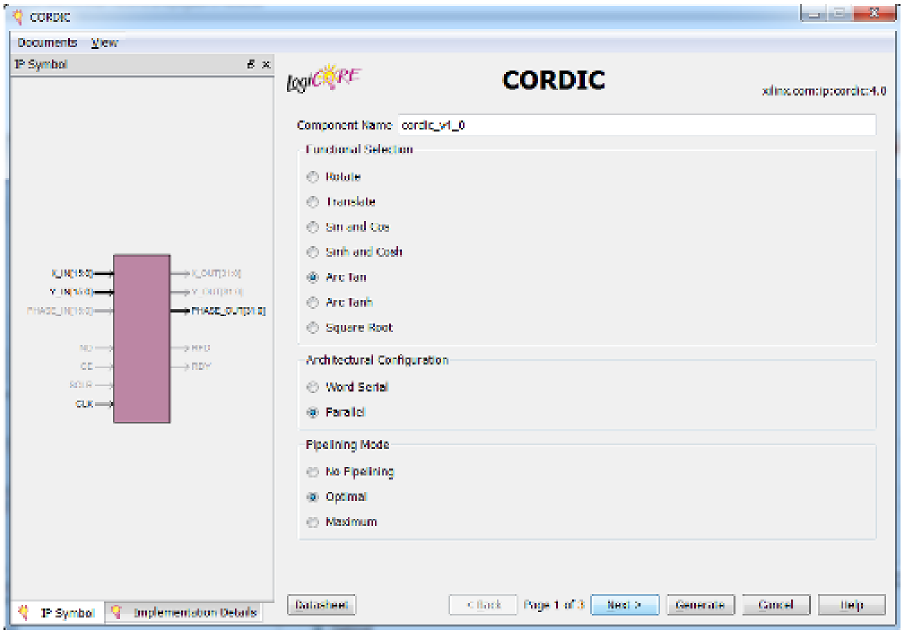
Component Name cordic_v4_0

Functional Selection
- ● Rotate
- ○ Translate
- ○ Sin and Cos
- ○ Sinh and Cosh
- ○ Arc Tan
- ○ Arc Tanh
- ○ Square Root

Architectural Configuration
- ○ Word Serial
- ● Parallel

Pipelining Mode
- ○ No Pipelining
- ○ Optimal
- ● Maximum

9. Apply the following settings:
   Page 1
   a. Functional Selection: Arc Tan
   b. Architectural Configuration: Parallel
   c. Pipelining Mode: Optimal



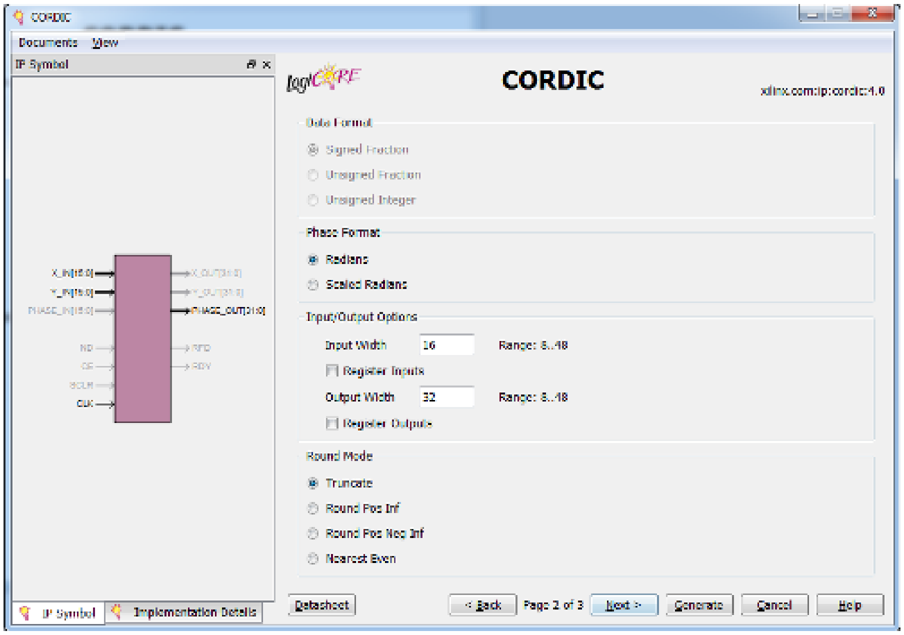
Component Name cordic_v4_0

Functional Selection
- ○ Rotate
- ○ Translate
- ○ Sin and Cos
- ○ Sinh and Cosh
- ● Arc Tan
- ○ Arc Tanh
- ○ Square Root

Architectural Configuration
- ○ Word Serial
- ● Parallel

Pipelining Mode
- ○ No Pipelining
- ● Optimal
- ○ Maximum

   Page 2
   d. Phase Format: Radians
   e. Input/Output Options: Input Width 16 , Output Width 32
   f. Round Mode: Truncate



CORDIC

Data Format

○ Signed Fraction

○ Unsigned Fraction

○ Unsigned Integer

Phase Format

● Radians

○ Scaled Radians

Input/Output Options

Input Width 16 Range: 8..48

□ Register Inputs

Output Width 32 Range: 8..48

□ Register Outputs

Round Mode

● Truncate

○ Round Pos Inf

○ Round Pos Neg Inf

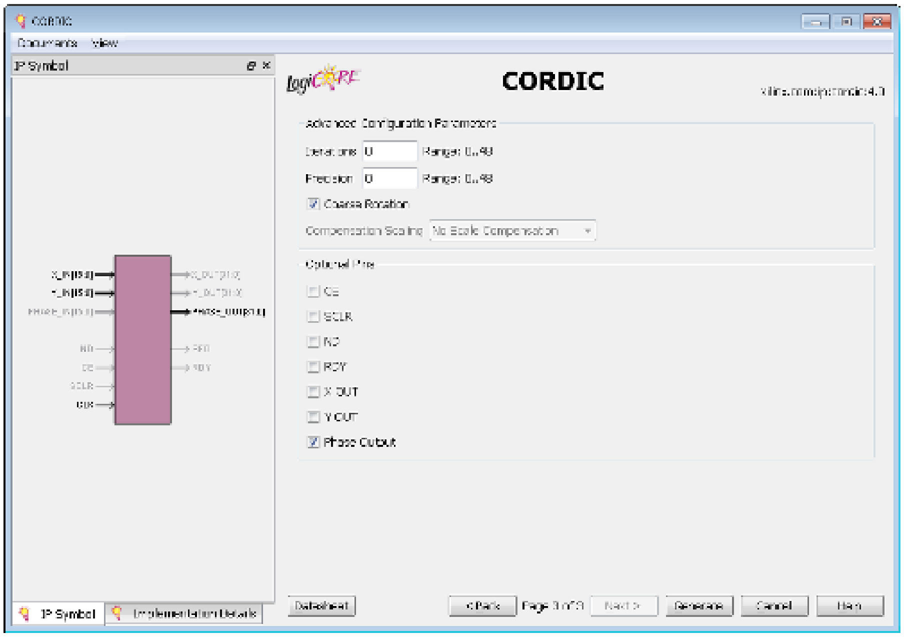
○ Nearest Even

Page 3

g. Iterations: 0 (Set automatically by Xilinx IP COre Generator)

h. Precision: 0 (Set automatically by Xilinx IP COre Generator)

i. Optional Pins: Phase Output is selected



CORDIC

Advanced Configuration Parameters

Iterations 0 Range: 0..48

Precision 0 Range: 0..48

☑ Coarse Rotation

Compensation Scaling No Scale Compensation

Optional Pins

□ CE

□ SCLR

□ ND

□ RDY

□ X OUT

□ Y OUT

☑ Phase Output

10. Click Generate. This may take a few minutes and a Readme cordic_v4_0 dialog will mark the completion.

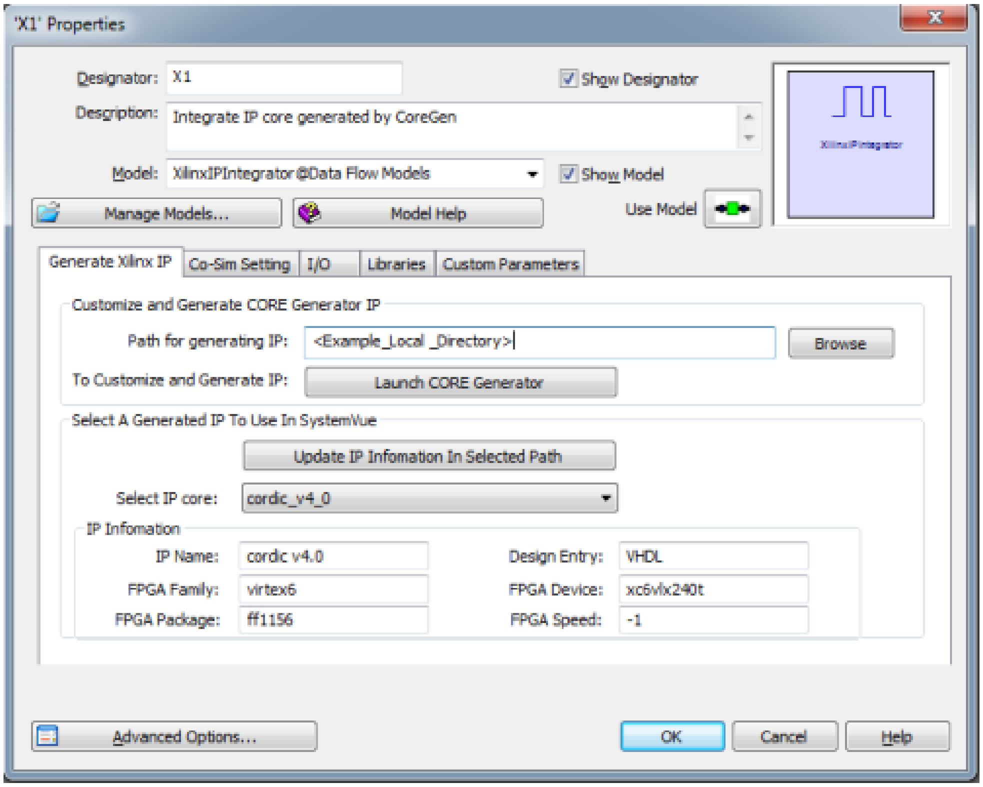
At this point, Xilinx IP Core will generate the following files in directory <Example_Local_Directory>\SystemVue_CoreGen_IP\

- XCO file ( cordic_v4_0.xco ): CORE Generator input file containing the parameters used to regenerate a core.

- Implementation Netlist ( cordic_v4_0.ngc ): Binary Xilinx implementation netlist files containing the information required to implement the module in a Xilinx (R) FPGA.
- Instantiation Templates ( cordic_v4_0.vho ): Template files containing code that can be used as a model for instantiating a CORE Generator module in a VHDL design.
- Simulation file ( cordic_v4_0.vhd ): This file is a VHDL wrapper file that is provided to support functional simulation.

Return back to XilinxIPIntegrator part's Properties dialog in SystemVue, and follow the steps below:

1. Click Update IP Information In Selected Path. This will update the list of IP cores and their associated information with the ones found in <Example_Local_Directory>. See the figure below:



'X1' Properties

Designator: X1

☑ Show Designator

Description: Integrate IP core generated by CoreGen

Model: XilinxIPIntegrator@Data Flow Models

☑ Show Model

Customize and Generate CORE Generator IP

Path for generating IP: <Example_Local _Directory>

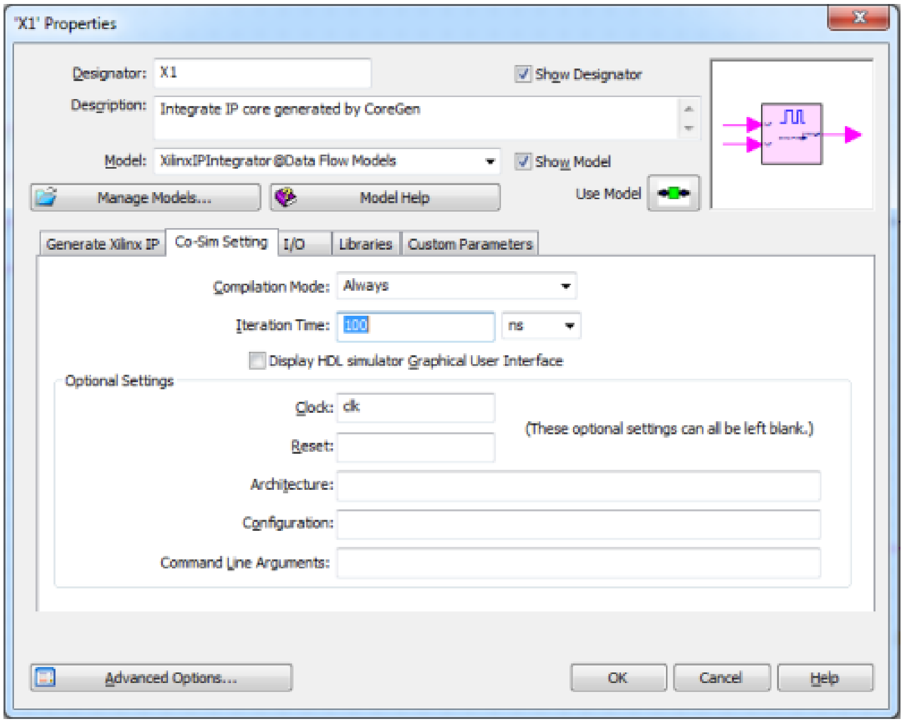
Select A Generated IP To Use In SystemVue

Select IP core: cordic_v4_0

IP Infomation

| | | | |
|---|---|---|---|
| IP Name: | cordic v4.0 | Design Entry: | VHDL |
| FPGA Family: | virtex6 | FPGA Device: | xc6vlx240t |
| FPGA Package: | ff1156 | FPGA Speed: | -1 |

2. Go to Co-Sim Settings page, and set
   a. Compilation to *Always*
   b. Clock port to *clk*



Designator: X1 ☑ Show Designator

Description: Integrate IP core generated by CoreGen

Model: XilinxIPIntegrator@Data Flow Models ☑ Show Model

Compilation Mode: Always

Iteration Time: 100 ns

☐ Display HDL simulator Graphical User Interface

Optional Settings

Clock: clk

Reset:

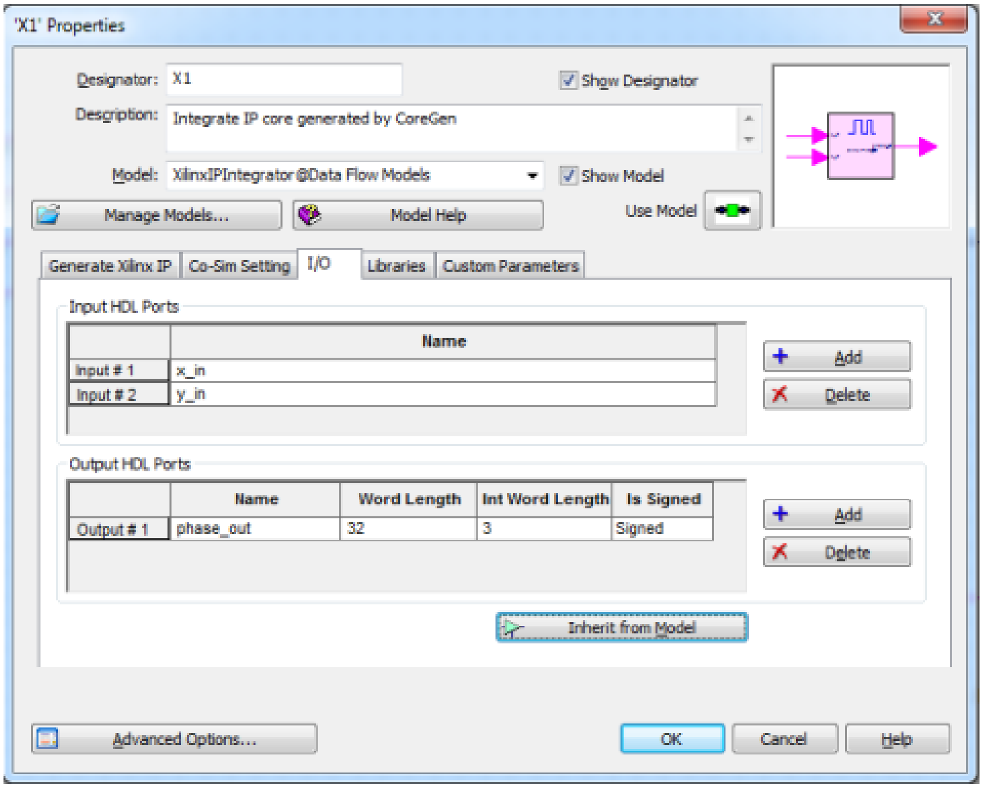
Architecture:

Configuration:

Command Line Arguments:

(These optional settings can all be left blank.)

3. Go to I/O page, click in Inherit from Model and select Top Level HDL. Change Integer Word Length of phase_output port from *32* to *3* and select *Signed*



Designator: X1 ☑ Show Designator

Description: Integrate IP core generated by CoreGen

Model: XilinxIPIntegrator@Data Flow Models ☑ Show Model

Input HDL Ports

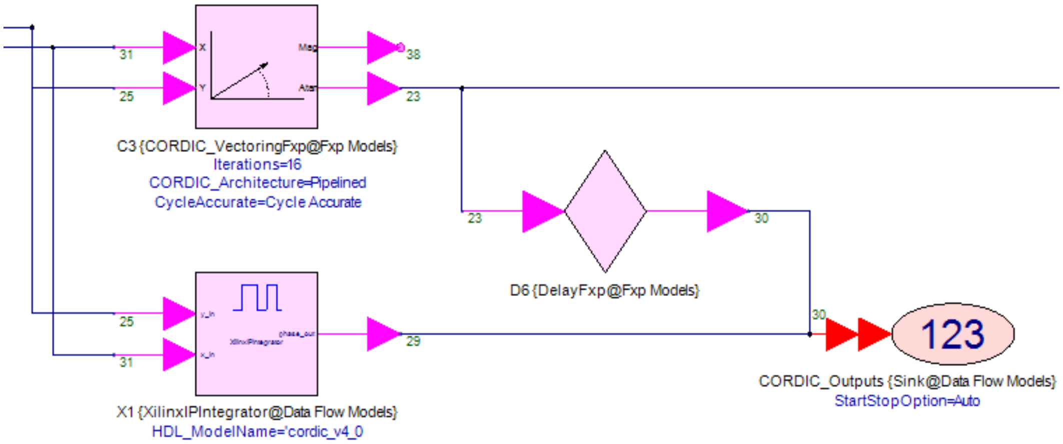
| | Name |
|---|---|
| Input # 1 | x_in |
| Input # 2 | y_in |

Output HDL Ports

| | Name | Word Length | Int Word Length | Is Signed |
|---|---|---|---|---|
| Output # 1 | phase_out | 32 | 3 | Signed |

4. Click OK to close the user interface and save the settings.

Place XilinxIPIntegrator beside Fixed Point CORDIC Vectoring part, compare the outputs of both parts as shown in the figure below:


31
25
X
Y
Mag
Atan
38
23
C3 {CORDIC_VectoringFxp@Fxp Models}
Iterations=16
CORDIC_Architecture=Pipelined
CycleAccurate=Cycle Accurate
23
30
D6 {DelayFxp@Fxp Models}
25
31
y_in
x_in
phase_out
XilinxIPIntegrator
29
X1 {XilinxIPIntegrator@Data Flow Models}
HDL_ModelName='cordic_v4_0
30
123
CORDIC_Outputs {Sink@Data Flow Models}
StartStopOption=Auto

Determine the difference in latencies between both parts and re-adjust the re-timing delays accordingly.

CAUTION According to the documentation of the **Fixed Point CORDIC Vectoring** part, the latency of this part is equal to the number of iterations + 7 samples. Hence, the original retiming-delay setting is 16+7=23.

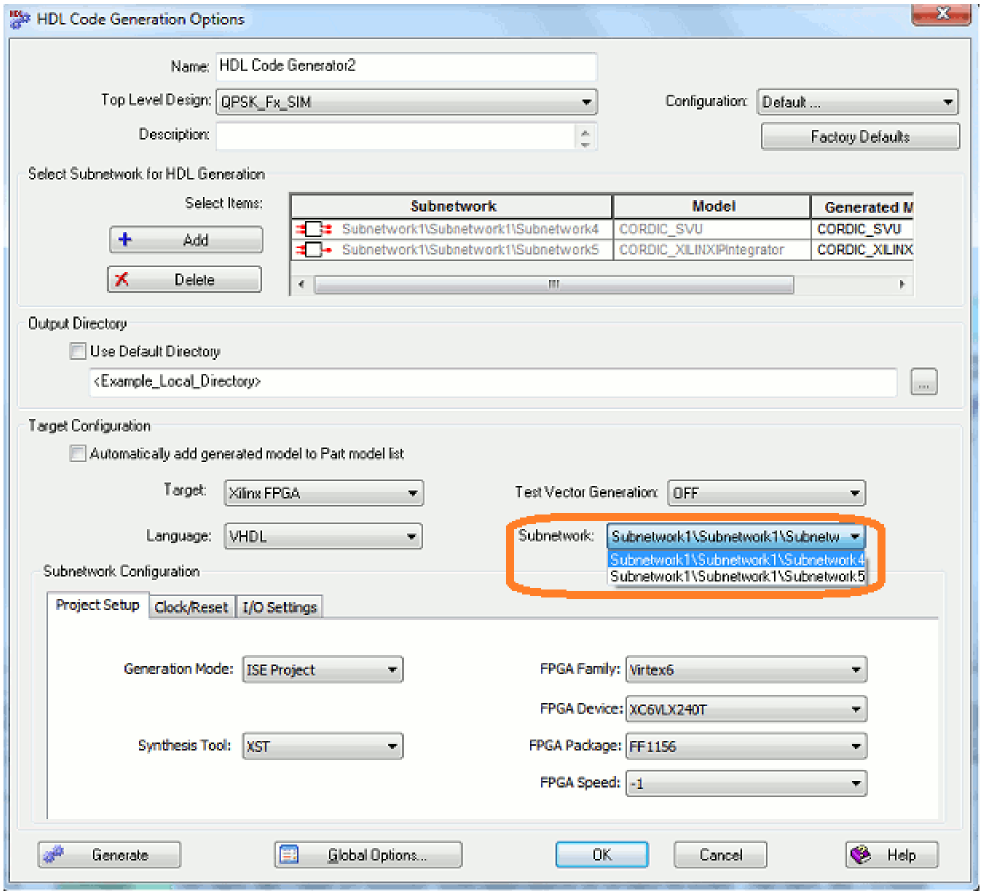
## HDL Code Generation of sub-systems with Xilinx IP Cores

1. Right click on Fixed Point CORDIC Vectoring part and select convert to subnetwork. Name it CORDIC_SVU.
2. Right-click on XilinxIPIntegrator part and select convert to subnetwork. Name it CORDIC_XILINXIPIntegrator.
3. Replace these two parts with their associated subnetworks,

   CAUTION When wiring the generated subnetwork, take into account that the order of pins in the generated subnetwork symbol can be different from the order of pins in the part itself.

4. To verify that the wiring is done correctly, run the simulation again and make sure that the results have not changed.
5. Add both subnetworks to the HDL code generator and configure the HDL code generation as shown in the figure below.

   CAUTION Select **Subnetwork:** field for each and configure it as highlighted. In particular, make sure for **CORDIC_XILINXIPIntegrator** subnetwork the FPGA target family configuration is consistent with what is in **XilinxIPIntegrator** part's **Properties** dialog earlier.



HDL Code Generation Options

Name: HDL Code Generator2

Top Level Design: QPSK_Fx_SIM

Configuration: Default ...

Description:

Select Subnetwork for HDL Generation

Select Items:

| Subnetwork | Model | Generated M |
|---|---|---|
| Subnetwork1\Subnetwork1\Subnetwork4 | CORDIC_SVU | CORDIC_SVU |
| Subnetwork1\Subnetwork1\Subnetwork5 | CORDIC_XILINXIPIntegrator | CORDIC_XILINX |

Output Directory

□ Use Default Directory

<Example_Local_Directory>

Target Configuration

□ Automatically add generated model to Part model list

Target: Xilinx FPGA

Test Vector Generation: OFF

Language: VHDL

Subnetwork: Subnetwork1\Subnetwork1\Subnetw
Subnetwork1\Subnetwork1\Subnetwork4
Subnetwork1\Subnetwork1\Subnetwork5

Subnetwork Configuration

Project Setup | Clock/Reset | I/O Settings

Generation Mode: ISE Project

FPGA Family: Virtex6

FPGA Device: XC6VLX240T

Synthesis Tool: XST

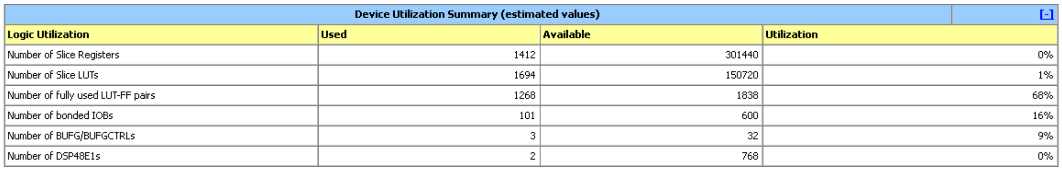
FPGA Package: FF1156

FPGA Speed: -1

6. Click Generate

At this point, two Xilinx ISE projects will be created and opened automatically for CORDIC_SVU and CORDIC_XilinxIPIntegrator in <Example_Local_Directory>\CORDIC_SVU_FPGA and <Example_Local_Directory>\CORDIC_XilinxIPIntegrator_FPGA folders respectively.

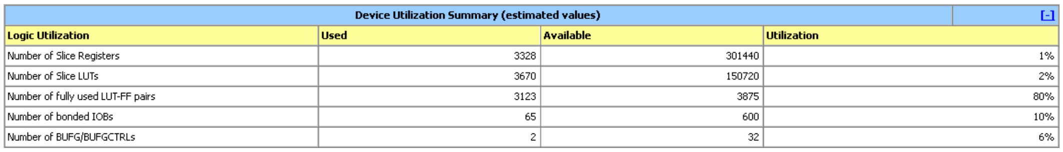
By synthesizing both projects (using Generate Post-Sythesis simulation model process), we can obtain the Maximum Frequency and the resource consumption for CORDIC_SVU and CORDIC_XilinxIPIntegrator.
CORDIC_SVU: Minimum period: 5.803ns (Maximum Frequency: 172.325MHz), and the resource consumption is as below



Device Utilization Summary (estimated values) [-]

| Logic Utilization | Used | Available | Utilization |
|---|---|---|---|
| Number of Slice Registers | 1412 | 301440 | 0% |
| Number of Slice LUTs | 1694 | 150720 | 1% |
| Number of fully used LUT-FF pairs | 1268 | 1838 | 68% |
| Number of bonded IOBs | 101 | 600 | 16% |
| Number of BUFG/BUFGCTRLs | 3 | 32 | 9% |
| Number of DSP48E1s | 2 | 768 | 0% |

**CORDIC_XilinxIPIntegrator**: Minimum period: 2.456ns (Maximum Frequency: 407.142 MHz), and the resource consumption is as below



Device Utilization Summary (estimated values) [-]

| Logic Utilization | Used | Available | Utilization |
|---|---|---|---|
| Number of Slice Registers | 3328 | 301440 | 1% |
| Number of Slice LUTs | 3670 | 150720 | 2% |
| Number of fully used LUT-FF pairs | 3123 | 3875 | 80% |
| Number of bonded IOBs | 65 | 600 | 10% |
| Number of BUFG/BUFGCTRLs | 2 | 32 | 6% |

## SystemVue M9703A and M9703B FPGA Design Flow

### Required Hardware and Software

Following is the list of required hardware and software.

#### Required Hardware

- M9703A AXIe 12-bit High-Speed Digitizer/Wideband Digital Receiver or M9703B AXIe 12-bit High-Speed Digitizer/Wideband Digital Receiver
  The M9703A or M9703B must be configured with –FDK option to enable its FPGA programming capability. For M9703A or M9703B hardware installation, see M9703A Startup Guide or M9703B Startup Guide.
- M9502A AXIe 2-Slot Chassis or
  M9505A AXIe 5-Slot Chassis or
  M9514A AXIe 14-Slot Chassis
  For M9502A and M9505A hardware installation, see M9502A and M9505A Startup Guide. For M9514A hardware installation, see M9514A Startup Guide .
- M9536A AXIe Embedded Controller or
  Connect a desktop with the AXIe chassis using M9048A PCIe Desktop Adapter and Y1202A PCIe cable or
  Connect a laptop with the AXIe chassis using M9045B PCIe ExpressCard Adapter and Y1200B PCIe cable

  When M9536A AXIe embedded controller is used, M9536A and M9703A must be inserted into the same AXIe chassis. For M9536A hardware installation, see M9536A Startup Guide

#### Required Software

- SystemVue 2016.08 or later version (W1462 license bundle is required)
- Keysight IO Libraries Suite: version 17.2 update 2 or later version
- For M9703A, Keysight MD1 High-Speed Digitizer Instrument Drivers and Soft Front Panel: 1.14.9 or later Windows version is required.
- For M9703B, Keysight MD2 High-Speed Digitizer Instrument Drivers: 1.12 or later Windows version is required.
- Xilinx ISE: version 14.7 or later

**NOTE** **SystemVue M9703A or M9703B FPGA design flow consists of FPGA design entry and software simulation, FPGA programming file generation and M9703A or M9703B instrument co-simulation.**

For FPGA design entry and software simulation, only SystemVue 2016.08 or later version is required. And SystemVue can be installed on any PC no matter whether it has the connection with AXIe chassis and M9703A or M9703B instrument or not. Keysight IO Libraries Suite, Keysight MD1 High-Speed Digitizer Instrument Driver, Keysight MD2 High-Speed Digitizer Instrument Driver and Xilinx ISE are not required for this step.

For FPGA programming file generation, SystemVue and Xilinx ISE are required. SystemVue and Xilinx ISE can be installed on any PC no matter whether it has connection with AXIe chassis and M9703A or M9703B instrument or not. Keysight IO Libraries Suite, Keysight MD1 High-Speed Digitizer Instrument Driver and Keysight MD2 High-Speed Digitizer Instrument Driver are not required for this step.

For **M9703A** instrument co-simulation, SystemVue, Keysight IO Libraries Suite and **Keysight MD1 High-Speed Digitizer Instrument Driver** are required. They must be installed on M9536A AXIe embedded controller or a PC that has PCIe connection with the AXIe chassis. Xilinx ISE is not required for this step.

For **M9703B** instrument co-simulation, SystemVue, Keysight IO Libraries Suite and **Keysight MD2 High-Speed Digitizer Instrument Driver** are required. They must be installed on M9536A AXIe embedded controller or a PC that has PCIe connection with the AXIe chassis. Xilinx ISE is not required for this step.

So you can do FPGA design entry, software simulation and FPGA programming file generation on any PC that may have no connection with AXIe chassis and M9703A or M9703B instrument. After the FPGA programming file is generated, you can deploy the generated FPGA programming file to run M9703A or M9703B instrument co-simulation on M9536A AXIe embedded controller or a PC with PCIe connection with the AXIe chassis.

Please note that M9703A and M9703B have the same hardware. The difference is that the driver for **M9703A is MD1 driver**, while the driver for **M9703B** is the **MD2 driver.**

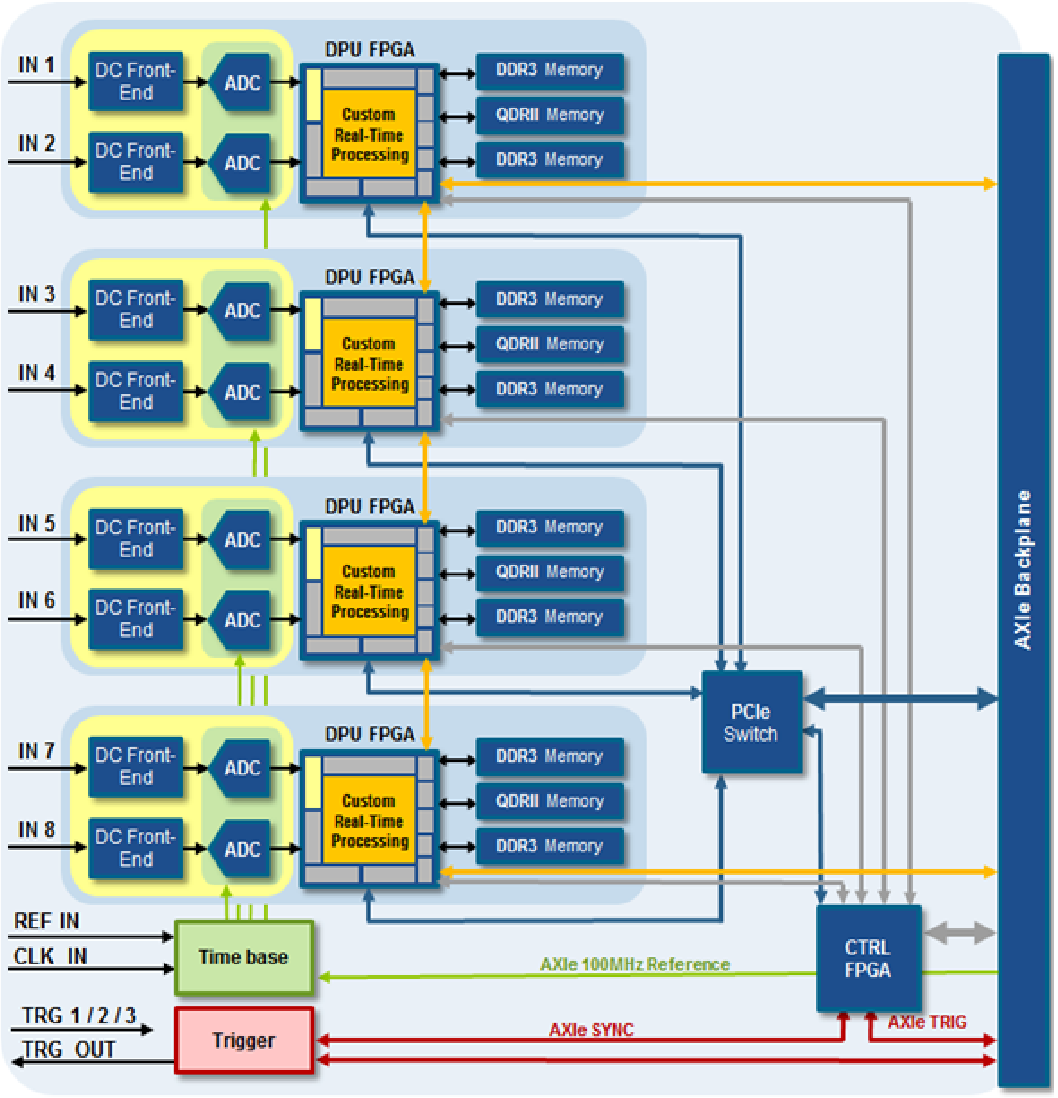
**In this document, we use M9703 to represent both M9703A and M9703B.**

## Overview of SystemVue M9703 FDK Design Flow

### Overview of M9703 High-Speed Digitizer

Based on the AXIe standard, the M9703 is an 8-channel, 12-bit wideband digital receiver/digitizer, able to capture signals from DC up to 2 GHz at 1.6GS/s. A channel interleaving capability allows waveform acquisition at up to 3.2 GS/s with exceptional measurement accuracy.

M9703 Hardware Diagram



**IN1~IN8** are 8 input ports from M9703 front panel. They go through **DC Front-End** that is electable analog low pass filter (pass band frequency is 600MHz). Then the analog input signals are fed into respective **ADC**s to convert to digital signals.
The digitizer architecture could be simply understood by the following data stream scheme: For each analog channel, a front-end electronics converts the customer analog signal into a digital stream of data (ADC). These digital streams are captured by a processing FPGA (further called **DPU FPGA**). The custom real-time processing usually provides a data reduction scheme to only store processed results into a temporary memory buffer. Then the host application retrieves these processed data through the PCIe data and control bus whose sustained data bandwidth is significantly lower than the raw data bandwidth.

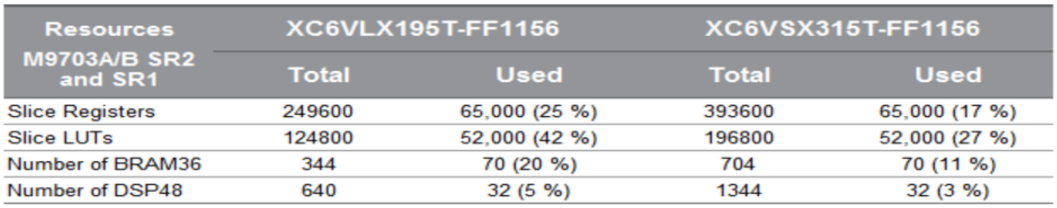
There are four DPU FPGA on an M9703.

- IN1 and IN2 are inputted into DPU FPGA0.
- IN3 and IN4 are inputted into DPU FPGA1.

- IN5 and IN6 are inputted into DPU FPGA2.
- IN7 and IN8 are inputted into DPU FPGA3.

The DPU FPGAs have two options, -LX2 and -SX3. For -LX2 option, the 4 DPU FPGAs are all Xilinx XC6VLX195T -FF1156 -2 FPGA. For -SX3 option, the 4 DPU FPGAs are all Xilinx XC6VSX315T -FF1156 -2 FPGA.

Each DPU FPGA opens a partial area for custom real-time processing. Below table shows the FPGA resources are used for M9703 infrastructure, the remaining FPGA resources can be used for custom real-time processing.



| Resources<br>M9703A/B SR2 and SR1 | XC6VLX195T-FF1156 | | XC6VSX315T-FF1156 | |
|---|---|---|---|---|
| | Total | Used | Total | Used |
| Slice Registers | 249600 | 65,000 (25 %) | 393600 | 65,000 (17 %) |
| Slice LUTs | 124800 | 52,000 (42 %) | 196800 | 52,000 (27 %) |
| Number of BRAM36 | 344 | 70 (20 %) | 704 | 70 (11 %) |
| Number of DSP48 | 640 | 32 (5 %) | 1344 | 32 (3 %) |

And each DPU FPGA has the following physical interface:

- ADC parallel data streams input
- Connectivity with two DDR3 memory and one QDRII memory
- Inter FPGAs data stream connectivity
- PCIe connectivity with backplane via PCIe switch
- Control signals from CTRL FPGA

In SystemVue M9703 FPGA design flow, these physical interfaces are transparent to users. SystemVue will provide some algorithm-level interfaces, such as OutPort, Register, and BlockRegister so that users can use these interfaces easily without caring about the format and protocol of the physical interfaces.
Each DPU FPGA is connected with two DDR3 SDRAM memories. The DDR3 SDRAM memories can be used to buffer the real-time processed data for PCIe transfer with embedded controller or PC. The memory size of each DDR3 SDRAM memory is indicated by M9703 option:
Option
-M10: The size of each DDR3 SDRAM is 128M Byte. There are 8 DDR3 SDRAM for total four DPU FPGA, so the total memory size of one M9703 is 1G Byte.
-M40: The size of each DDR3 SDRAM is 512M Byte. There are 8 DDR3 SDRAM for total four DPU FPGA, so the total memory size of one M9703 is 4G Byte.
-M16: The size of each DDR3 SDRAM is 2G Byte. There are 8 DDR3 SDRAM for total four DPU FPGA, so the total memory size of one M9703 is 16G Byte.

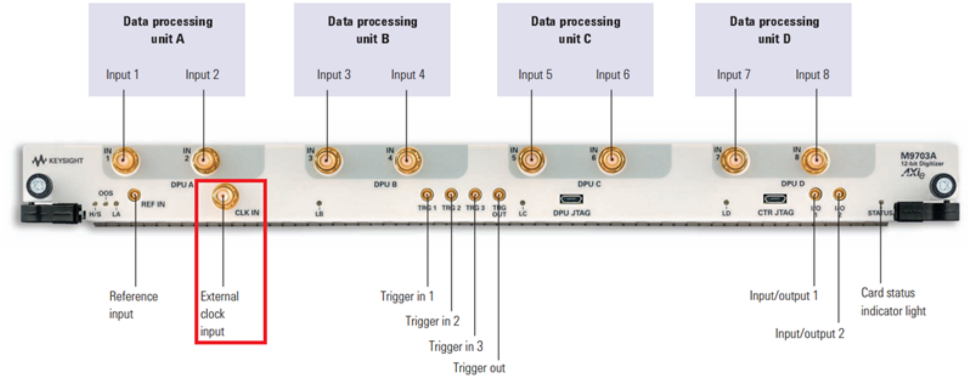
### M9703 DPU FPGAs Clock

All custom real-time processing are in the same system clock domain. The system clock is synchronized with DPU FPGA input clock source. M9703 provides four kinds of clock sources for its DPU FPGAs:

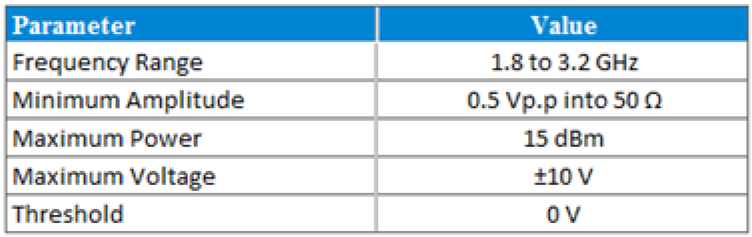
- Internal clock
  The internal clock is generated from M9703 module hardware.

- External clock
  The instruments external clock input connector (as shown in below figure) is selected as the source.

Front panel connectors



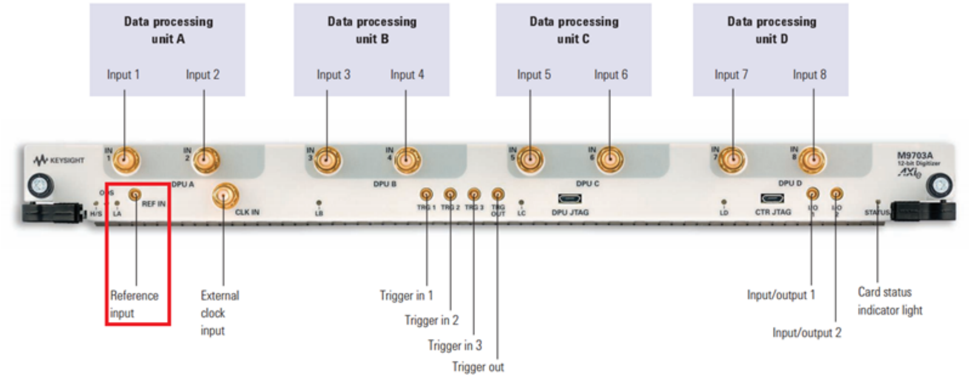
The External Clock may be used to vary the sampling rate of the digitizer, it must be continuously present if selected for the digitizer to operate correctly. The input is AC coupled. The requirement of external clock signal is:



| Parameter | Value |
|---|---|
| Frequency Range | 1.8 to 3.2 GHz |
| Minimum Amplitude | 0.5 Vp.p into 50 Ω |
| Maximum Power | 15 dBm |
| Maximum Voltage | ±10 V |
| Threshold | 0 V |

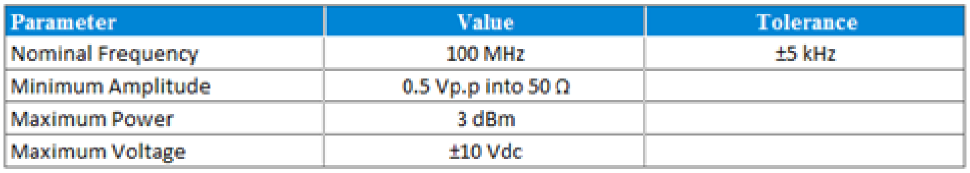
- External reference clock
  The instruments external reference clock input connector (as shown in below figure) is selected as the source.

Front panel connectors



For applications that require greater timing precision and long-term stability than is obtainable from the internal clock, a 100 MHz Reference signal can be used. The External Reference is nominally at 100 MHz. However, frequencies in a range will be accepted.

If your input is not at exactly 100 MHz, you must remember to compensate for the difference in your application since the digitizer and the driver has no way to know about such deviations.



| Parameter | Value | Tolerance |
|---|---|---|
| Nominal Frequency | 100 MHz | ±5 kHz |
| Minimum Amplitude | 0.5 Vp.p into 50 Ω | |
| Maximum Power | 3 dBm | |
| Maximum Voltage | ±10 Vdc | |

- AXIe reference clock
  This 100 MHz signal is provided via the AXIe backplane to the M9703 digitizer and may be selected as a reference clock.
  This AXIe Reference may also be optionally locked to an external 10 MHz input applied to the 'CLOCK IN' connector of the AXIe chassis. To implement this, the 10 MHz signal must be present before the chassis is powered-up, and when detected the chassis will automatically lock the AXIe Reference to this signal.

All custom real-time processing on DPU FPGA run on DPU FPGA system clock. When clock source is selected as Internal Clock, External Reference Clock or AXIe Reference clock, the DPU FPGA system clock frequency is fixed. When M9703 is configured as 1GS/s option (-SR1), the DPU FPGA system clock is 125MHz. All custom real-time processing and ADC captured data in DPU FPGAs run on the 125MHz DPU FPGA system clock. When M9703 is configured as 1.6GS/s option (-SR2), the DPU FPGA system clock is 200MHz. All custom real-time processing and ADC captured data in DPU FPGAs run on the 200MHz system clock.

When clock source is selected as External Clock, the DPU FPGA system clock frequency varies as the frequency of External Clock input. The DPU FPGA system clock frequency is equal to one sixteenth of the frequency of External Clock input. When M9703 is configured as 1GS/s option (-SR1), the frequency of External Clock input is in range 1.8GHz ~ 2.0GHz, so the DPU FPGA system clock frequency is in range 112.5MHz ~ 125MHz. When M9703 is configured as 1.6GS/s option (-SR2), the frequency of External Clock input is in range 1.8GHz ~ 3.2GHz, so the DPU FPGA system clock frequency is in range 112.5MHz ~ 200MHz.

DPU FPGA system clock frequency:

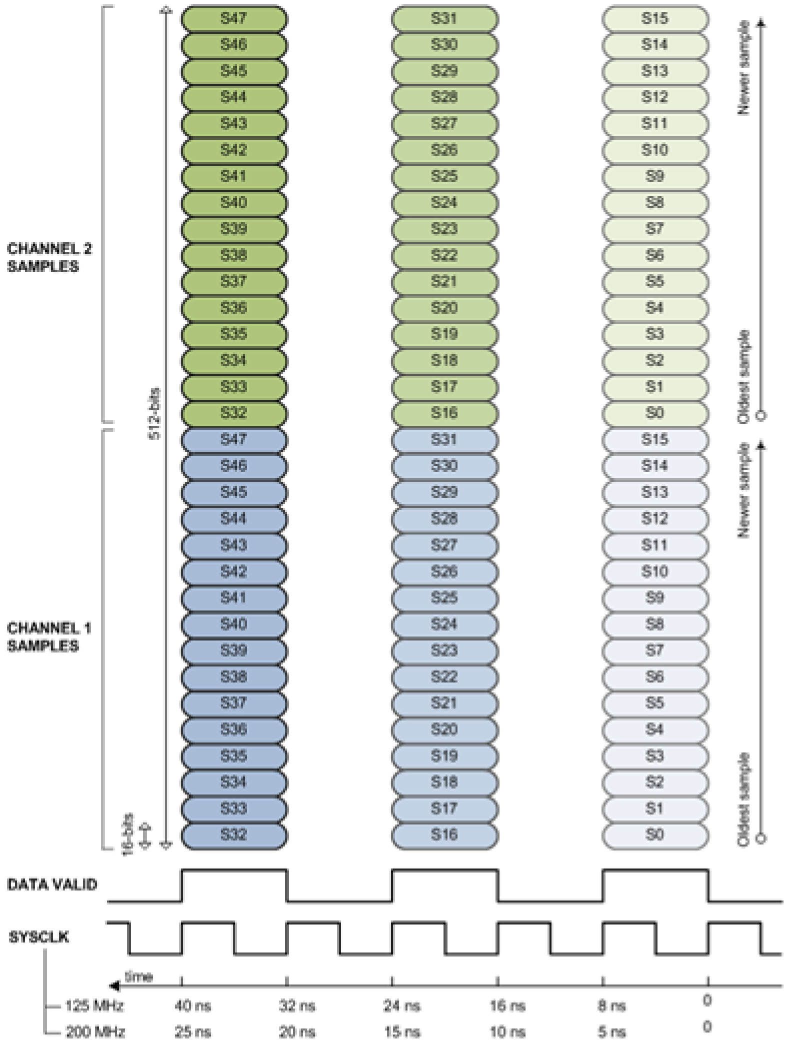
| | 1GS/s option (-SR1) | 1.6GS/s option (-SR2) |
|---|---|---|
| Clock Source:<br>**Internal Clock** Or<br>**External Reference Clock** Or<br>**AXIe Reference Clock** | 125MHz | 200MHz |
| Clock Source:<br>**External Clock** | External Clock Frequency / 16<br>External Clock Range:<br>1.8GHz ~ 2.0GHz | External Clock Frequency / 16<br>External Clock Range:<br>1.8GHz ~ 3.2GHz |

### ADC Parallel Input Streams of M9703 DPU FPGAs

For ADC input to DPU FPGA, the incoming ADC data streams are de-multiplexed into multiple parallel data streams at lower data rate and then perform some pre-correction of the sampling defaults (e.g. linearity correction, interleave mismatches correction). So for each sample, the 12-bit ADC data are extended to 16-bit after the pre-correction.

In DPU FPGA, the parallel ADC input streams are all sampled based on DPU FPGA system clock.

For normal sample mode (1.0GS/s for SR1 option and 1.6GS/s for SR2 option), each DPU FPGA receives two channels ADC input streams. Below figure shows how the ADC input streams are inputted into the custom real-time processing part.

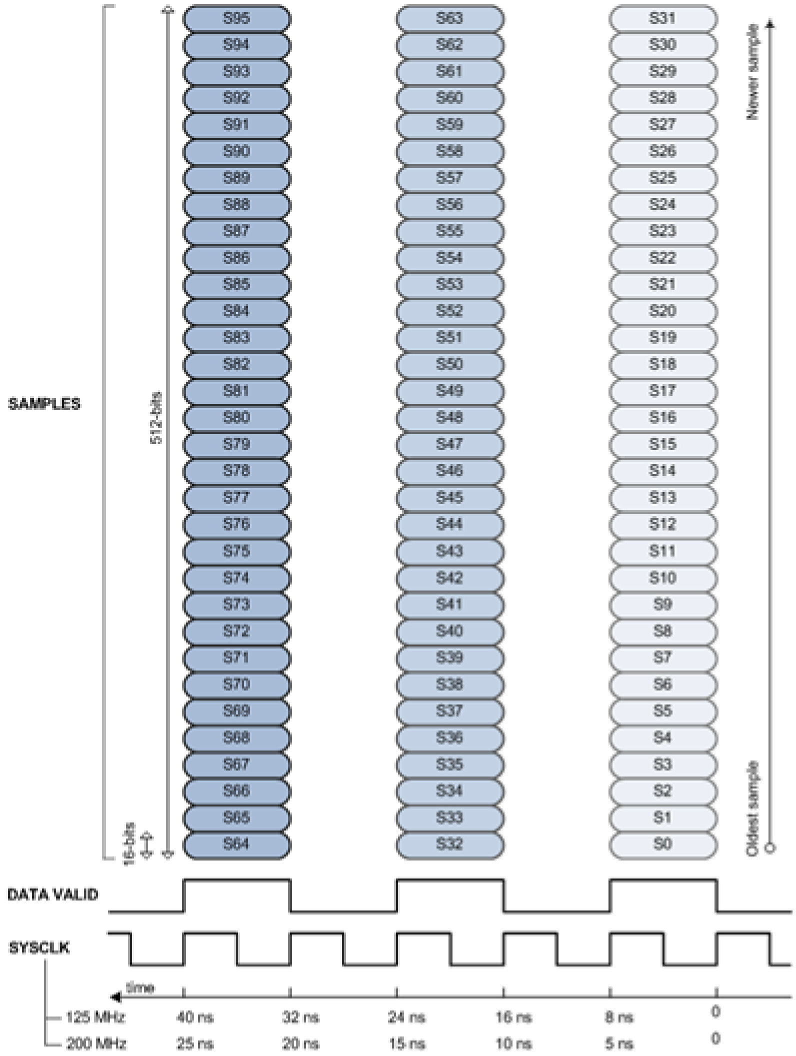

CHANNEL 2 SAMPLES
CHANNEL 1 SAMPLES
512-bits
16-bits
S47 S46 S45 S44 S43 S42 S41 S40 S39 S38 S37 S36 S35 S34 S33 S32
S31 S30 S29 S28 S27 S26 S25 S24 S23 S22 S21 S20 S19 S18 S17 S16
S15 S14 S13 S12 S11 S10 S9 S8 S7 S6 S5 S4 S3 S2 S1 S0
Newer sample
Oldest sample
DATA VALID
SYSCLK
time
125 MHz
40 ns
32 ns
24 ns
16 ns
8 ns
0
200 MHz
25 ns
20 ns
15 ns
10 ns
5 ns
0

Each ADC channel has 16 parallel streams. Every two system clock cycles, there are valid ADC samples. The DATA VALID signal in above figure indicates when valid ADC samples arrive.

For 1.0GS/s SR1 option, its DPU FPGA system clock is 125MHz and valid samples arrive every two system clock cycles, so the total sample rate is: 16 parallel streams x 125MS / 2 = 1.0GS/s. For 1.6GS/s SR2 option, its DPU FPGA system clock is 200MHz and valid samples arrive every two system clock cycles, so the total sample rate is: 16 parallel streams x 200MS / 2 = 1.6GS/s.
Because each sample has 16-bit, every time the valid samples arrive, there are

(16 parallel samples/channel) x (16-bits/sample) x 2 channel = 512 bit

For interleaved sample mode (2.0GS/s for SR1 option and 3.2GS/s for SR2 option), each DPU FPGA receives one channel ADC input streams. Below figure shows how the ADC input streams are inputted into the custom real-time processing part.


SAMPLES
512-bits
16-bits
S95 S94 S93 S92 S91 S90 S89 S88 S87 S86 S85 S84 S83 S82 S81 S80 S79 S78 S77 S76 S75 S74 S73 S72 S71 S70 S69 S68 S67 S66 S65 S64
S63 S62 S61 S60 S59 S58 S57 S56 S55 S54 S53 S52 S51 S50 S49 S48 S47 S46 S45 S44 S43 S42 S41 S40 S39 S38 S37 S36 S35 S34 S33 S32
S31 S30 S29 S28 S27 S26 S25 S24 S23 S22 S21 S20 S19 S18 S17 S16 S15 S14 S13 S12 S11 S10 S9 S8 S7 S6 S5 S4 S3 S2 S1 S0
Newer sample
Oldest sample
DATA VALID
SYSCLK
time
125 MHz
200 MHz
40 ns
25 ns
32 ns
20 ns
24 ns
15 ns
16 ns
10 ns
8 ns
5 ns
0
0

Each ADC channel has 32 parallel streams. Every two system clock cycles, there are valid ADC samples. The DATA VALID signal in above figure indicates when valid ADC samples arrive.

For 2.0GS/s SR1 option, its DPU FPGA system clock is 125MHz and valid samples arrive every two system clock cycles, so the total sample rate is: 32 parallel streams x 125MS / 2 = 2.0GS/s. For 3.2GS/s SR2 option, its DPU FPGA system clock is 200MHz and valid samples arrive every two system clock cycles, so the total sample rate is: 32 parallel streams x 200MS / 2 = 3.2GS/s.
Because each sample has 16-bit, every time the valid samples arrive, there are

(32 parallel samples/channel) x (16-bits/sample) x 1 channel = 512 bit

### Overview of M9703 FPGA Design Flow

The SystemVue M9703 FPGA design flow consists of three steps: design entry and software simulation, M9703 FPGA programming file auto generation and M9703 instrument co-simulation with SystemVue.

- Design entry and software simulation
  SystemVue provides a hierarchical subnet template to ask users to configure M9703 four FPGAs and design FPGAs in a model-based environment. In addition, SystemVue provides some software peripheral models to mimic the hardware behaviors of M9703 FPGA interface, so users can run a pure software simulation to debug and verify their FPGA design.
- M9703 FPGA programming file auto generation
  When the FPGA design in SystemVue schematic is verified by software simulation and is ready for generating FPGA programming file, SystemVue provides a "one-button-push" automatic flow to launch Xilinx ISE to generate final M9703 FPGA programming files in background.
- M9703 instrument co-simulation with SystemVue
  After the M9703 FPGA programming files are generated, SystemVue provides a M9703 instrument co-simulation model to connect SystemVue with M9703 hardware. This model can download M9703 FPGA programming file to DPU FPGAs, configure the user-defined FPGA registers and capture data output from M9703 DPU FPGA to SystemVue. Users can verify the custom real-time processing on real hardware in this step.

NOTE

After M9703 instrument co-simulation verification, users can use AgMD1 IVI-C or AgMD1 IVI-COM driver to create their own program to deploy the user-defined DPU FPGA programming file and control M9703 instrument without launching SystemVue.

AgMD1 IVI-C and AgMD1 IVI-COM driver are installed automatically when you install Keysight MD1 High-Speed Digitizer Instrument Drivers and Soft Front Panel.

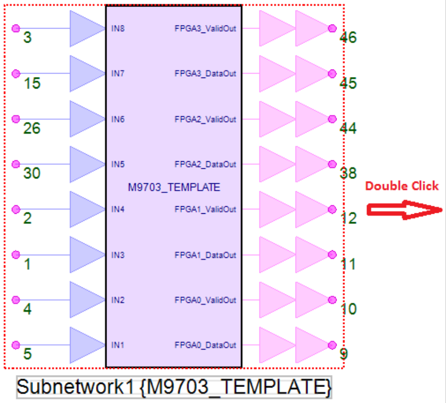
For details of programming using AgMD1 IVI-C and AgMD1 IVI-COM, please refer to the document of **IVI Driver Reference** . You will get this document after installing Keysight MD1 High-Speed Digitizer Instrument Drivers and Soft Front Panel.

## M9703 Design Template

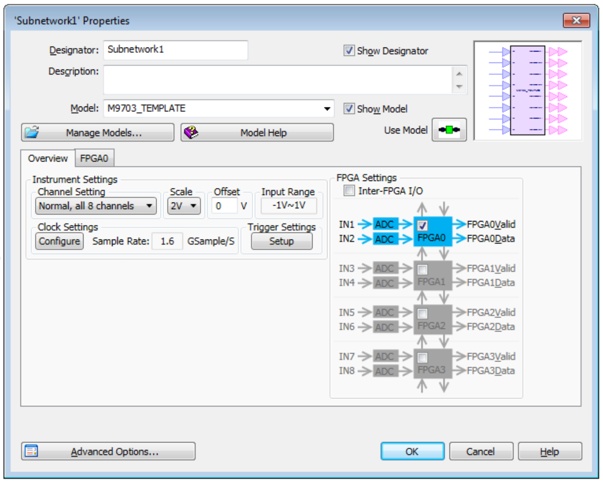
M9703 Design Template is a set of hierarchical subnets to model M9703's hardware architecture. It mimics and simplifies the behavior of M9703 DPU FPGAs' interface, so that users can easily configure M9703 DPU FPGAs and enter model-based FPGA design under M9703's hardware architecture. Then users can do software simulation with M9703 Design Template to debug and verify whether their real-time processing part works well under M9703's hardware architecture before generating FPGA programming file.

When you drag-and-drop the top level subnet of M9703 Design Template from workspace tree to a schematic, you will get a M9703 Design Template model in the schematic. This model has a GUI to help you configure the M9703 interfaces and working mode easily.

**M9703 Design Template Model**



**M9703 Design Template Model GUI**



'Subnetwork1' Properties

Designator: Subnetwork1 ☑ Show Designator

Description:

Model: M9703_TEMPLATE ☑ Show Model

Overview | FPGA0

Instrument Settings

Channel Setting: Normal, all 8 channels

Scale: 2V

Offset: 0 V

Input Range: -1V~1V

Clock Settings: Sample Rate: 1.6 GSample/S

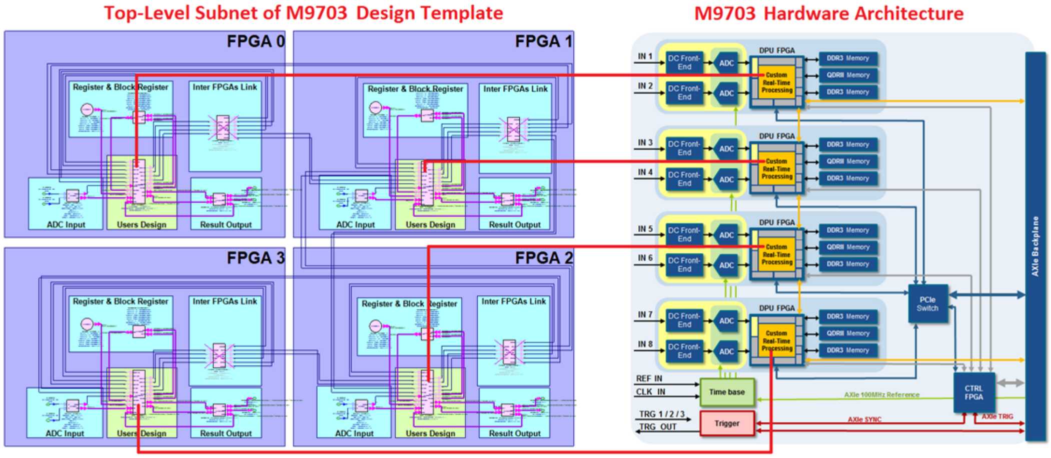
FPGA Settings

☐ Inter-FPGA I/O

☑ FPGA0 ☐ FPGA1 ☐ FPGA2 ☐ FPGA3

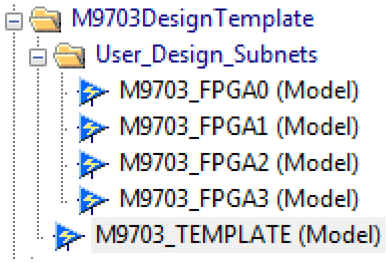
Below figure shows the top-level subnet of M9703 Design Template. The four DPU FPGAs of M9703 are mimicked in the top-level subnet. For each mimicked DPU FPGA, there is a green area Users Design. This area is mapped to Custom real-time processing part on the corresponding DPU FPGA hardware. Users can enter their model-based FPGA design in the subnet of Users Design area.

M9703 DPU FPGAs' physical interfaces, such as ADC Input, Inter-FPGA link, DDR3 Memory link and PCIe link, are simplified in the M9703 Design Template subnets. So users just need to connect their own real-time processing part with these simplified interfaces in M9703 Design Template, instead of being familiar with the sophisticated physical interfaces of M9703 hardware. It will save a lot of development time. Then when you generate FPGA programming file, SystemVue will provide an automatic flow to connect users' real-time processing part design with the physical interfaces well and generate the final FPGA programming file.



## M9703 Design Template Hierarchy

You can find the subnets architecture of M9703 Design Template in SystemVue workspace tree from example workspace " *<SystemVue installation dir>* \Examples\Hardware Design\M9703_FDK\M9703_Design_Template\ M9703_FDK_Design_Template.wsv", as shown below:

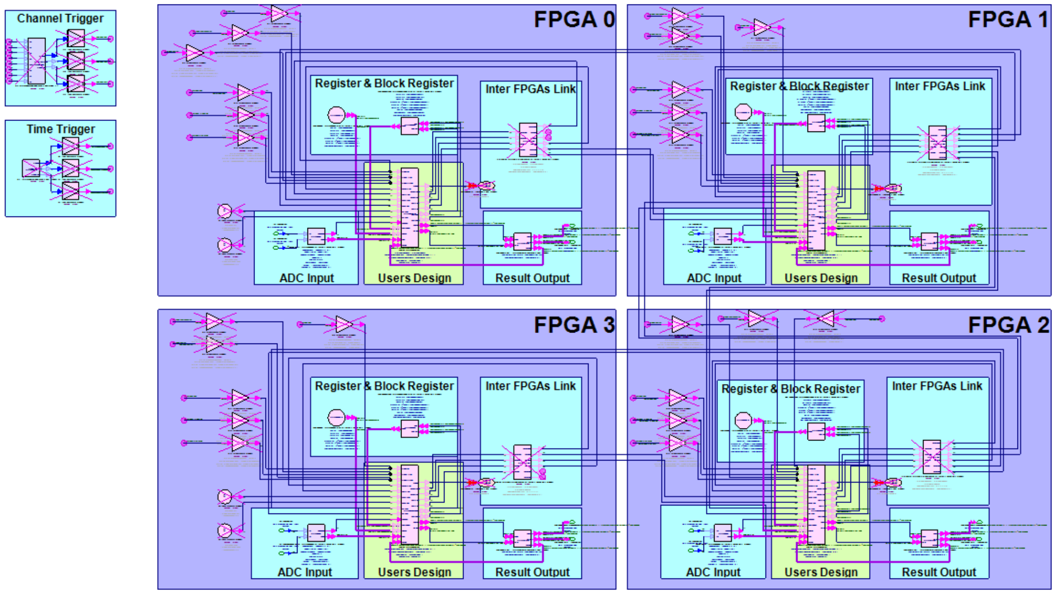


M9703DesignTemplate
User_Design_Subnets
M9703_FPGA0 (Model)
M9703_FPGA1 (Model)
M9703_FPGA2 (Model)
M9703_FPGA3 (Model)
M9703_TEMPLATE (Model)

## Top-level subnet: M9703_TEMPLATE

This subnet is the top level subnet of the M9703 Design Template. You can drag-and-drop it from the workspace tree to a schematic to instantiate a M9703 Design Template model in a schematic, and double click the model to open M9703 Design Template GUI to configure the M9703 DPU FPGAs interfaces. All subnets and their parameters are all controlled by the GUI, so you can use the GUI to configure the M9703 DPU FPGA interface and working mode and all subnets under M9703 Design Template can be configured well automatically according to your input in the GUI.

If you double click the M9703_TEMPLATE subnet in workspace tree, you can open the subnet to view the implementation of the subnet. All things in this subnet implementation have be pre-configured well and controlled by M9703 Design Template GUI, so users need do nothing in this subnet.

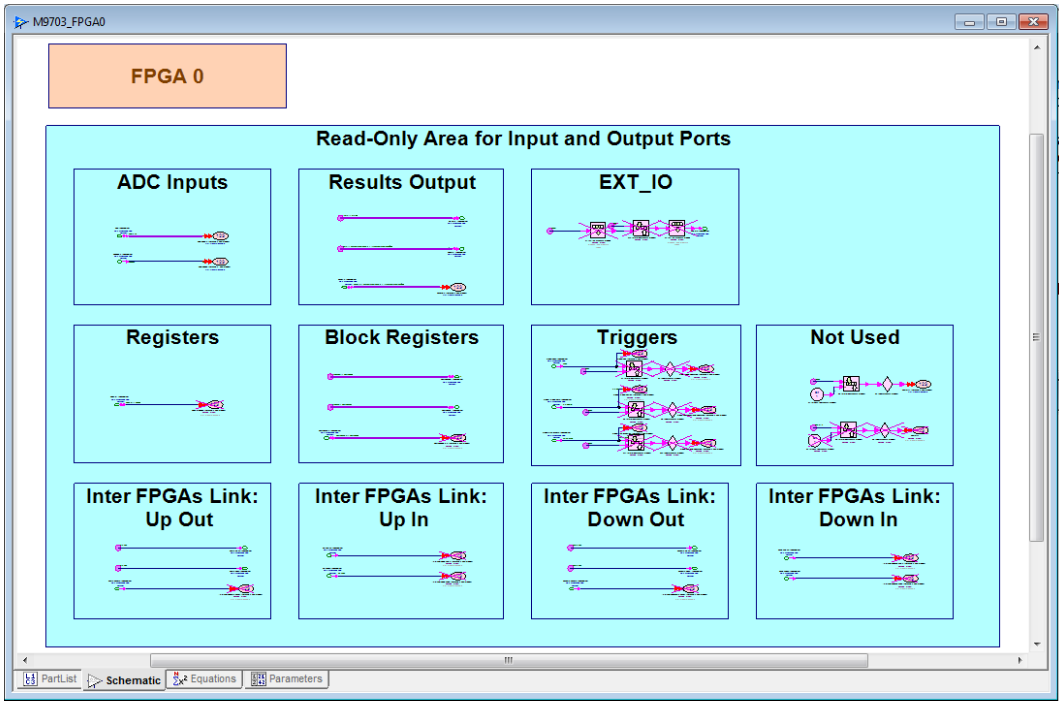


In the M9703_TEMPLATE subnet, there is a subnet for custom real-time processing FPGA design entry in the green area Users Design of each DPU FPGA. The subnets and models in the other blue areas mimic M9703 DPU FPGAs hardware architecture and provide the simplified interfaces for the custom real-time processing part.

In workspace tree of M9703 Design Template, the subnets in folder User_Design_Subnets – M9703_FPGA0, M9703_FPGA1, M9703_FPGA2 and M9703_FPGA3, are in green Users Design area in the top level M9703_TEMPLATE subnet. You can enter your model-based FPGA design in these subnets for the corresponding DPU FPGA.

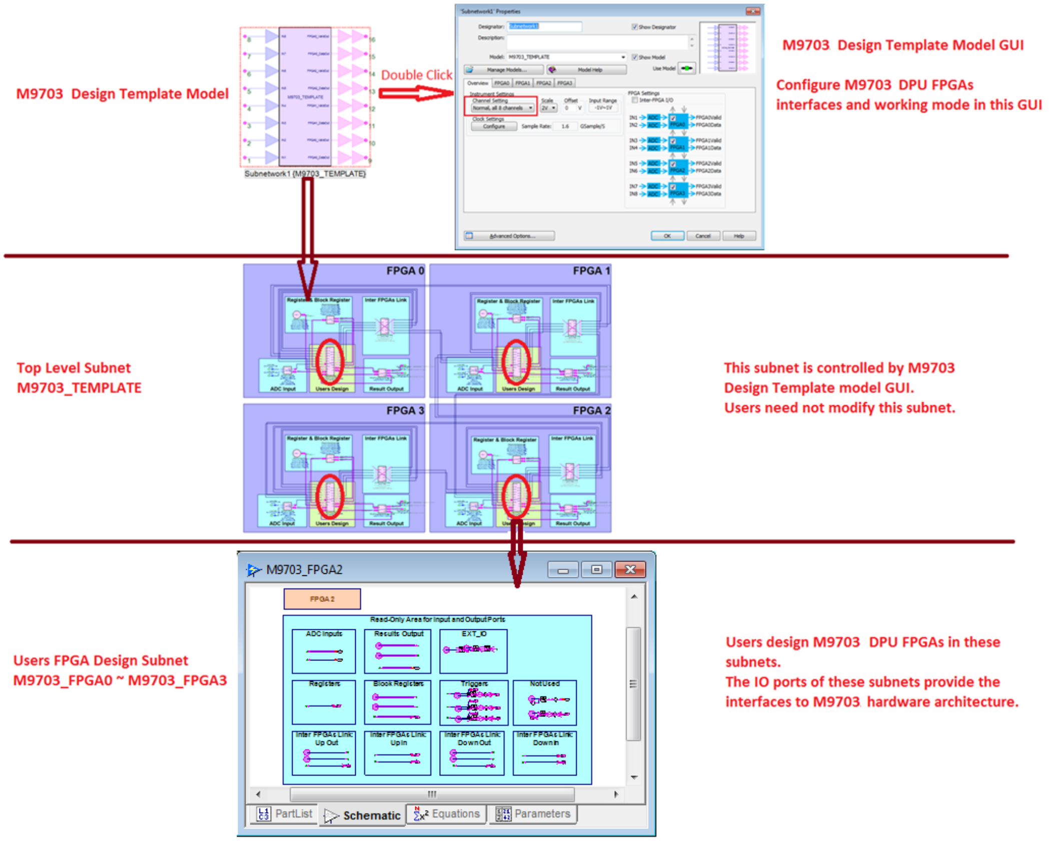
### Users FPGA Design subnets: M9703_FPGA0 ~ M9703_FPGA3

You can double click subnet M9703_FPGA*x* in workspace tree to look into the implementation of the subnet, as shown below.



When you enable FPGAx in M9703 Design Template model GUI, the corresponding subnet M9703_FPGAx is enabled. Above figure shows a blank M9703_FPGA0 subnet and its interface models have been placed in this subnet in advance. These interface models are configured automatically according to your input in M9703 Design Template model GUI. Please don't modify anything in the blue "Read-Only" area in order to keep these pre-configured IO ports working correctly. Users can create their own model-based fixed-point design connecting with the defined interface. See Users FPGA Design Interfaces in M9703 Design Template to get details about how to connect model-based user application design with the defined FPGA interface.

In summary, the hierarchy of M9703 design template is shown in below figure.



M9703 Design Template Model

M9703 Design Template Model GUI

Configure M9703 DPU FPGAs interfaces and working mode in this GUI

Top Level Subnet M9703_TEMPLATE

This subnet is controlled by M9703 Design Template model GUI. Users need not modify this subnet.

Users FPGA Design Subnet M9703_FPGA0 ~ M9703_FPGA3

Users design M9703 DPU FPGAs in these subnets. The IO ports of these subnets provide the interfaces to M9703 hardware architecture.

1. The top level is the M9703 Design Template model that you drag-and-drop from the workspace tree to schematic. You can double click this model in a schematic to open its GUI. You can configure M9703 interfaces and working mode in the GUI.
2. Look into M9703 Design Template model (subnet M9703_TEMPLATE), and you will go a lower level to view the implementation of the M9703 Design Template. You can find the peripheral subnets and user FPGA design subnet in this level. All things in this level are controlled by your input in M9703 Design Template model GUI automatically. Please don't modify anything in this level.
3. Look into one of user FPGA design subnets, M9703_FPGA0 ~ M9703_FPGA3, and you will go a lower level to create your own model-based design that you want to implement on the corresponding M9703 DPU FPGA in this level. The I/O ports have been placed in these subnets in advance and are also controlled by your input in M9703 Design Template model GUI automatically. So please don't modify anything in blue "Read-Only" area in these subnets.

Finally, you can follow the below steps to design M9703 DPU FPGAs using M9703 Design Template:

1. Find M9703 Design Template from SystemVue example workspace: *<your SystemVue installation dir>*\Examples\Hardware Design\M9703_FDK\M9703_Design_Template\ M9703_FDK_Design_Template.wsv. Save this example workspace to your working directory or copy folder M9703DesignTemplate from workspace tree of this example workspace to that of your own workspace.
2. Drag-and-drop subnet M9703_TEMPLATE from workspace to a schematic. Double click the M9703 Design Template model in your schematic to open its GUI. You can configure which FPGAs you want to use and the interface (OutPort, Register, BlockRegister and Inter-FPGA I/O) of your selected FPGAs.
3. Find the M9703_FPGA*x* subnets whose corresponding DPU FPGA you enable in M9703 Design Template GUI. Double click them in workspace tree to open the subnet. Then you can create your own model-based design and connect your design with your defined FPGA interfaces. Note that you can only use the models in SystemVue "Hardware Design Library", the models beyond this library can't generate HDL codes and can't be implemented on M9703 DPU FPGAs finally.
4. Then after you finish your model-based design entry, you can go back to the top level schematic to create the input waveforms for the corresponding input ports of M9703 Design Template model and link the output ports of M9703 Design Template model to data sink models. Then you can run SystemVue simulation to verify your M9703 FPGA design via SystemVue software simulation.

### Users FPGA Design Interfaces in M9703 Design Template

In Overview of M9703 High-Speed Digitizer, M9703 DPU FPGAs' physical interfaces are introduced. Each DPU FPGA has the following physical interface:

- ADC parallel data streams input
- Connectivity with two DDR3 memory and one QDRII memory
- Inter FPGAs data stream connectivity
- PCIe connectivity with backplane via PCIe switch
- Control signals from CTRL FPGA

In order to make the custom real-time processing part implement under M9703's hardware architecture easily, M9703 Design Template simplifies these physical interfaces. These simplified interfaces have less relationship with physical protocol, and they are all algorithm-level interfaces, so users can connect these simplified interfaces with their FPGA design easier. So users can save the time that is spent to study the details of M9703 DPU FPGAs' physical interfaces.

There are 6 types simplified interfaces in M9703 Design Template:

- ADC Input
- OutPort

- Register
- BlockRegister
- Inter-FPGA I/O
- Trigger

You can configure these interfaces in M9703 Design Template model GUI for your selected M9703 DPU FPGAs and connect your configured interfaces with your model-based fixed-point design in the corresponding users FPGA design subnets ( M9703_FPGA*x* ).

Register, BlockRegister, Inter-FPGA I/O and Trigger are optional interfaces. You can configure whether to use them in M9703 Design Template model GUI.

These simplified interfaces have been placed in users FPGA design subnets M9703_FPGA*x* in advance, and the I/O ports of M9703_FPGA*x* subnets are controlled by your configuration in M9703 Design Template model GUI automatically. Net labels of lines are used to connect these ports. Users can use net labels to connect their own model-based design with these interfaces. For details of how to use net labels of connection line, please refer to Connection Line Net Labels.

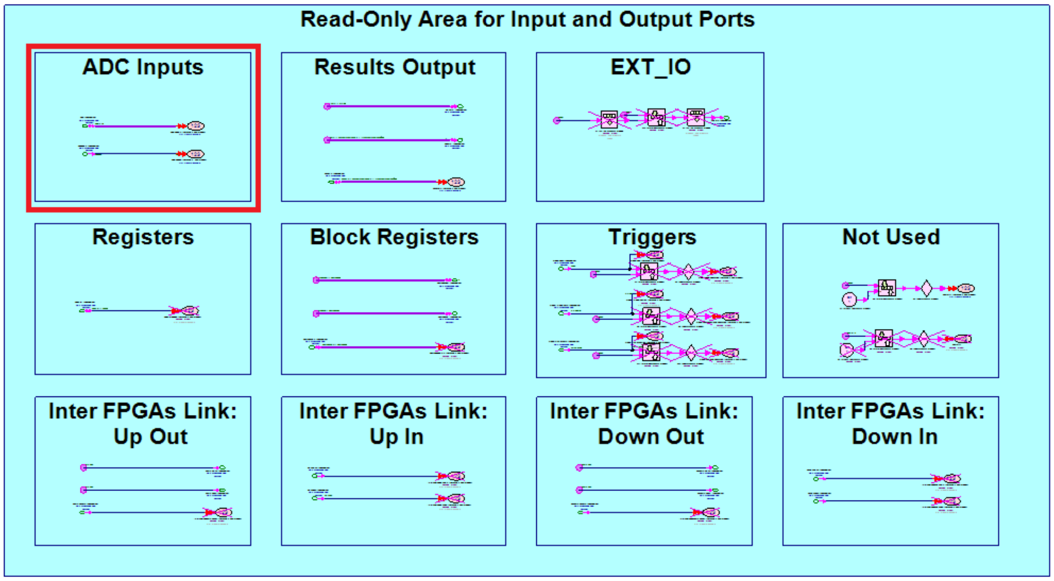
### ADC Input

ADC Input is corresponding to M9703 DPU FPGAs' physical interface ADC Input (see Overview of M9703 High-Speed Digitizer). Users can use this interface to connect ADC parallel input streams with their own FPGA design.

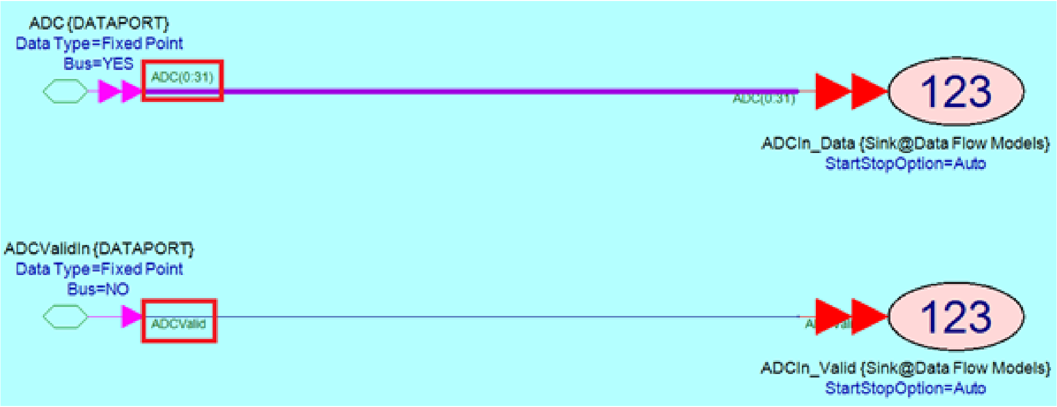
ADC Input in M9703 Design Template has the same behavior as it physical ADC Input parallel streams. It provides parallel ADC input streams and a valid signal to indicate whether the corresponding ADC parallel streams are valid. As described in Overview of M9703 High-Speed Digitizer, there are valid parallel ADC samples every two DPU FPGA system clock cycles.

Connect ADC input with your FPGA design in subnet M9703_FPGA0 ~ M9703_FPGA3:

In user FPGA design subnet M9703_FPGA0 ~ M9703_FPGA3, ADC parallel input streams are a input bus with width 32 and a valid input port. They are configured by M9703 Design Template model GUI automatically.



The input ports are connected to lines with Net Label. The Net Label of parallel ADC input data is ADC(0:31). Net Label of ADC input valid signal is ADCValid. So user can use Net Label name to connect the parallel ADC input with their own model-based FPGA design.



As described in Overview of M9703 High-Speed Digitizer, ADC input data ADC(0:31) is related M9703 ADC sample mode:

For Normal, all 8 channels mode:

- IN1 and IN2 connect with FPGA0;
- IN3 and IN4 connect with FPGA1;
- IN5 and IN6 connect with FPGA2;
- IN7 and IN8 connect with FPGA3.

ADC(0:15) are 16 parallel ADC samples of the first channel for current FPGA. ADC(0) is the oldest sample and ADC(15) is the newest sample.
ADC (16:31) are 16 parallel ADC samples of the second channel for current FPGA. ADC(16) is the oldest sample and ADC(31) is the newest sample.

For example, IN5 and IN6 connect with FPGA2. In subnet M9703_FPGA2, ADC(0:15) are 16 parallel samples of IN5 and ADC(16:31) are 16 parallel samples of IN6.

For Interleaved, 1+3+5+7 mode:

- IN1 connects with FPGA0;
- IN3 connects with FPGA1;
- IN5 connects with FPGA2;
- IN7 connects with FPGA3.

ADC(0:31) are 32 parallel samples of the connected channel of the current FPGA. ADC(0) is the oldest sample and ADC(31) is the newest sample.

For example, if IN5 connects with FPGA2 for interleaved mode, in subnet M9703_FPGA2, ADC(0:31) are 32 parallel samples of IN5.

For Interleaved, 2+4+6+8 mode:

- IN2 connects with FPGA0;
- IN4 connects with FPGA1;
- IN6 connects with FPGA2;
- IN8 connects with FPGA3.

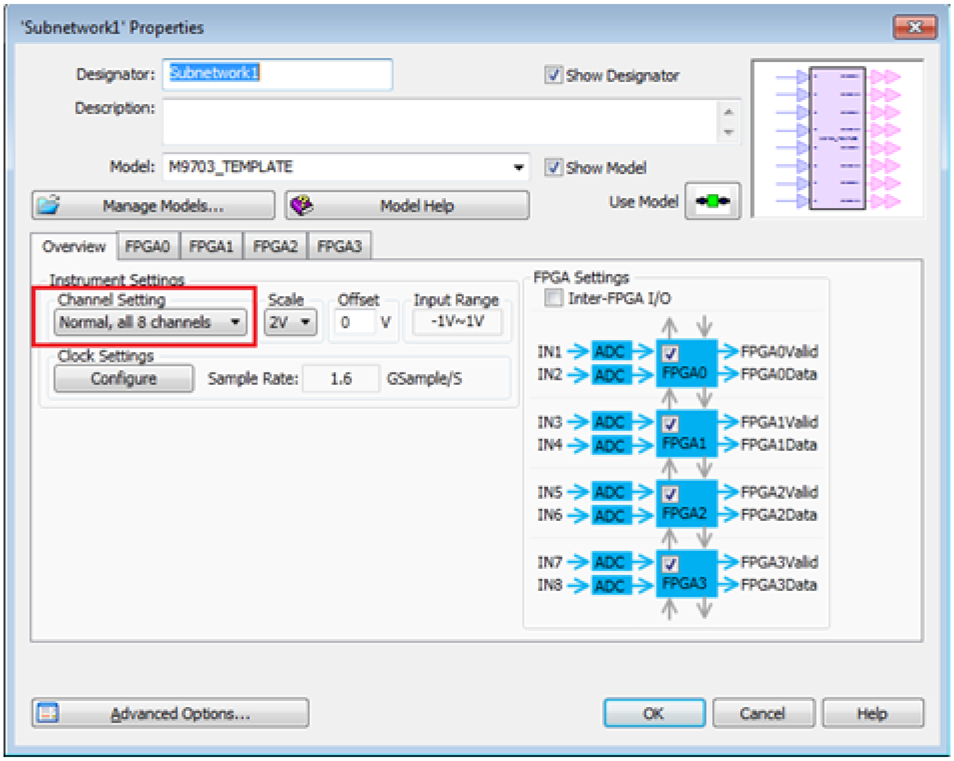
ADC(0:31) are 32 parallel samples of the connected channel of the current FPGA. ADC(0) is the oldest sample and ADC(31) is the newest sample.

For example, if IN2 connects with FPGA0 for interleaved mode, in subnet M9703_FPGA0, ADC(0:31) are 32 parallel samples of IN2.

Configure ADC input mode in GUI of M9703 Design Template:

ADC sample mode can be configured in M9703 Design Template model GUI. Then in each FPGA tab of the GUI, the connection relationship between IN1~IN8 and ADC (0:31) is also shown.

For example, Normal, all 8 channels mode and FPGA0 tab:



'Subnetwork1' Properties

Designator: Subnetwork1 ☑ Show Designator

Description:

Model: M9703_TEMPLATE ☑ Show Model

Overview | FPGA0 | FPGA1 | FPGA2 | FPGA3

Instrument Settings

Channel Setting: Normal, all 8 channels

Scale: 2V

Offset: 0 V

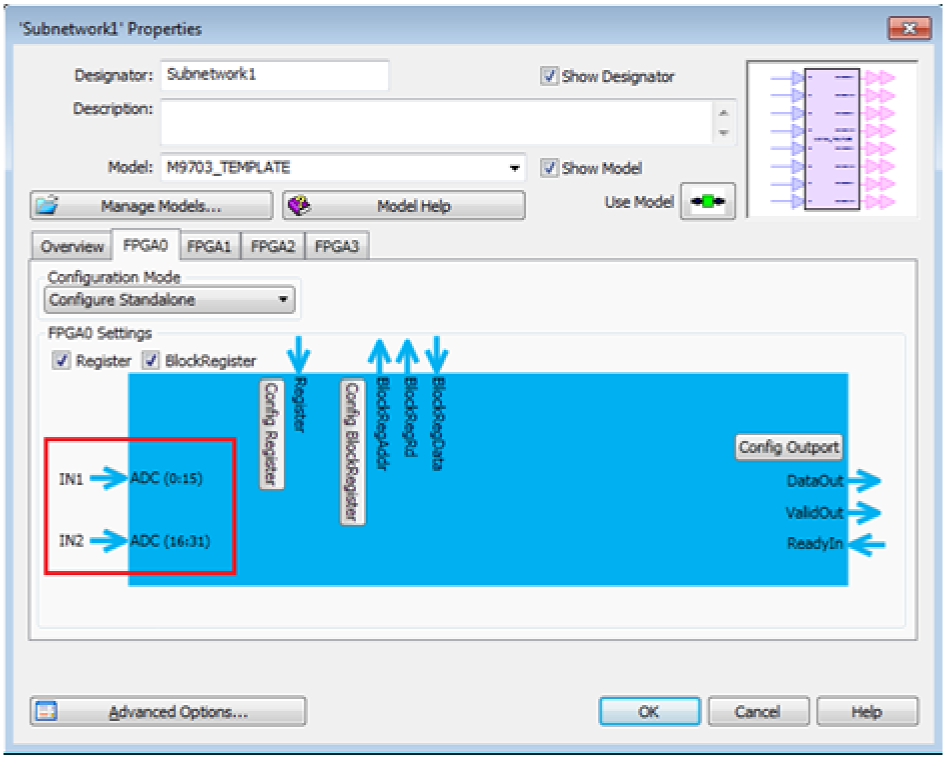
Input Range: -1V~1V

Clock Settings

Sample Rate: 1.6 GSample/S

FPGA Settings

☐ Inter-FPGA I/O



'Subnetwork1' Properties

Designator: Subnetwork1 ☑ Show Designator

Description:

Model: M9703_TEMPLATE ☑ Show Model

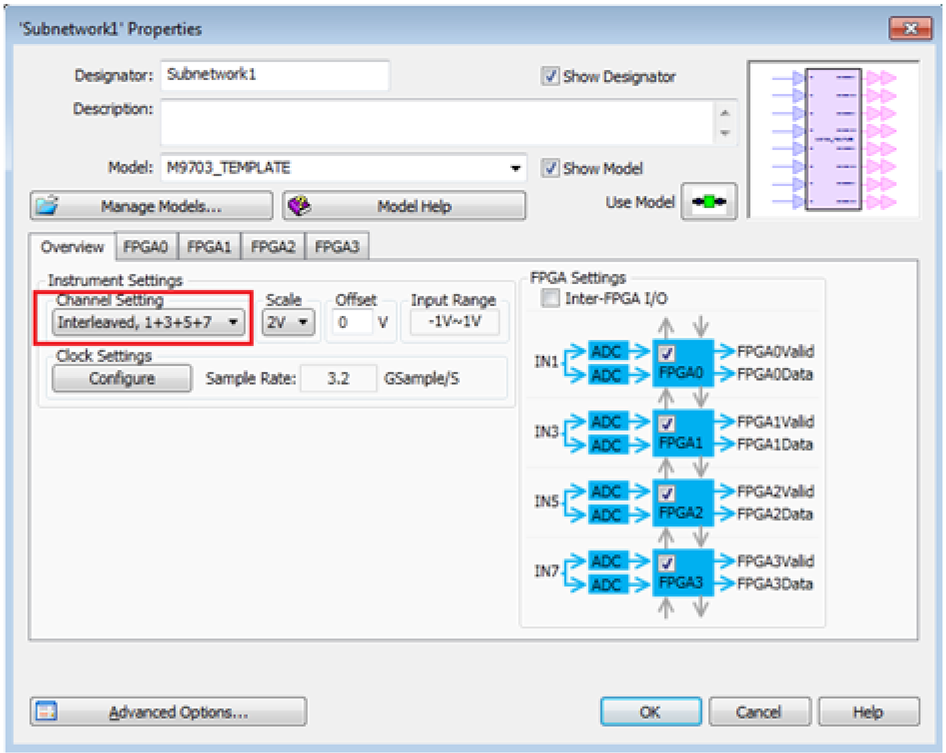
Overview | FPGA0 | FPGA1 | FPGA2 | FPGA3

Configuration Mode: Configure Standalone

FPGA0 Settings

☑ Register ☑ BlockRegister

For example, Interleaved, 1+3+5+7 mode and FPGA1 tab:



'Subnetwork1' Properties

Designator: Subnetwork1

☑ Show Designator

Description:

Model: M9703_TEMPLATE

☑ Show Model

Overview | FPGA0 | FPGA1 | FPGA2 | FPGA3

Instrument Settings

Channel Setting: Interleaved, 1+3+5+7

Scale: 2V

Offset: 0 V

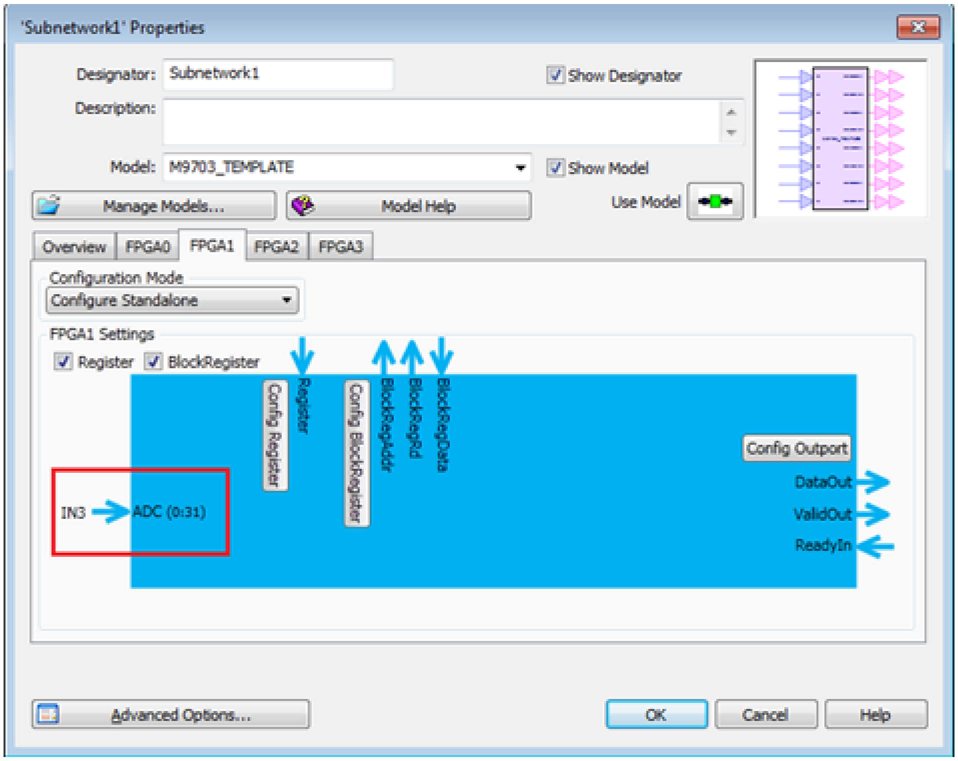
Input Range: -1V~1V

Clock Settings

Sample Rate: 3.2 GSample/S

FPGA Settings

☐ Inter-FPGA I/O



'Subnetwork1' Properties

Designator: Subnetwork1

☑ Show Designator

Description:

Model: M9703_TEMPLATE

☑ Show Model

Overview | FPGA0 | FPGA1 | FPGA2 | FPGA3

Configuration Mode: Configure Standalone

FPGA1 Settings

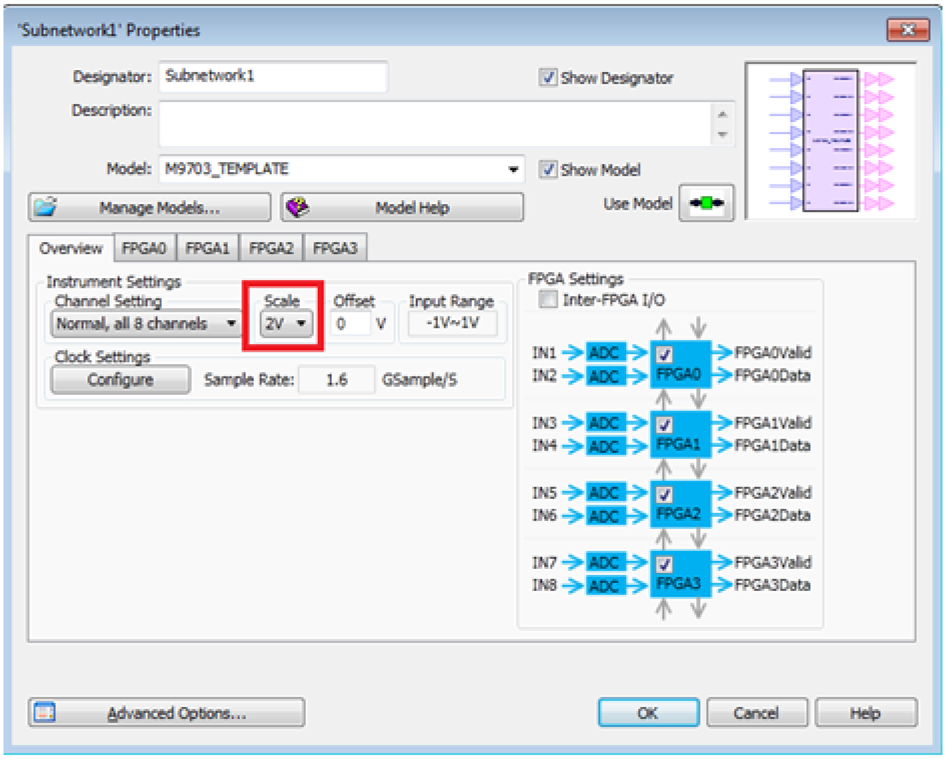
☑ Register ☑ BlockRegister

Each line of bus ADC(0:31) is an input port of ADC sample and its data type is F ixed-point.

- Its WordLength is 16 bit.

- Its Integer WordLength is 1 bit when Full Scale is 2V (Input Range: -1V~1V); Its Integer WordLength is 0 bit when Full Scale is 1V (Input Range: -0.5V~0.5V)
- It's Signed.

Full Scale can be set in M9703 Design Template model GUI:



'Subnetwork1' Properties

Designator: Subnetwork1

☑ Show Designator

Description:

Model: M9703_TEMPLATE

☑ Show Model

Manage Models...

Model Help

Use Model

Overview | FPGA0 | FPGA1 | FPGA2 | FPGA3

Instrument Settings

Channel Setting: Normal, all 8 channels

Scale: 2V

Offset: 0 V

Input Range: -1V~1V

Clock Settings

Configure

Sample Rate: 1.6 GSample/S

FPGA Settings

□ Inter-FPGA I/O

Advanced Options...

OK

Cancel

Help

The data type of M9703 Design Template model's input ports IN1~IN8 is Real. M9703 Design Template will quantize the Real input waveforms to get fixed-point ADC sample value for ADC(0:31) according to M9703's ADC behavior. It will quantize the Real waveform to a fixed-point number with WordLength 12 and then extend the 12bit fixed-point number to 16bit. Integer WordLength and Signed are always as described above. Please refer to Overview of M9703 High-Speed Digitizer for M9703's ADC behavior.

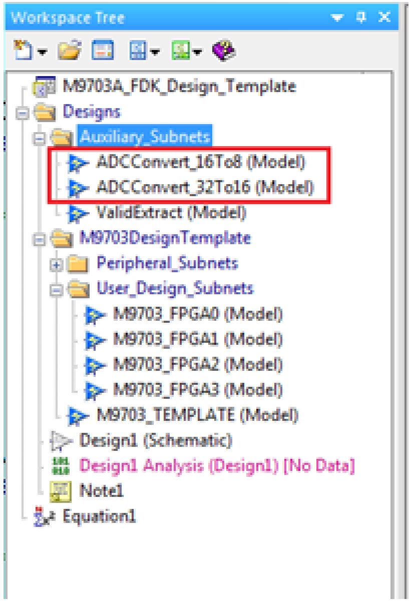
Subnets that can help you remove ADCValid:

ADCValid is Net Label of the non-bus ADC valid input port. Its data type is also fixed-point.

- Its WordLength is 1 bit.
- Its Integer WordLength is 1 bit
- It's UnSigned.

ADCValid alternates between 0 and 1 always. When ADCValid is 1, the corresponding ADC(0:31) are valid ADC samples. When ADCValid is 0, ADC(0:31) will hold their current values. So every two samples, there is a valid ADC(0:31).

If you want to get all valid ADC parallel samples (valid signal is always 1, so all ADC samples are valid. Then you need not ADC valid signal), two subnets ADCConvert_16To8 and ADCConvert_32To16 are provided in example workspace: *<your SystemVue installation dir>*\Examples\Hardware Design\M9703_FDK\M9703_Design_Template\ M9703_FDK_Design_Template.wsv

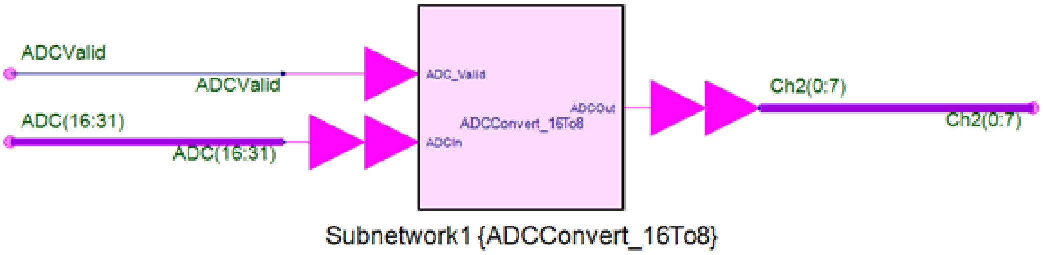


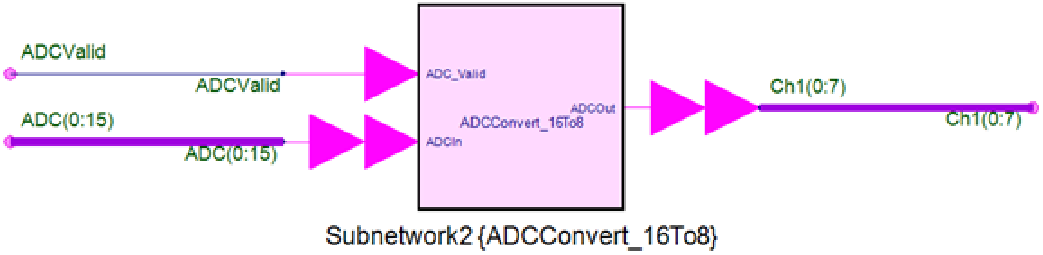
Workspace Tree

- M9703A_FDK_Design_Template
  - Designs
    - Auxiliary_Subnets
      - ADCConvert_16To8 (Model)
      - ADCConvert_32To16 (Model)
      - ValidExtract (Model)
    - M9703DesignTemplate
      - Peripheral_Subnets
      - User_Design_Subnets
        - M9703_FPGA0 (Model)
        - M9703_FPGA1 (Model)
        - M9703_FPGA2 (Model)
        - M9703_FPGA3 (Model)
      - M9703_TEMPLATE (Model)
    - Design1 (Schematic)
    - Design1 Analysis (Design1) [No Data]
    - Note1
  - Equation1

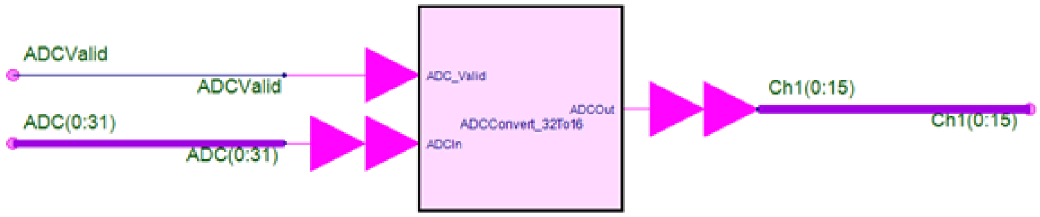
Because ADCValid is 1 every two samples, the width of bus ADC needs go down 2 times when you get all valid parallel streams.

For Normal, all 8 channels mode, the width of bus ADC become from 16 to 8, you can use subnet ADCConvert_16To8 as below to get all valid parallel ADC samples for two channels.





For Interleaved, 1+3+5+7 or Interleaved, 2+4+6+8 mode, the width of bus ADC become from 32 to 16, you can use subnet ADCConvert_32To16 as below to get all valid parallel ADC samples for one channel.



Subnetwork3 {ADCConvert_32To16}

It is not mandatory to connect all ADC(0:31) lines with user's FPGA design in subnets M9703_FPGA0~M9703_FPGA3. For example, if you just want to use IN1 and don't use IN2 for FPGA0, then you can only connect ADC(0:15) to your design and don't use ADC(16:31) in subnet M9703_FPGA0.

NOTE **Net Label Key Words for ADC Input**

ADC parallel input data: **ADC(0:31)**

ADC Valid: **ADCValid**

Use the net labels to connect your design with ADC input interface in M9703 user design subnets **M9703_FPGA0 ~ M9703_FPGA3.**

### OutPort

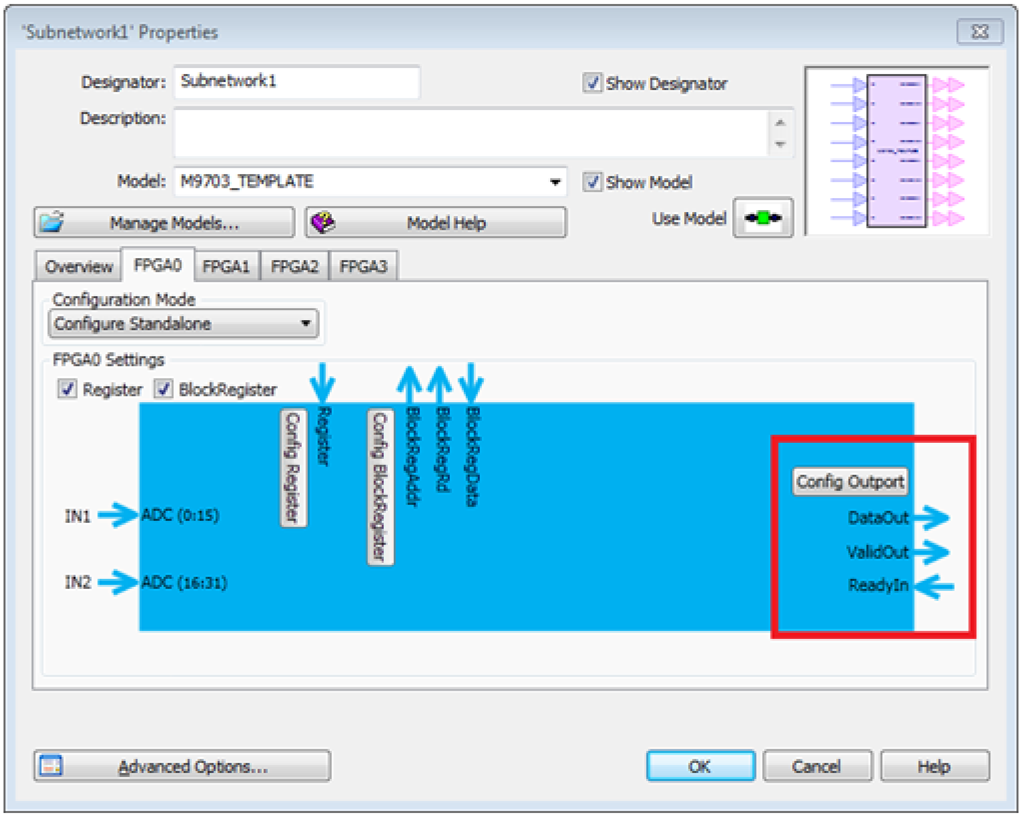
Users can process ADC input samples with their own FPGA design in subnets M9703_FPGA0~M9703_FPGA3. Then the processed results can be outputted via OutPort. Users can define multiple OutPort and each their desired output result can be outputted via the corresponding OutPort. Because users' output results from M9703 Design Template user FPGA design subnets (M9703_FPGA0~M9703_FPGA3 ) are Fixed-point data type, the corresponding OutPort can be defined as users desired Fixed-point data format (WordLength, Integer WordLength and IsSigned).

Because the real-time processed results from M9703 DPU FPGAs may be high-speed data stream, PCIe bandwidth may not be enough to fetch all results to PC without data loss. And also, software running on PC is usually too slow to process the real-time processed results from M9703 hardware. So in M9703 FPGA design flow, the real-time processed results are buffered to DDR3 SDRAM on M9703 at first, because the data bandwidth between DPU FPGA and DDR3 SDRAM is much larger than PCIe data bandwidth. User can specify a block of SDRAM for the data buffering. When the block of SDRAM is filled in, SystemVue that runs a M9703 instrument co-simulation on a computer can read back the buffered results via PCIe. Then SystemVue can parse PCIe data format to user defined OutPort Fixed-point data type automatically.

So user just need define their own OutPort Fixed-point data type for software simulation and M9703 instrument co-simulation and don't need care the physical DDR3 and PCIe protocol and data format. SystemVue will automatically generate HDL wrapper to convert between user defined OutPort data format and DDR3 and PCIe data format during generating M9703 DPU FPGA programming file.

Define your own OutPort in GUI of M9703 Design Template:

For each M9703 DPU FPGA, OutPort can be defined in the corresponding FPGA*x* tab of M9703 Design Template model GUI:


'Subnetwork1' Properties

Designator: Subnetwork1

☑ Show Designator

Description:

Model: M9703_TEMPLATE

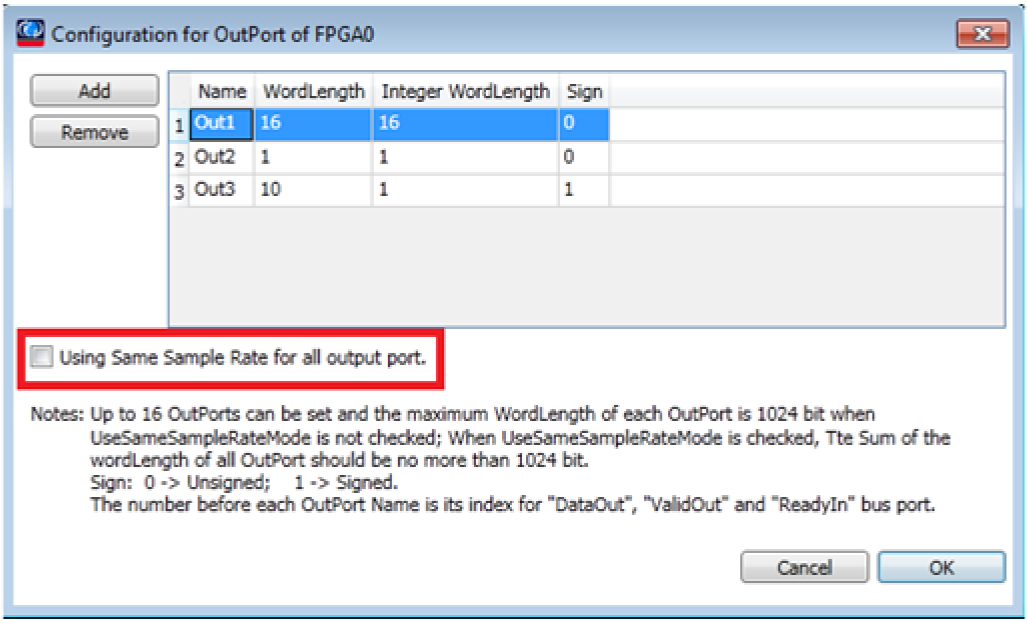
☑ Show Model

Configuration Mode: Configure Standalone

FPGA0 Settings: ☑ Register ☑ BlockRegister

As shown in the interface diagram, each OutPort is represented as a set of ports: DataOut output port, ValidOut output port and ReadyIn input port. It's a typical data stream style interface. DataOut is the data values output. ValidOut shows the corresponding DataOut is valid when it's 1, otherwise when it's 0, the corresponding DataOut is not valid. ReadyIn shows the following hardware is ready to receive the current output. When both ValidOut and ReadyIn are 1, a data transfer occurs.

Click button Config Outport to open a new GUI to define the OutPort for each DPU FPGA.



Configuration for OutPort of FPGA0

|  | Name | WordLength | Integer WordLength | Sign |
|---|---|---|---|---|
| 1 | Out1 | 16 | 16 | 0 |
| 2 | Out2 | 1 | 1 | 0 |
| 3 | Out3 | 10 | 1 | 1 |

□ Using Same Sample Rate for all output port.

Notes: Up to 16 OutPorts can be set and the maximum WordLength of each OutPort is 1024 bit when UseSameSampleRateMode is not checked; When UseSameSampleRateMode is checked, Tte Sum of the wordLength of all OutPort should be no more than 1024 bit.
Sign: 0 -> Unsigned; 1 -> Signed.
The number before each OutPort Name is its index for "DataOut", "ValidOut" and "ReadyIn" bus port.

In the OutPort configuration GUI, user can define multiple OutPort. User needs define the following items for each OutPort:

- Name
- WordLength
- Integer WordLength
- Sign

Name is a meaningful string to help you to identify each OutPort. WordLength, Integer WordLength, and Sign specify the fixed-point format for data of the corresponding OutPort. The fixed point format is just used to specify DataOut. Both ValidOut and ReadyIn are 1 bit unsigned integer (its value is 0 or 1).

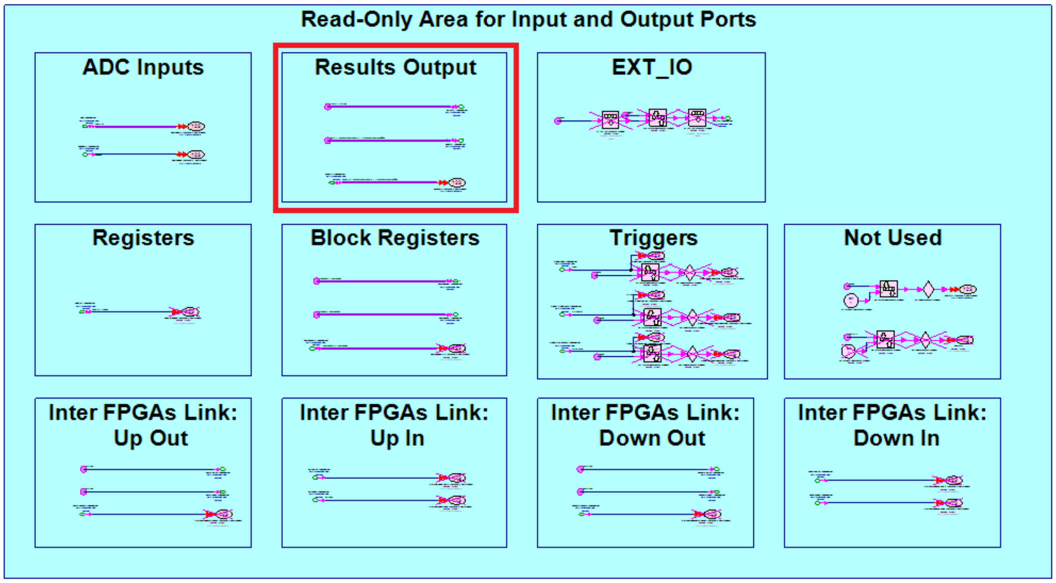
In each row, an OutPort is defined. There is a number on the left of OutPort's Name . The number is the index of each OutPort. The index is a very useful number for each defined OutPort. In user FPGA design subnets M9703_FPGA0~M9703_FPGA3, OutPort are bus IO ports. This index is the bus index for each defined OutPort. The index is one-based.

OutPort has two working modes:

- Same Sample Rate Mode: checkbox Using Same Sample Rate for all output port is checked. All defined OutPorts share the same ValidOut signal. So they will generate valid output in the same rate. In this mode, you can define up to 16 OutPorts for each M9703 FPGA, and the maximum bit width of each OutPort is 1024.
- Non-Same Sample Rate Mode: checkbox Using Same Sample Rate for all output port is unchecked. Each defined OutPort has its own ValidOut signal. So they can generate valid output in different rates. In this mode, the sum of all OutPorts' bit width must be less or equal to 1024.

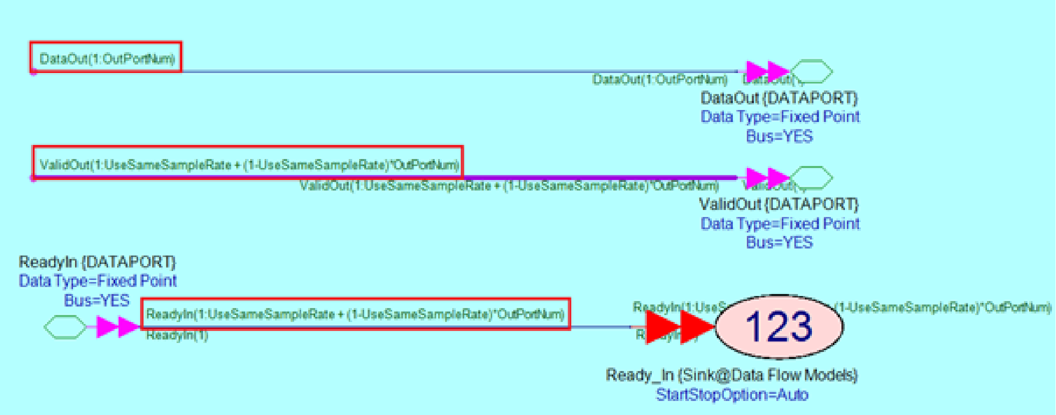
Connect OutPort with your FPGA design in subnet M9703_FPGA0 ~ M9703_FPGA3:

In user FPGA design subnet M9703_FPGA0 ~ M9703_FPGA3, the IO ports of your defined OutPort in GUI are configured by M9703 Design Template model GUI automatically. You don't need edit them and you just need use net labels to connect your model-based design with the pre-configured IO ports.



The OutPort interface is represented by a set of fixed-point bus ports. If you defined N OutPort for FPGA*x*, the OutPort interface will be:

- Data output bus: DataOut(1:N)
- Valid output bus: For Non-Same Sample Rate Mode, ValidOut(1:N); For Same Sample Rate Mode, ValidOut(1). All defined OutPort will share the same ValidOut(1).
- Ready input bus: For Non-Same Sample Rate Mode, ReadyIn(1:N); For Same Sample Rate Mode, ReadyIn(1). All defined OutPort will share the same ReadyIn(1).


DataOut(1:OutPortNum)
DataOut(1:OutPortNum)
DataOut {DATAPORT}
Data Type=Fixed Point
Bus=YES
ValidOut(1:UseSameSampleRate + (1-UseSameSampleRate)*OutPortNum)
ValidOut(1:UseSameSampleRate + (1-UseSameSampleRate)*OutPortNum)
ValidOut {DATAPORT}
Data Type=Fixed Point
Bus=YES
ReadyIn {DATAPORT}
Data Type=Fixed Point
Bus=YES
ReadyIn(1:UseSameSampleRate + (1-UseSameSampleRate)*OutPortNum)
ReadyIn(1)
(1-UseSameSampleRate)*OutPortNum)
123
Ready_In {Sink@Data Flow Models}
StartStopOption=Auto

The number of OutPort and fixed-point data type of each DataOut are defined in GUI of M9703 Design Template. The fixed-point data type of all ValidOut and ReadyIn is one bit logic (unsigned, word length = integer word length = 1).

For each OutPort, it has a set of ports DataOut(*x*) output, ValidOut(*x*) output and ReadyIn(*x*) input. The set of ports follows the handshaking rule of AXI4-stream. So you also need follow the AXI4-stream handshaking rule to deal with the timing of DataOut, ValidOut and ReadyIn ports in your application design logic. You can use net labels DataOut(*x*), ValidOut(*x*) and ReadyIn(*x*) to connect the set of ports with your application design ports.

- In Non-Same Sample Rate Mode, *x* is the bus index between 1:N (N is the number of your defined OutPort).
- In Same Sample Rate Mode, for DataOut(*x*), *x* is the bus index between 1:N ( N is the number of your defined OutPort). For ValidOut(*x*) and ReadyIn(*x*), *x* can only be 1.

For example, if you define 2 OutPorts in GUI of M9703 Design Template, you can:

In Non-Same Sample Rate Mode:

- Connect the data output of the first OutPort to the line with net label DataOut(1);
- Connect the valid output of the first OutPort to the line with net label ValidOut(1);
- Connect the ready input of the first OutPort to the line with net label ReadyIn (1);
- Connect the data output of the second OutPort to the line with net label DataOut(2);
- Connect the valid output of the second OutPort to the line with net label ValidOut(2);
- Connect the ready input of the second OutPort to the line with net label ReadyIn(2);

In Same Sample Rate Mode,

- Connect the data output of the first OutPort to the line with net label DataOut(1);
- Connect the data output of the second OutPort to the line with net label DataOut(2);
- Just generate one valid out for all output data in your FPGA design and connect it to the line with net label ValidOut(1);
- Connect the line with net label ReadyIn(1) as input to your FPGA design to control your output

For each OutPort, your fixed-point design needs follow AXI4-stream handshaking rule for the timing of data, valid and ready signals. When both ValidOut and ReadyIn are 1, a data transfer occurs.

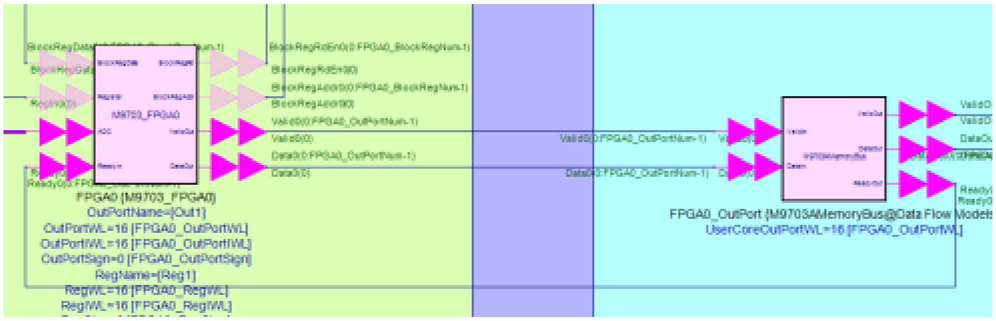
NOTE

- If you define **N OutPort** in GUI, don't use net labels **DataOut(*x*)**, **ValidOut(*x*)** and **ReadyIn(*x*)** that ***x*** is beyond the range of **1 to N.** Otherwise, when you run the simulation, SystemVue will post you an error.

- Especially, in **Same Sample Rate** Mode**,** if ***x*** for **ValidOut(*x*)** and **ReadyIn(*x*)** is not **1,** when you run the simulation, SystemVue will post you an error.

- The fixed-point data format of your data output must match with the corresponding **OutPort** definition in GUI of M9703 Design Template. And the fixed-point data format of your **ValidOut(*x*)** must be one-bit logic (word length = integer word length = 1, unsigned). Otherwise, SystemVue will post an error when running the simulation.

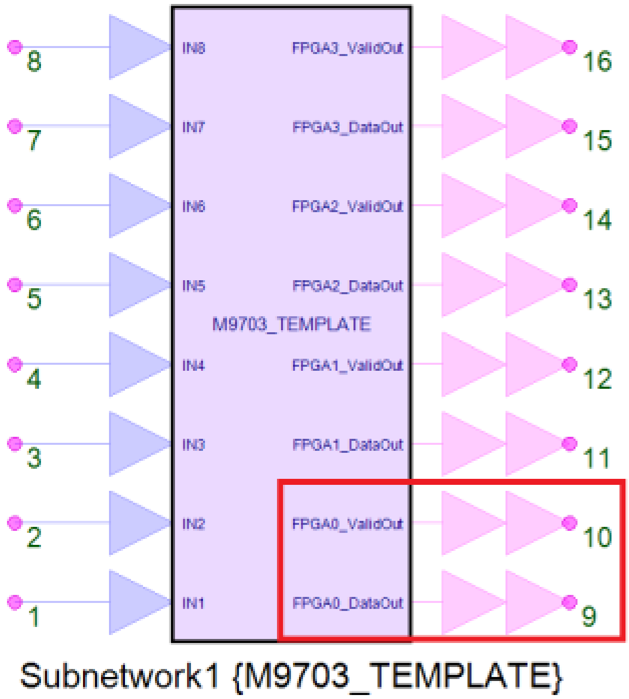
In M9703 Design Template, each selected FPGA has an M9703MemoryBus model to link its input with DataOut and ValidOut output buses of M9703_FPGAx subnet and generate output to ReadyIn input bus of M9703_FPGAx subnet. You can observe it in M9703_TEMPLATE subnet:


BlockRegRdEn0(0:FPGA0_BlockRegNum-1)
BlockRegRdEn0(0)
BlockRegAddr0(0:FPGA0_BlockRegNum-1)
BlockRegAddr0(0)
M9703_FPGA0
ADC
ReadyIn
ValidOut
DataOut
Valid0(0:FPGA0_OutPortNum-1)
Valid0(0)
Data0(0:FPGA0_OutPortNum-1)
Data0(0)
FPGA0 {M9703_FPGA0}
OutPortName=[Out1]
OutPortWL=16 [FPGA0_OutPortWL]
OutPortIWL=16 [FPGA0_OutPortIWL]
OutPortSign=0 [FPGA0_OutPortSign]
RegName=[Reg1]
RegWL=16 [FPGA0_RegWL]
RegIWL=16 [FPGA0_RegIWL]
Valid0(0:FPGA0_OutPortNum-1)
Data0(0:FPGA0_OutPortNum-1)
ValidIn
DataIn
M9703AMemoryBus
ValidOut
DataOut
ReadyOut
FPGA0_OutPort {M9703AMemoryBus@Data Flow Models
UserCoreOutPortWL=16 [FPGA0_OutPortWL]

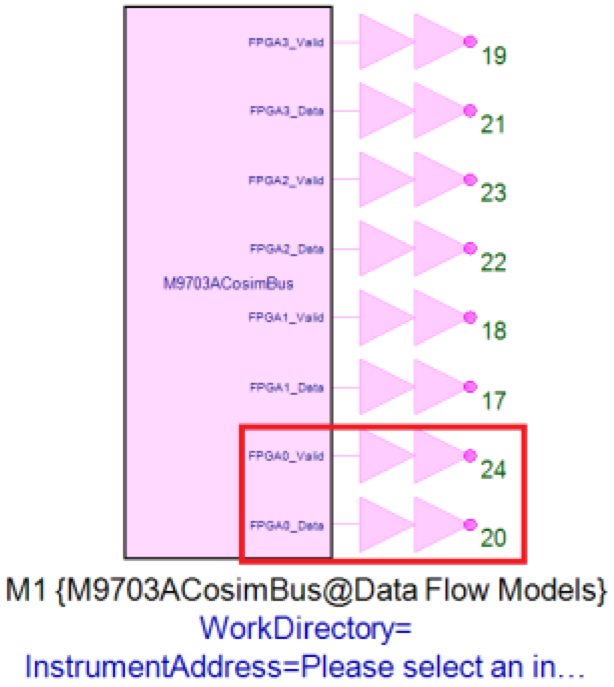
When SystemVue generates FPGA programming file automatically, it inserts an N-to-1 AXI4-stream switcher to connect N defined OutPorts to FPGA DDR3 RAM AXI4 interface. It also packages different OutPort data with header indicators. The M9703MemoryBus model will mimic the hardware behavior to generate ReadyIn feedback for user's application design when you do the software simulation.

<u>OutPort on top level M9703 Design Template model and M9703CosimBus model:</u>

As your desired results are outputted via OutPort, the results will be outputted from top level M9703 Design Template model in software simulation, and will be outputted from M9703CosimBus model in M9703 instrument co-simulation.



For top level M9703 Design Template model, it has a set of output bus ports for each FPGA, FPGA*x*_DataOut and FPGA*x*_ValidOut. They have the same function as the output from M9703 user FPGA design subnets M9703_FPGA0 ~ M9703_FPGA3. But as it mimics the hardware behavior and when SystemVue reads data from M9703, actually the data have been buffered in on-board DDR3 SDRAM. So it doesn't need a ReadyIn port anymore.



For M9703CosimBus model, its output ports are similar as top level M9703 Design Template model. It has also a set of output bus ports for each FPGA, FPGA*x*_Data and FPGA*x*_Valid and doesn't need ReadyIn port.

For the top level M9703 Design Template model and M9703CosimBus model, the DataOut values of OutPort in M9703_FPGA0~M9703_FPGA3 subnets can be outputted through FPGAx_DataOut, but the behavior of the FPGAx_ValidOut will be different . Please refer to.

NOTE **Net Label Key Words for OutPort**

If you define **N OutPort**, the net label you can use to connect your FPGA design in **M9703_FPGA0~M9703_FPGA3** subnets:

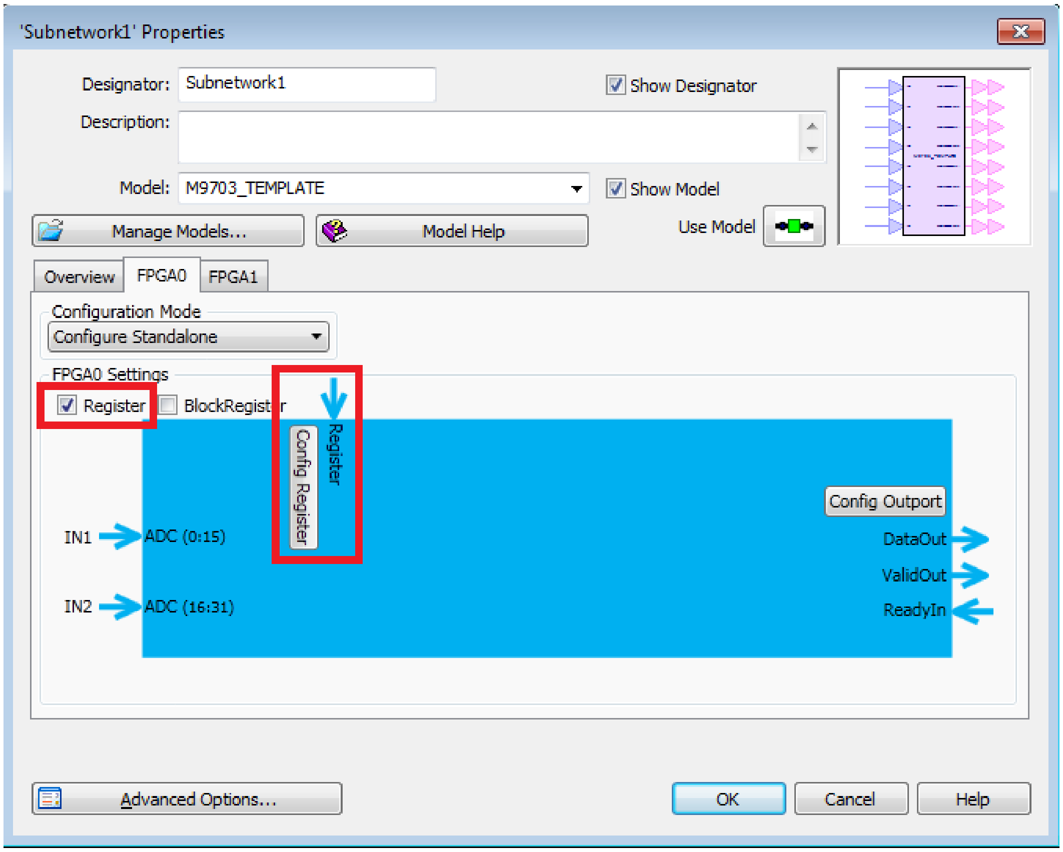
- For **Non-Same Sample Rate** mode: **DataOut(1:N)**, **ValidOut(1:N)** and **ReadyIn(1:N)**. **DataOut(1:N)** and **ValidOut(1:N)** are output ports and **ReadyIn(1:N)** is input port.
- For **Same Sample Rate** mode: **DataOut(1:N)**, **ValidOut(1)** and **ReadyIn(1)**. **DataOut(1:N)** and **ValidOut(1)** are output ports and **ReadyIn(1)** is input port.

## Register

Register is an optional FPGA interface. You can define arbitrary number of Registers as long as it can be implemented on DPU FPGA. Each Register reserves a value that can be configured dynamically after FPGA programming file is generated. It can make your generated FPGA programming file more flexible.

Configure Register in M9703 Design Template GUI:

You can enable or disable Register in M9703 Design Template model GUI by checking or unchecking Register checkbox as shown below.



'Subnetwork1' Properties

Designator: Subnetwork1

☑ Show Designator

Description:

Model: M9703_TEMPLATE

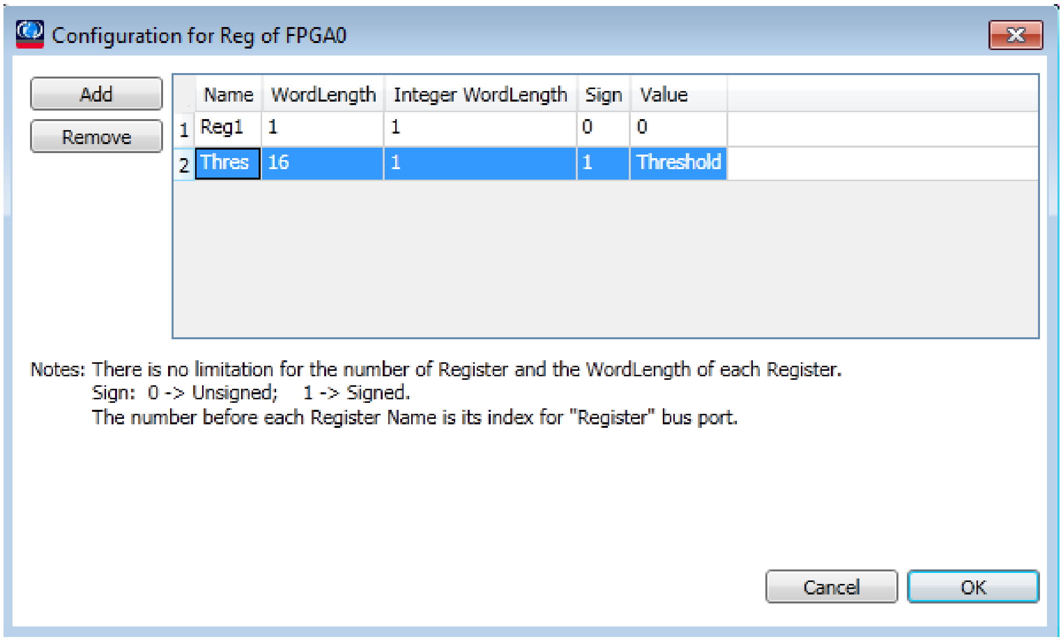
☑ Show Model

Configuration Mode: Configure Standalone

FPGA0 Settings

☑ Register □ BlockRegist r

Register interface is a set of reserved fixed-point registers whose values can be re-configured at the beginning of a simulation, and then the Register values are kept until the end of simulation. After you enable Register, you can click Config Register button to open a new GUI:



| | Name | WordLength | Integer WordLength | Sign | Value |
|---|---|---|---|---|---|
| 1 | Reg1 | 1 | 1 | 0 | 0 |
| 2 | Thres | 16 | 1 | 1 | Threshold |

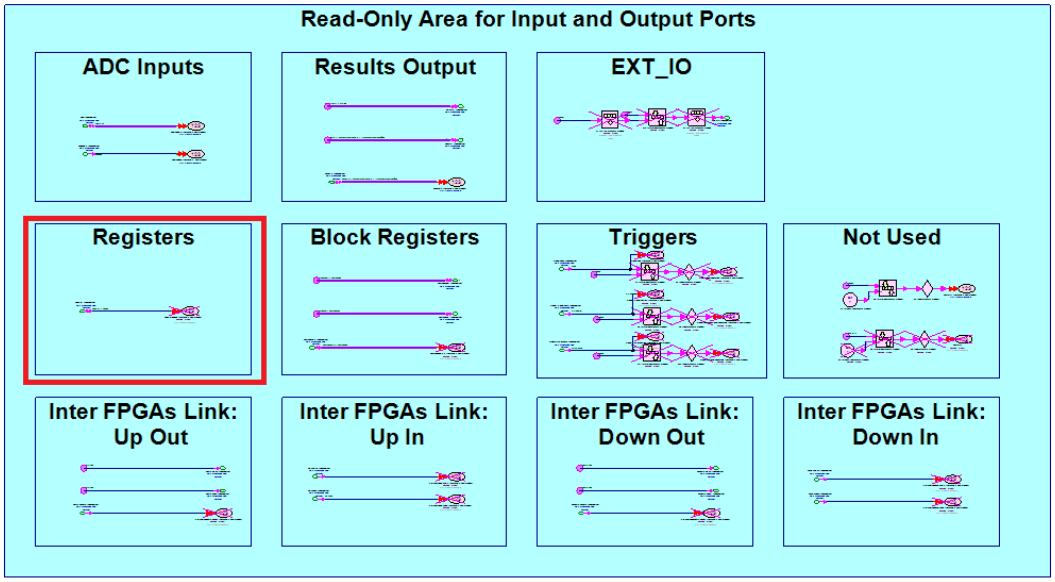
Notes: There is no limitation for the number of Register and the WordLength of each Register.
Sign: 0 -> Unsigned; 1 -> Signed.
The number before each Register Name is its index for "Register" bus port.

You can define the number of Registers and the fixed-point data format of each Register in this GUI. You can also specify Value for each defined Register, then the value of each Register will be set at the beginning of a simulation.

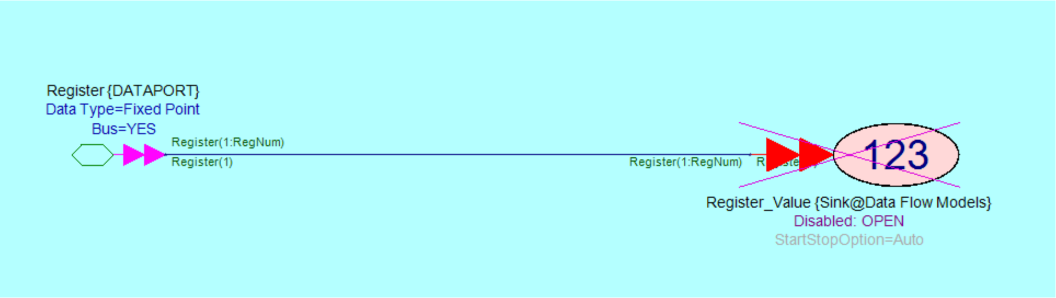
If you generate M9703 FPGA programming file for a M9703 Design Template with defined Registers, the number of Registers and the fixed-point data format of each Register are fixed in FPGA implementation, but you can still re-configure the value of each Register. Then when you run SystemVue and M9703 Co-Simulation, all Register values will be re-configured before ADC data capture and kept until the end of the Co-Simulation.

Connect Register with your FPGA design in subnet M9703_FPGA0 ~ M9703_FPGA3:

In subnet M9703_FPGAx for user's application fixed-point design, Register input bus port has been configured in advance.



If you define Register interface in GUI of M9703 Design Template, you can use the Register input port for your application fixed-point design. Assuming you defined N Registers in GUI, you can use the line with net label Register(x) from Register bus input Register(1:N) to get Register value for your fixed-pointed design.



For example, if you defined two Registers in GUI of M9703 Design Template, the first one with name threshold1 and the second one with name threshold2, assuming that they are the thresholds in your design. You leave the two Registers in order to adjust the thresholds flexibly. Then in M9703_FPGAx subnet, you can get the values of the defined Registers for your application design in this way:

- Draw a line and double click it to input its net label Register(1). Then this line will be the input of Register threshold1.
- Draw a line and double click it to input its net label Register(2). Then this line will be the input of Register threshold2.

Then you can connect the lines with Register net label to your own fixed-point design.

NOTE

- If you don't define any **Register** in GUI of M9703 Design Template, you can't use the Register net label. In addition, you can't use the net label **Register(*x*)** that ***x*** is beyond the range you defined.

- Besides, your defined Fixed-point data format of each Register must be consistent with your FPGA design, otherwise, an error will be posted when your run SystemVue simulation.

**Net Label Key Words for Register**

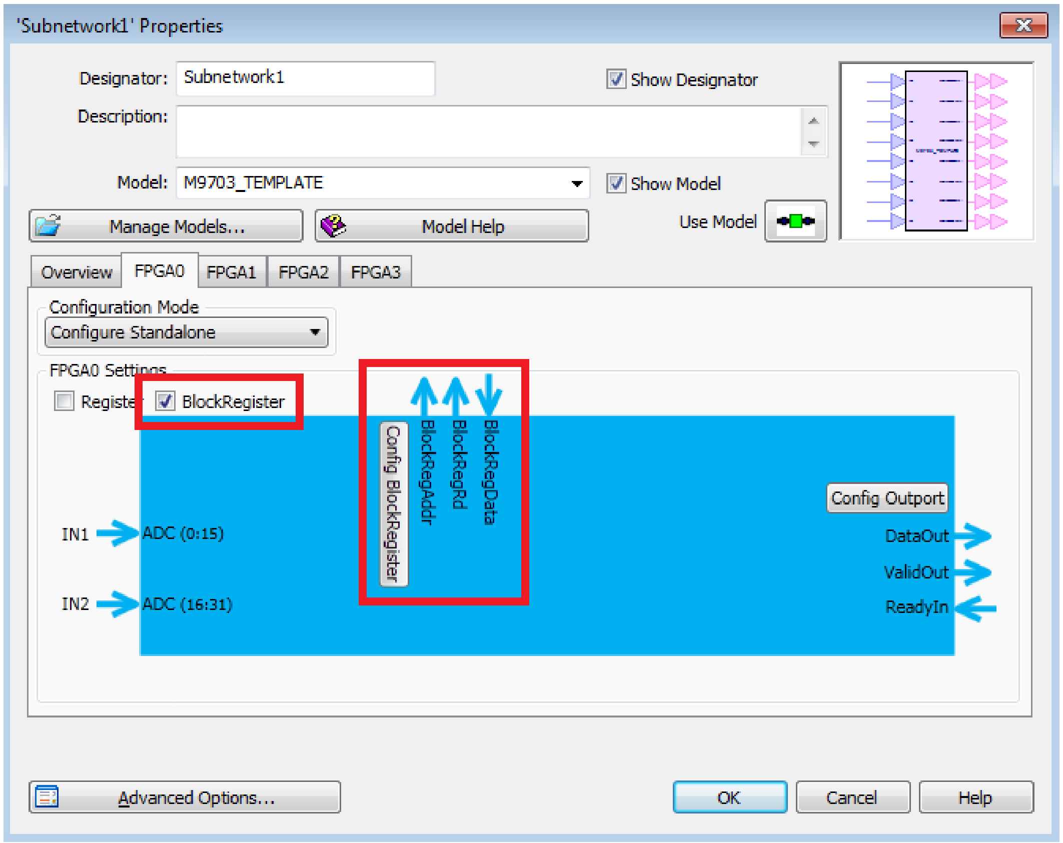
- **Register(*x*)**, where *x* is a number between 1 to N (N is the Register number you defined).

### BlockRegister

BlockRegister is an optional FPGA interface. You can define arbitrary number of BlockRegister as long as it can be implemented on DPU FPGA. Each BlockRegister reserves a table of values that can be configured dynamically after FPGA programming file is generated. It can make your generated FPGA programming file more flexible. In your M9703 FPGA design subnets, you can use an address to look up the values of the table, just like looking up a RAM.

Configure BlockRegister in M9703 Design Template GUI:

You can enable or disable it in M9703 Design Template GUI by checking or unchecking BlockRegister checkbox as shown below.



'Subnetwork1' Properties

Designator: Subnetwork1

☑ Show Designator

Description:

Model: M9703_TEMPLATE

☑ Show Model

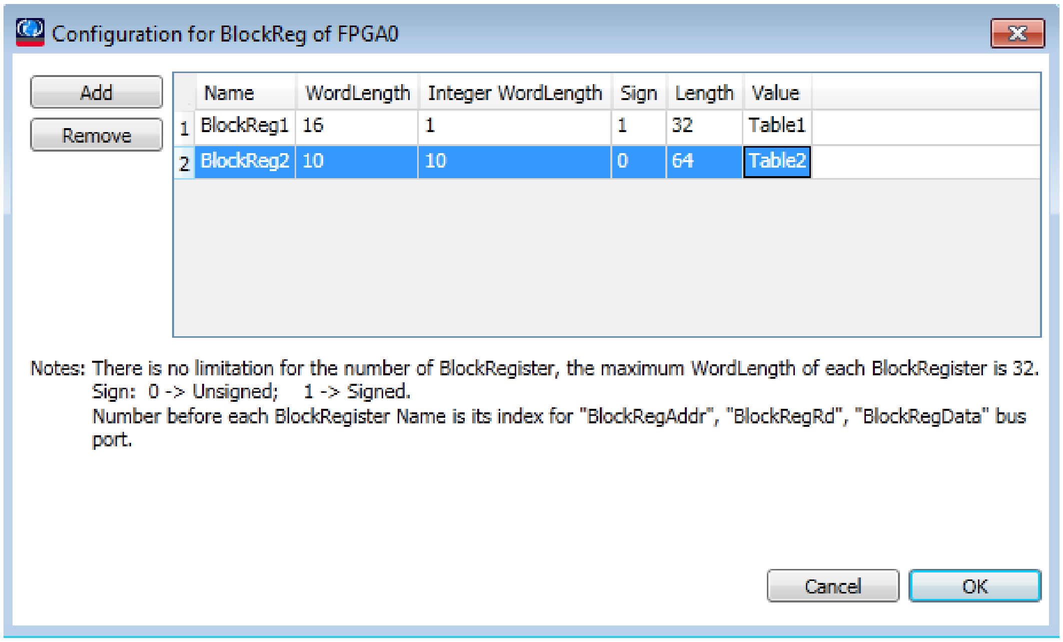
Configuration Mode: Configure Standalone

FPGA0 Settings

□ Registe ☑ BlockRegister

BlockRegister is similar as Register interface. It is a table of reserved fixed-point memories (A memory is a block of registers that have the same fixed-point data format. The block of registers can be accessed via address. So a memory is like a RAM.)

After you enable BlockRegister, you can click Config BlockRegister button to open a new GUI:



| | Name | WordLength | Integer WordLength | Sign | Length | Value |
|---|---|---|---|---|---|---|
| 1 | BlockReg1 | 16 | 1 | 1 | 32 | Table1 |
| 2 | BlockReg2 | 10 | 10 | 0 | 64 | Table2 |

Notes: There is no limitation for the number of BlockRegister, the maximum WordLength of each BlockRegister is 32.
Sign: 0 -> Unsigned; 1 -> Signed.
Number before each BlockRegister Name is its index for "BlockRegAddr", "BlockRegRd", "BlockRegData" bus port.

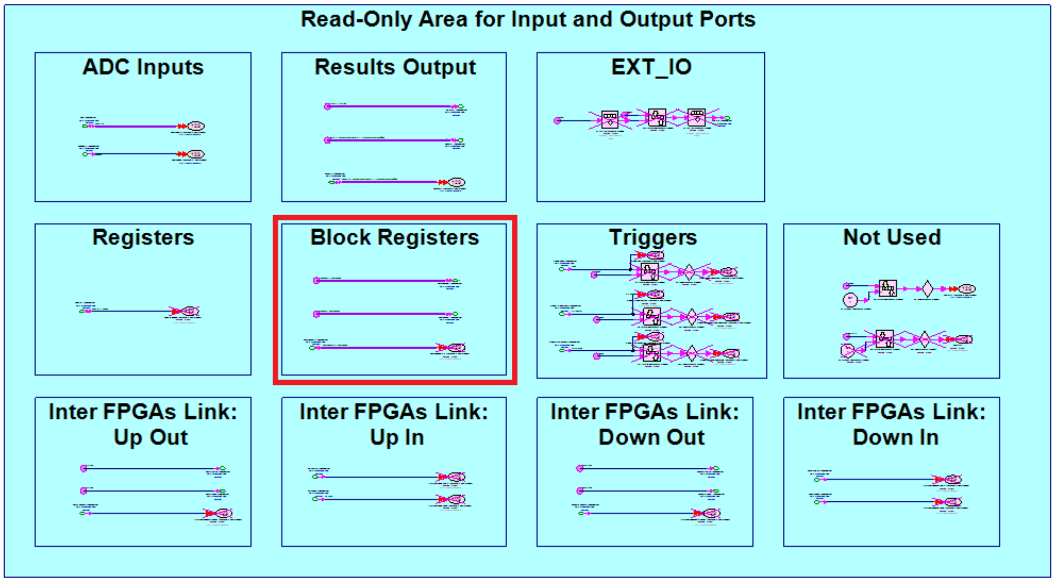
You can define the number of BlockRegister and the fixed-point data format of each BlockRegister in this GUI. You can also specify Length and Value for each defined BlockRegister, Length is the table size and Value is the values of all element of the table. Value must be a row vector whose size is equal to your specified Length value. Then the values of each BlockRegister will be set at the beginning of a simulation.

BlockRegister values can be re-configured before capturing ADC input data, and then the BlockRegister values are kept until the end of simulation. You can define the number of BlockRegisters and the fixed-point data format of each BlockRegister in GUI of M9703 Design Template. You can also specify the values for each defined BlockRegister, then the values of each BlockRegister will be initialized for simulation.

If you generate M9703 FPGA programming file for a M9703 Design Template with defined BlockRegisters, the number of BlockRegisters and the fixed-point data format of each BlockRegister are fixed in FPGA implementation, but you can still re-configure the values of each BlockRegister. Then when you run SystemVue and M9703 Co-Simulation, all BlockRegister values will be re-configured before ADC data capture and kept until the end of the Co-Simulation.

Connect BlockRegister with your FPGA design in subnet M9703_FPGA0 ~ M9703_FPGA3:

In subnet M9703_FPGA*x* for user's application fixed-point design, BlockRegister ports have been configured in advance. If you define BlockRegister interface in the GUI of M9703 Design Template, you can use the BlockRegister ports for your application fixed-point design.



Because a BlockRegister is like a block of RAM, so it has the memory mapper IO ports. For a BlockRegister, its memory mapped IO ports are:

BlockRegAddr: it is an output bus port of user's application design. Users can use this port to output the address to look up the corresponding value. The fixed-point data format of this port must be:

Word length = ceil( log2(Length of BlockRegister) )
Integer word length = ceil( log2(Length of BlockRegister) )
Unsigned
Length of BlockRegister is defined in the GUI of M9703 Design Template.

BlockRegRd: it is an output bus port of user's application design. Users can use this port to output "Read Enable". When the "Read Enable" is high, the look-up value at the current address will be valid at the next simulation sample.

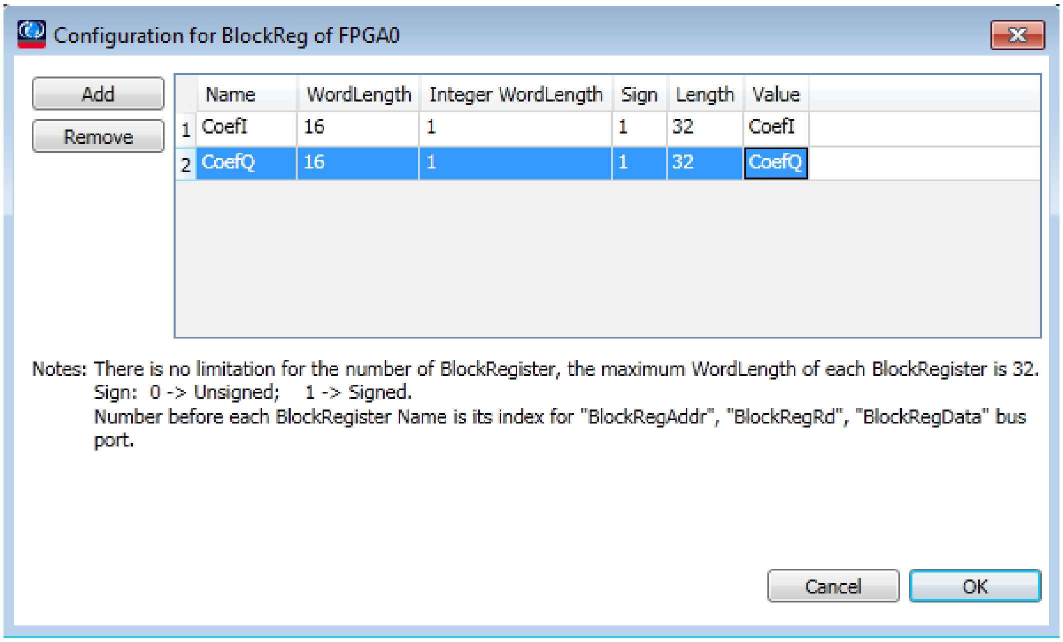
The fixed-point data format of this port must be:
Word length = Integer word length = 1
Unsigned

BlockRegData: it is an input bus port of user's application design. Users can get the look-up value from this port. Note that which input samples of this port are valid depends on output port BlockRegRd.

The fixed-point data format of this port is defined in GUI of M9703 Design Template.

Because you can define multiple BlockRegister in M9703 Design Template GUI, above three ports are all bus ports. Assuming that you defined *N* BlockRegister, the BlockRegister interface in subnet M9703_FPGA*x* are three bus ports: BlockRegAddr(*1:N*), BlockRegRd(*1:N*) and BlockRegData(*1:N*). You can use net labels BlockRegAddr(*x*), BlockRegRd(*x*) and BlockRegData(*x*) to access one of the defined BlockRegister.

For example, if you have below BlockRegister definition in the GUI:



|  | Name | WordLength | Integer WordLength | Sign | Length | Value |
|---|---|---|---|---|---|---|
| 1 | CoefI | 16 | 1 | 1 | 32 | CoefI |
| 2 | CoefQ | 16 | 1 | 1 | 32 | CoefQ |

Notes: There is no limitation for the number of BlockRegister, the maximum WordLength of each BlockRegister is 32.
Sign: 0 -> Unsigned; 1 -> Signed.
Number before each BlockRegister Name is its index for "BlockRegAddr", "BlockRegRd", "BlockRegData" bus port.

Assuming that the two BlockRegister are defined for two coefficients re-configurable FIR filter in FPGA0, the length is 32 for both of them.

Then in subnet M9703_FPGA0, you can use the BlockRegister like:

- Connect the "CoefI" address output port of your design to the line with net label BlockRegAddr(1).
- Connect the "CoefI" read enable output port of your design to the line with net label BlockRegRd(1).
- Connect the "CoefI" look-up value input port of your design to the line with net label BlockRegData(1).
- Connect the "CoefQ" address output port of your design to the line with net label BlockRegAddr(2).
- Connect the "CoefQ" read enable output port of your design to the line with net label BlockRegRd(2).
- Connect the "CoefQ" look-up value input port of your design to the line with net label BlockRegData(2).

The fixed point data format of BlockRegAddr(1) and BlockRegAddr(2) must be:

- Word length = Integer word length = 5 (because the length of BlockRegister is defined as 32)
- Unsigned

The fixed point data format of BlockRegRd(1) and BlockRegRd(2) input is:

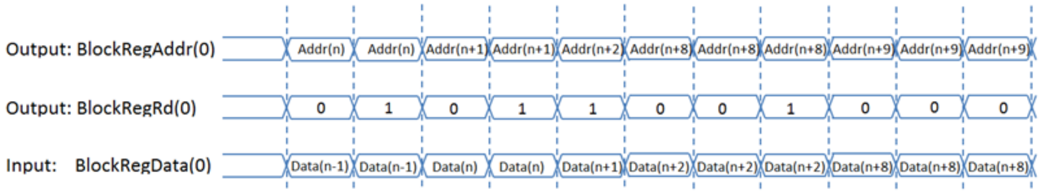
- Word length = 1
- Integer word length = 1
- Unsigned

The fixed point data format of BlockRegData(1) and BlockRegData(2) input is:

- Word length = 16
- Integer word length = 1
- Signed

As defined in GUI.

The timing of BlockRegister interface is shown below:



In users' application design M9703_FPGA*x* subnet, you can generate BlockRegAddr and BlockRegRd output and read back BlockRegData. When you output BlockRegRd as 0, BlockRegData will hold the current value at the next clock cycle; While you output BlockRegRd as 1, BlockRegData will be updated to the look-up value of the current address at the next clock cycle.

NOTE **Net Label Key Words For BlockRegister**

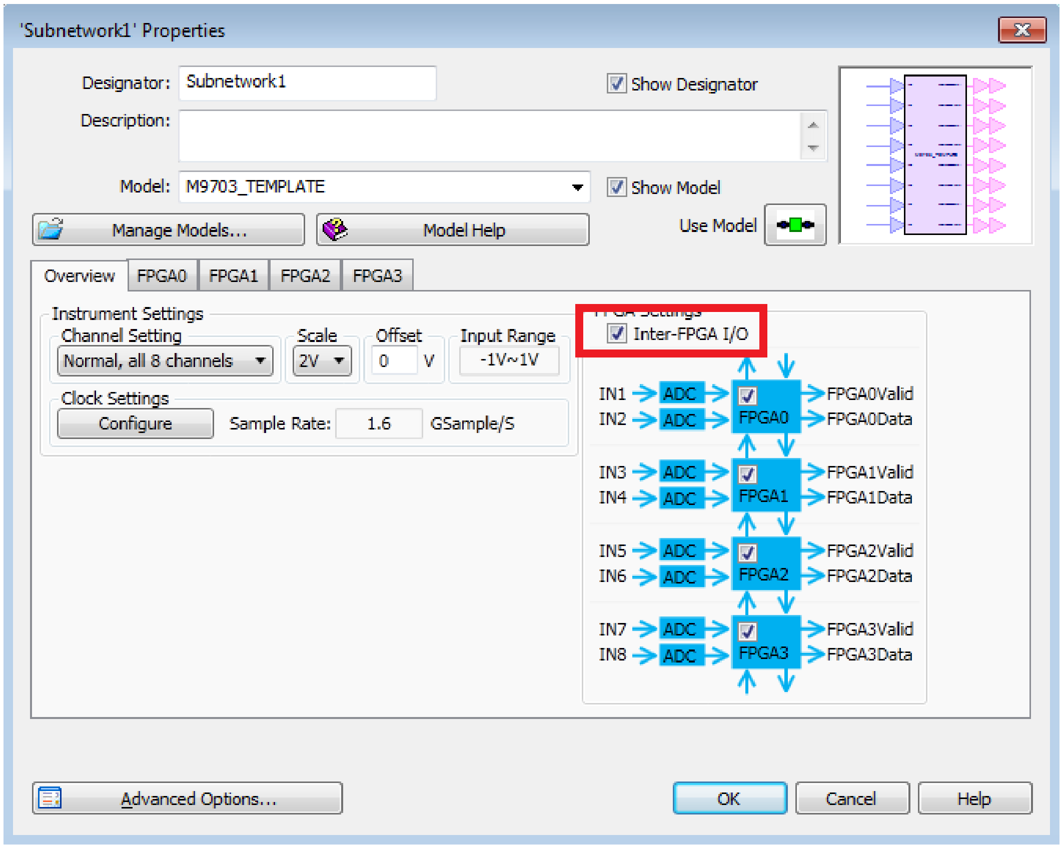
- **BlockRegAddr(*x*)**
- **BlockRegRd(*x*)**
- **BlockRegData(*x*)**
- where ***x*** is a number between **1 to N** (**N** is the number of your definedBlockRegister).

### Inter-FPGA I/O

As shown in Overview of M9703 High-Speed Digitizer, there are dedicated high speed data links between four FPGAs on a M9703 digitizer. Between each two FPGAs, there are independent bi-direction data links. The effective throughput of each direction is about 1GB/s.

Configure Inter-FPGA I/O in M9703 Design Template GUI:

In Overview tab of M9703 Design Template GUI, there is a checkbox Inter-FPGA I/O . You can check it to enable Inter-FPGA I/O or uncheck it to disable Inter-FPGA I/O. Please note that when you enable or disable Inter-FPGA I/O, all Inter-FPGA I/O between all four FPGAs are all enabled or disabled. The diagram can also show the Inter-FPGA I/O status (blue for enabled; grey for disabled).



'Subnetwork1' Properties

Designator: Subnetwork1 ☑ Show Designator

Description:

Model: M9703_TEMPLATE ☑ Show Model

Use Model

Overview | FPGA0 | FPGA1 | FPGA2 | FPGA3

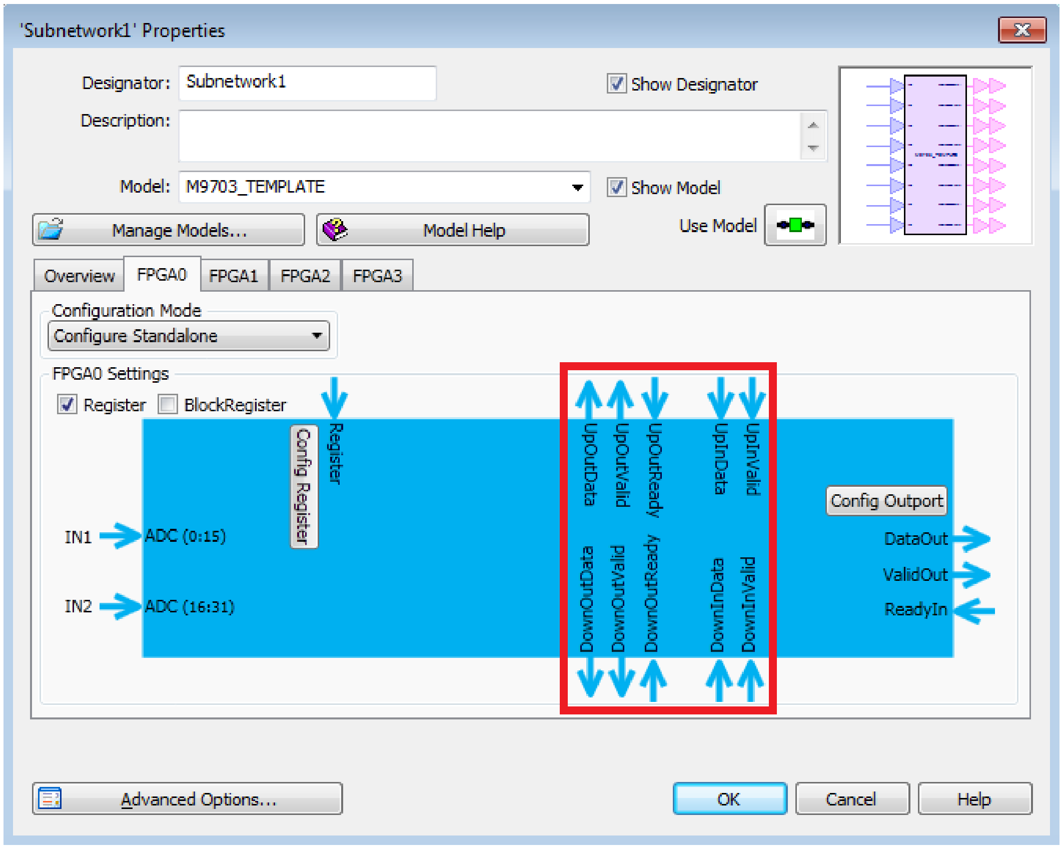
Instrument Settings

Channel Setting: Normal, all 8 channels; Scale: 2V; Offset: 0 V; Input Range: -1V~1V

Clock Settings: Sample Rate: 1.6 GSample/S

FPGA Settings: ☑ Inter-FPGA I/O

When Inter-FPGA I/O is enabled, in all enabled FPGA tabs, Inter-FPGA I/O interface is shown in their diagrams.



'Subnetwork1' Properties

Designator: Subnetwork1 ☑ Show Designator

Description:

Model: M9703_TEMPLATE ☑ Show Model

Use Model

Overview | FPGA0 | FPGA1 | FPGA2 | FPGA3

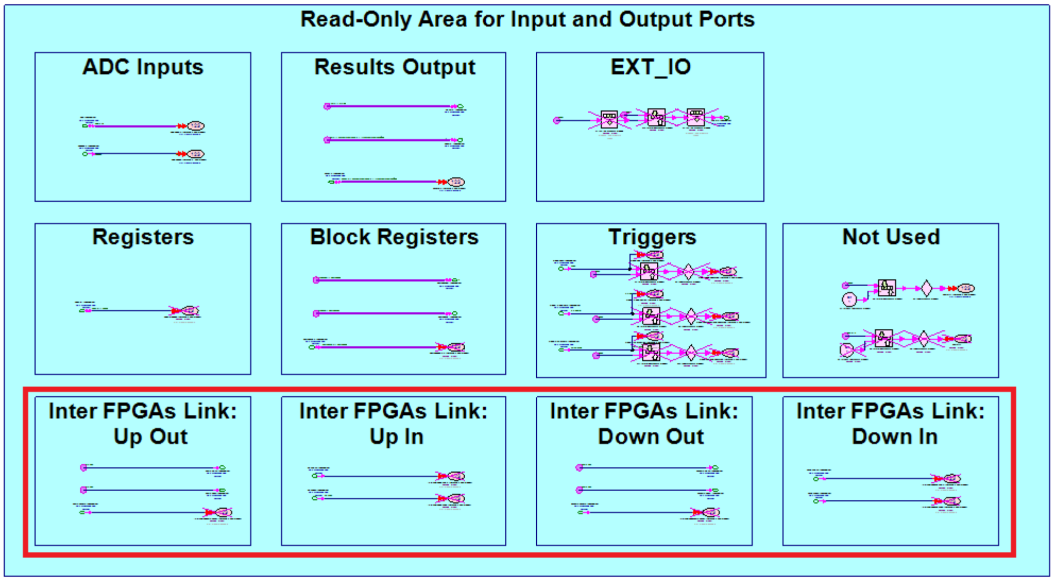
Configuration Mode: Configure Standalone

FPGA0 Settings: ☑ Register ☐ BlockRegister

Connect Inter-FPGA I/O with your FPGA design in subnet M9703_FPGA0 ~ M9703_FPGA3:

For each DPU FPGA, it has four independent Inter-FPGA interface – UpIn, UpOut, DownIn and DownOut. For UpOut and DownOut interfaces, they are the same as OutPort interface. It has data and valid output and a ready intput. For UpOut, its interface is output ports UpOutData and UpOutValid and input port UpOutReady. For DownOut, its interface is output ports DownOutData and DownOutValid and input port DownOutReady.

For UpIn and DownIn interfaces, they just have data and valid input ports and they have NOT backward ready output. For UpIn, its interface is input ports UpInData and UpInValid. For DownIn, its interface is output ports DownInData and DownInValid.



Because the production and consumption of inter-FPGA data stream are both designed by users, users know how to process the received data stream. So they don't need generate a ready signal backward to data production side to control the data stream. In this way, the inter-FPGA interface is simpler and the latency of inter-FPGA data transfer is minimum. For most real-time signal processing, the low latency between FPGAs can make design simpler and consume less FPGA hardware resources.

All ports of Inter-FPGA I/O are NOT bus ports. The data bit width between FPGA is 64 bits. In order to make the latency between FPGAs minimum, we don't add any arbitration for multiple data streams. So if you have multiple data streams, you need concatenate them as 64 bits format in your FPGA design.

In subnet M9703_FPGA0 ~ M9703_FPGA3, the fixed-point data format of UpInData, UpOutData, DownInData and DownOutData ports must be:

- WordLength = 64
- Integer WordLength = 64
- Unsigned

In subnet M9703_FPGA0 ~ M9703_FPGA3, the fixed-point data format of UpInValid , DownInValid, UpOutValid, UpOutReady, DownOutValid and DowOutReady ports must be:

- WordLength = 1
- Integer WordLength = 1
- Unsigned

If the fixed-point data format in your FPGA design doesn't match the required fixed-point data format, when you run a software simulation, SystemVue will post an error.

Inter-FPGA data stream transfer behavior:

The latency between two FPGAs varies between 70~80 clock cycles, which means transmitted FPGA send a valid data and 70~80 clock cycles later the receiving FPGA can receive the data. In M9703 Design Template software simulation behavior, we set the latency as maximum 80 clock cycles. You have to consider the latency effect in your own FPGA design.

The bandwidth between two FPGAs is about 1GB/s. It's a average bandwidth, the instant bandwidth may variable slightly. So you need add a small FIFO on consumption sides of inter-FPGA data stream. Usually, a depth 32 FIFO is enough for removing the bandwidth variation effect. If your FPGA working frequency is 200MHz (-SR2 option), then 1GB/s bandwidth means that you can only send 5 samples every 8 clock cycles. On transmitted side, you have to buffer the data with a FIFO according to ready input signal.

For example, you want to send FPGA0 data to FPGA1 and align with data stream in FPGA1, then you need add a FIFO to removing the effect of latency and bandwidth variation on receiving side FPGA1, considering latency 80 cycles and bandwidth variation 32 cycles, you may select a FIFO with depth 128 to make your inter-FPGA data transfer work safely. On transmitted side FPGA0, you also need add a FIFO, the FIFO can only be read when ready input is high. The depth of FIFO can be decided according to your actual transfer throughput and data transfer length.

NOTE

When **Inter-FPGA I/O** is enabled, all four directions of inter-FPGA interface **UpIn, UpOut, DownIn and DownOut** are all enabled. Then in your FPGA design subnets, you have to generate values for all output ports – **UpOutData, UpValid, DownOutData and DownOutValid**. If you don't use any of them, you also need to connect some dummy values (for example, constant 0 with <64, 64, unsigned> fixed-point format for data output and constant 0 with <1,1, unsigned> fixed-point format for valid output) for these output ports.

**Net Label Key Words For Inter-FPGA I/O**

For **UpOut** direction:

- **UpOutData** (output port)
- **UpOutValid** (output port)
- **UpOutReady** (input port)

For **UpIn** direction:

- **UpInData** (input port)
- **UpInValid** (input port)

For **DownOut** direction:

- **DownOutData** (output port)
- **DownOutValid** (output port)
- **DownOutReady** (input port)

For **DownIn** direction:

- **DownInData** (input port)
- **DownInValid** (input port)

### Trigger

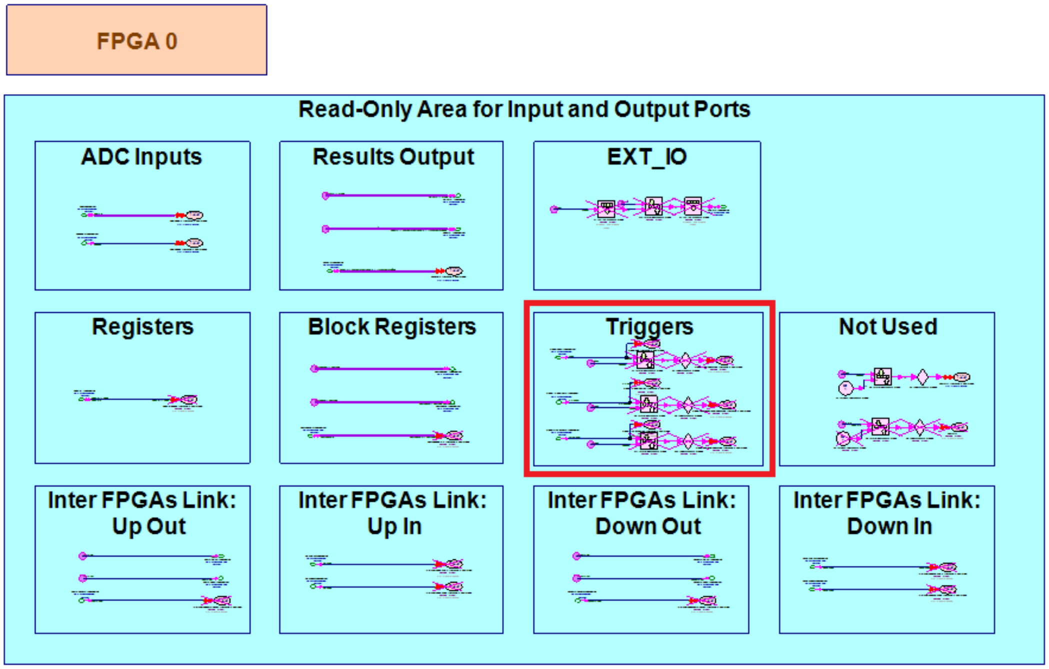
In M9703 digitizer, each DPU FPGA has trigger input signals from control FPGA. Users can configure the trigger source, and then once the trigger condition is met, the trigger signals will reach to each DPU FPGA at the same time. Users can use the input trigger signals on each DPU FPGA to assist their own real-time signal processing. In SystemVue M9703 FPGA flow, we provide design, simulation, FPGA implementation and digitizer hardware control for trigger.

Trigger Interface on DPU FPGA

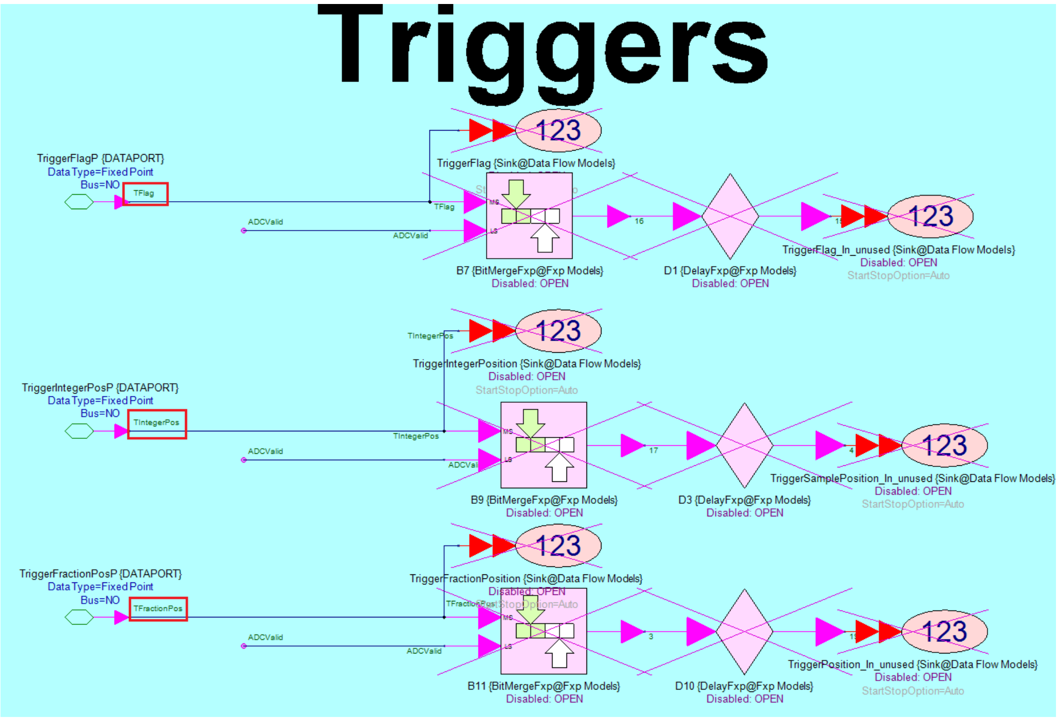
The Trigger interface on each DPU FPGA are 3 input fixed-point signals:

- TriggerFlag: It represents whether it's triggered on the current FPGA system clock cycle. When it's 1, it represents that it's triggered on the current FPGA system clock cycle; When it's 0, it represents that it's not triggered on the current FPGA system clock cycle. Its fixed-point format is <WordLength = 1, IntegerWordLength = 1, Unsigned>.
- TriggerIntegerPos: It represents the ADC sample index before the trigger instant. Because the ADC samples are inputted to DPU FPGA parallel, on each FPGA system clock cycle, there are NumParallel ADC samples are inputted (NumParallel=16 for normal sampling mode and NumParallel=32 for interleaved sampling model). And also the block of the input ADC samples hold for two FPGA system clock cycles and ADCValid input signal is used to represent whether the block of ADC samples are valid on the current FPGA system clock cycles. So TriggerIntegerPos will be an integer between 0 and NumParallel-1. Its fixed-pointed format is <WordLength=7, IntegerWordLength=7, Unsigned>.
- TriggerFractionPos: It represents the precise trigger position between the trigger instant and the next ADC sample. It's a value in the range [0, 1). Its fixed-pointed format is <WordLength=32, IntegerWordLength=0, Unsigned>.

In each DPU FPGA design subnet M9703_FPGA*x*, the 3 Trigger input ports are pre-configured.



If you configure to use the trigger interface in your DPU FPGA in M9703 Design Template GUI, you can connect the 3 input ports to your FPGA design using the pre-configured Net Labels: TFlag, TIntegerPos and TFractionPos as shown below.



**NOTE** Only when you select to use Trigger interface in M9703 Design Template GUI, the 3 input ports are enabled and you can connect them to your design. (You can also leave them open even you enable the trigger ports.)

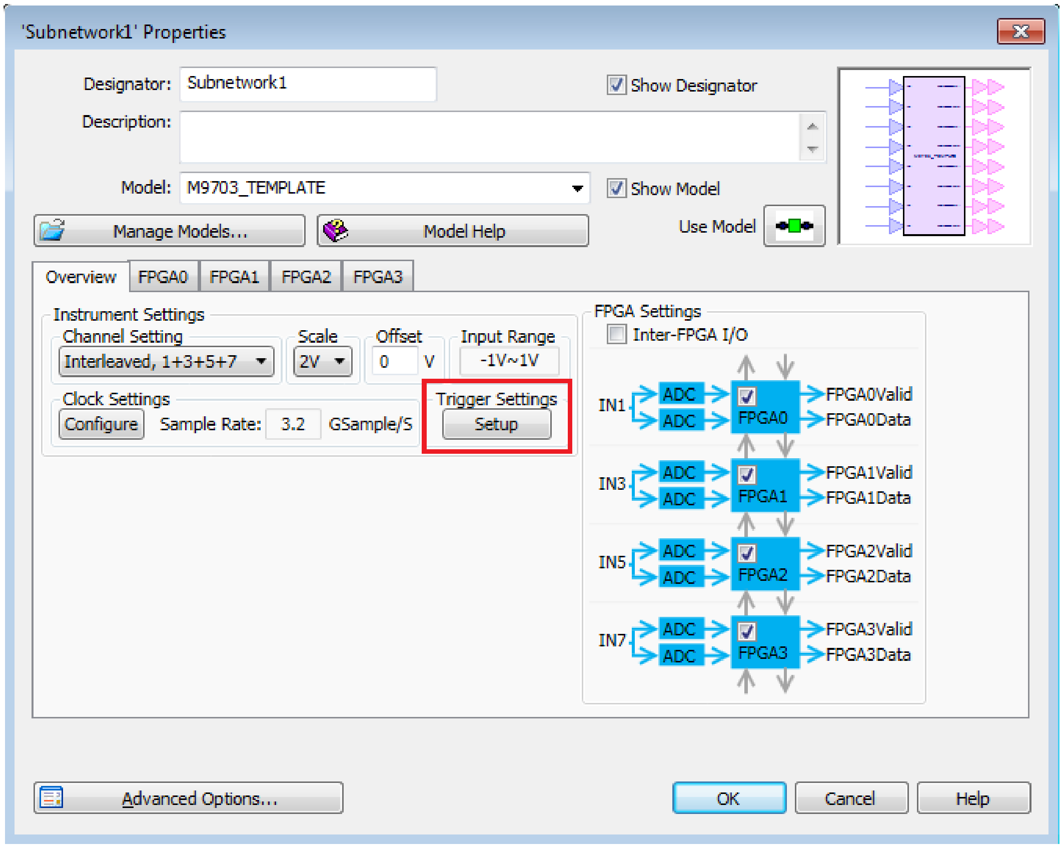
Otherwise, if you configure to not use Trigger interface, the 3 input ports are disabled. You can't connect them to your design.

**Net Label Key Words For Trigger:**

- **TFlag**
- **TIntegerPos**
- **TFractionPos**

Trigger Simulation in M9703 Design Template

In Overview tab of M9703 Design Template GUI, there is a button for Trigger Settings.



'Subnetwork1' Properties

Designator: Subnetwork1

☑ Show Designator

Description:

Model: M9703_TEMPLATE

☑ Show Model

Use Model

Instrument Settings

Channel Setting: Interleaved, 1+3+5+7

Scale: 2V

Offset: 0 V

Input Range: -1V~1V

Clock Settings: Configure; Sample Rate: 3.2 GSample/S

Trigger Settings: Setup

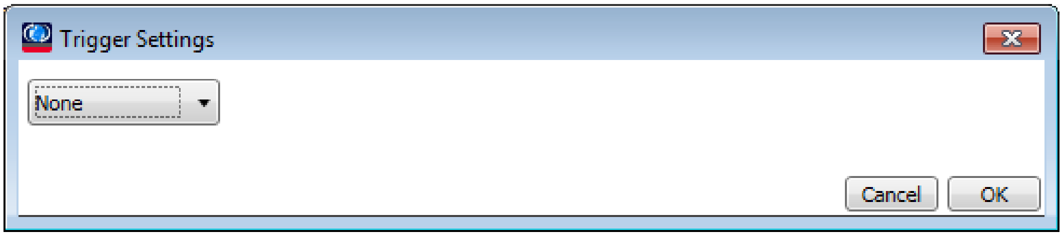
FPGA Settings

□ Inter-FPGA I/O

You can click it to open a new GUI to configure the trigger mode for M9703 FPGA design and simulation.

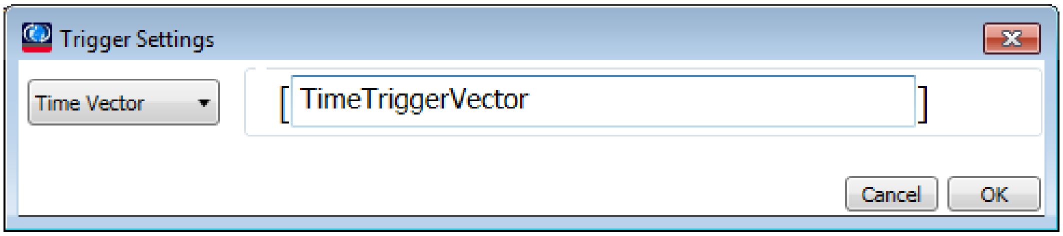
You can select to not to use trigger interface or use the trigger interface and specify the triggers time in a time vector.

None



Trigger interface is not used. The 3 trigger input ports are disabled in M9703_FPGA*x* subnet. You can't connect them with your FPGA design.

Time Vector



[ TimeTriggerVector ]

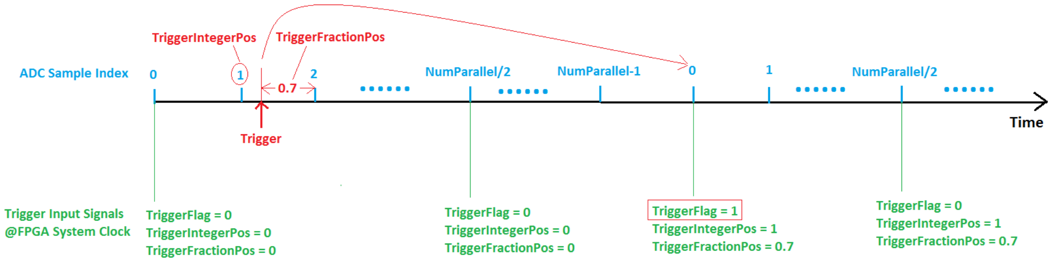
You specify a time vector. On each time point of the time vector, a trigger will be generated. This trigger mode is used to generate triggers at any desired instants. Then you can simulate whether your FPGA design works well with your specified triggers.

A trigger can arrive on any instant, below figure shows the 3 trigger input ports behavior on DPU FPGA when a trigger arrives.

For example, a trigger arrives at 1.3 * (ADC sampling period). The ADC sample index before the trigger instant is 1, so TriggerIntegerPos will be 1 for this trigger. And the ADC sampling period normalized time from trigger instant to the next ADC sample is 0.7, so TriggerFractionPos will be 0.7 for this trigger.

The 3 trigger input ports work at FPGA system clock. And FPGA system clock period is NumParallel/2 times of ADC sampling period. So for each block of NumParallel ADC parallel input samples, there are 2 FPGA system clock cycles. When a trigger arrives, the 3 trigger input signal values will be valid from the next block of NumParallel ADC input samples. And TriggerFlag is 1 for only one FPGA system clock cycles. The values of TriggerIntegerPos and TriggerFractionPos will hold until the next trigger.

When a trigger arrives exactly on an ADC sample position, this ADC sample's index will be TriggerIntegerPos and TriggerFractionPos will be 0. For example, a trigger arrives at 2 * (ADC sampling period). Its TriggerIntegerPos will be 2 and its TriggerFractionPos will be 0.



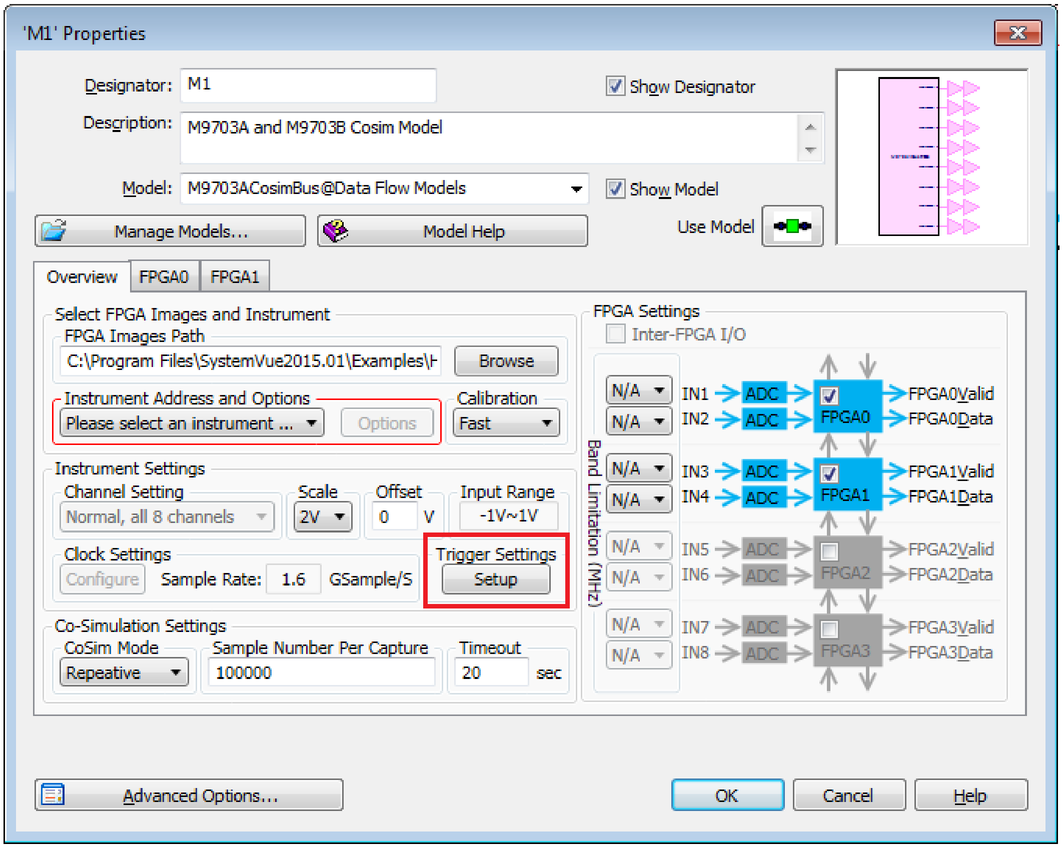
**NOTE** Corresponding to M9703 hardware behavior, if a trigger is too close to its previous trigger, it can't be generated.

So the rule is that when a trigger is generated, the next trigger has to be generated after 40 FPGA system clock cycles. All triggers that are specified within the 40 FPGA system clock cycles will be ignored.

In **M9703_FPGA*x*** design subnet, the parameter **FPGAWorkingCLKFreq** is the FPGA system clock frequency.

## Trigger Configuration for M9703 Hardware Co-Simulation

In M9703ACosimBus model GUI, there is a Trigger Settings button. You can click it to open a new GUI to configure the actual trigger source for M9703 hardware co-simulation.



'M1' Properties

Designator: M1

☑ Show Designator

Description: M9703A and M9703B Cosim Model

Model: M9703ACosimBus@Data Flow Models

☑ Show Model

Use Model

Overview | FPGA0 | FPGA1

Select FPGA Images and Instrument

FPGA Images Path: C:\Program Files\SystemVue2015.01\Examples\H

Instrument Address and Options: Please select an instrument ...

Calibration: Fast

Instrument Settings

Channel Setting: Normal, all 8 channels

Scale: 2V

Offset: 0 V

Input Range: -1V~1V

Clock Settings

Sample Rate: 1.6 GSample/S

Trigger Settings: Setup

Co-Simulation Settings

CoSim Mode: Repeative

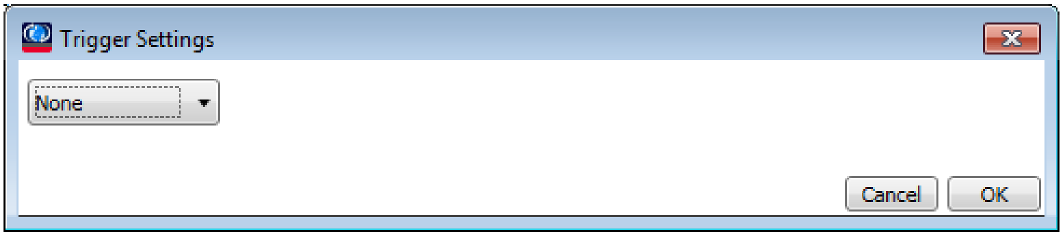
Sample Number Per Capture: 100000

Timeout: 20 sec

FPGA Settings

□ Inter-FPGA I/O

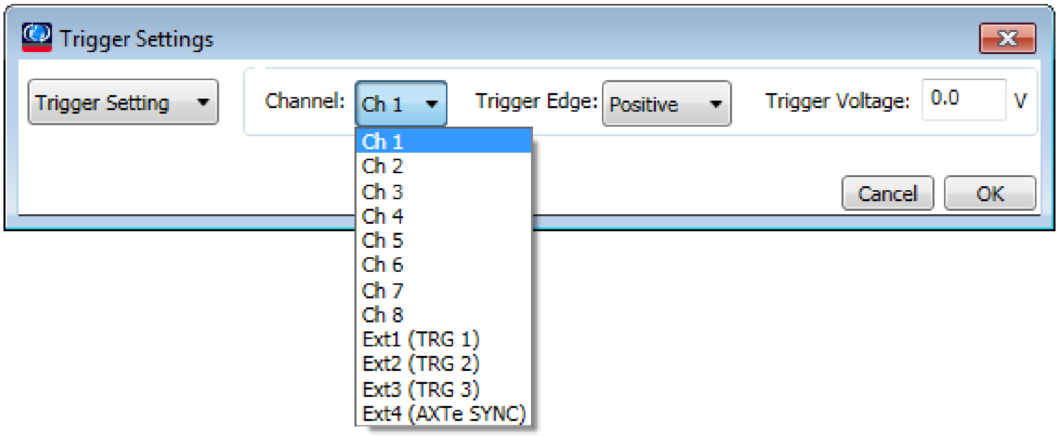
You can select None to disable the actual trigger source. Then TriggerFlag input to DPU FPGA will be 0.



Trigger Settings

None

You can select Trigger Setting to configure the actual trigger source.



Trigger Settings

Trigger Setting

Channel: Ch 1

Trigger Edge: Positive

Trigger Voltage: 0.0 V

The actual trigger source can be all enabled channels and External Trigger source. External Trigger 1~3 are corresponding to the 3 Trigger Input ports on M9703 digitizer front panel. External Trigger 4 is from AXIe chassis back plane and is reserved for multi-modules synchronization.

Trigger Edge:

The Trigger Edge defines which one of the two possible transitions will be used to initiate the trigger when it passes through the specified Trigger Voltage. Positive slope indicates that the signal is transitioning from a lower voltage to a higher voltage. Negative slope indicates the signal is transitioning from a higher voltage to a lower voltage.

Trigger Voltage:

The Trigger Voltage specifies the voltage at which the selected trigger source will produce a valid trigger. All trigger circuits have sensitivity levels that must be exceeded in order for reliable triggering to occur.

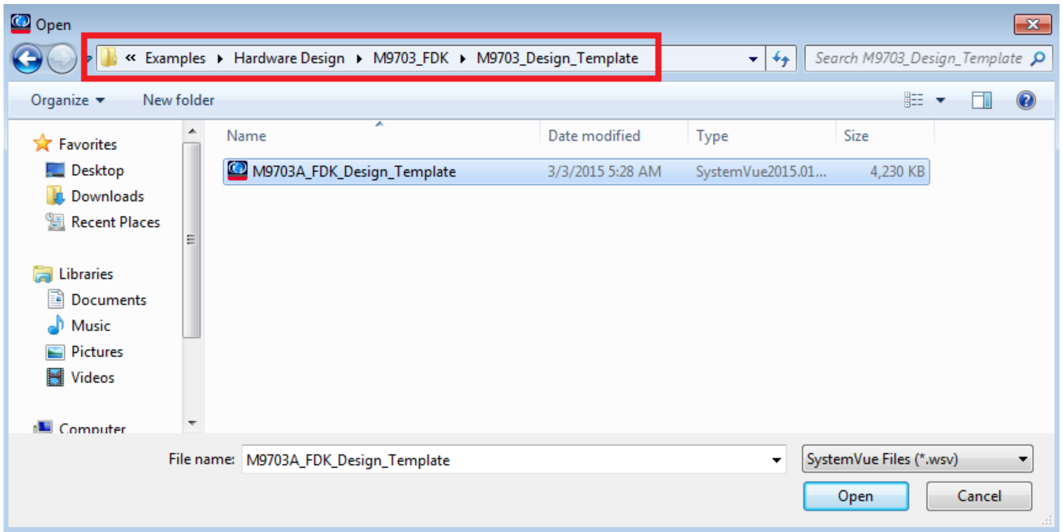
Both the external trigger input and channel triggers have a hysteresis of 5% of Full Scale Range -- The span of the voltage input of the Digitizer (negative to positive) including the configured offset voltage.

On external trigger, the Full Scale Range is ±5 V, therefore the digitizer will trigger on signals with a peak-to-peak amplitude > 0.5 V. The input range of Trigger Voltage is ±5 V for Ext1~Ext4.

When using the channel triggers, the trigger level must be set within Offset ± Scale . For example, when Offset is 0 V and Scale is 2 V, the input range of all enable channels is ±2 V.

### M9703 Design Template GUI

You can find M9703 Design Template from SystemVue Example workspace M9703_FDK_Design_Template.wsv.

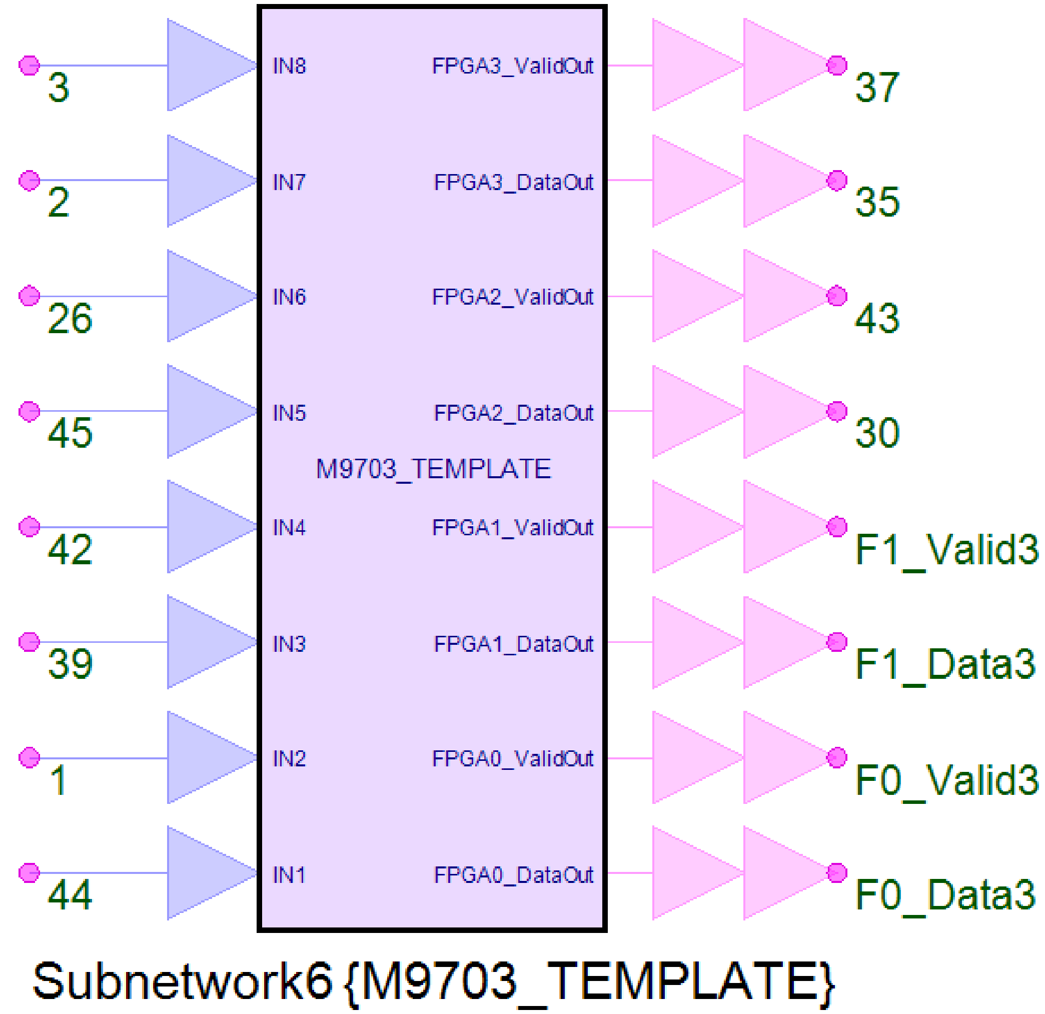


| Name | Date modified | Type | Size |
|---|---|---|---|
| M9703A_FDK_Design_Template | 3/3/2015 5:28 AM | SystemVue2015.01... | 4,230 KB |

File name: M9703A_FDK_Design_Template

SystemVue Files (*.wsv)

You can find subnet M9703_TEMPLATE (model) in SystemVue Workspace Tree of this example and drag-and-drop it to a SystemVue top level schematic. Then you will see the M9703 design template model as show below:



Input Ports:

It has 8 input ports that are all floating point (real) data type input. The input ports IN1 ~ IN8 are corresponding to the input ports on the front panel of M9703 instrument. Because the M9703 design template is used for software simulation, you need create input waveforms for the input ports of M9703 design template in SystemVue. Then you can simulate your M9703 FPGA designs with your input waveforms.

You can configure the clock source and sample mode in the GUI of M9703 Design Template. Then the M9703's sample rate can be displayed in the GUI. It supports both internal and external clock modes and also interleaved channel mode. When you run simulation, SystemVue will check the sample rates of input waveform and M9703. If they are not the same, the simulation will post an error.

These input ports are all optional input ports, which means you need not connect all of them. For example, you just want to use FPGA0 of M9703 in Normal, 8 channels mode, then you can just create input waveforms and connect IN1 and IN2 and leave the other input ports open. If you just want to use FPGA0 of M9703 in Interleaved, 1+3+5+7 mode, then you can just create input waveforms and connect IN1 and leave the other input ports open.

NOTE For all enabled input ports, the SystemVue will check its connectivity when running simulation. For example, if you enable FPGA0 in **Normal, 8 channels** mode, it means you enable **IN1** and **IN2**. Then you have to provide input waveforms for both **IN1** and **IN2**. If you leave one of them open, SystemVue will post an error when you run the simulation.

Output Ports:

It has 8 bus output ports. For each FPGA*x* in FPGA0 ~ FPGA3, it has a pair of bus output ports, FPGA*x*_DataOut *and* FPGA*x*_ValidOut.

The bus widths of FPGA*x*_DataOut and FPGA*x*_ValidOut:

The custom real-time processing results are outputted through OutPort in M9703 Design Template subnet M9703_FPGA0~M9703_FPGA3. OutPorts are defined in the GUI of M9703 Design Template. You can define OutPorts as two modes: Non-SameSampleRate mode and SameSampleRate mode.

For Non-SameSampleRate mode, the bus widths of FPGA*x*_DataOut and FPGA*x* _ValidOut are the same and are equivalent to the OutPort number that you defined in M9703 Design Template GUI. For example, you define two OutPort in M9703 Design Template GUI for FPGA0. Then FPGA0_DataOut(1) is the output data of your first OutPort of FPGA0 and FPGA0_ValidOut(1) is the corresponding valid indicator of FPGA0_DataOut(1). When FPGA0_ValidOut(1) is 1, it indicates the corresponding data sample of FPGA0_DataOut(1) is valid. Similarly, FPGA0_DataOut (2) and FPGA0_ValidOut(2) are the output pair of the second OutPort of FPGA0.

For SameSampleRate mode, the bus widths of FPGA*x* DataOut are equivalent to the OutPort number that you defined in M9703 Design Template GUI. The bus widths of FPGA*x*_ValidOut is 1. All OutPorts' DataOut share the same ValidOut. For example, you define two OutPorts in M9703 Design Template GUI for FPGA0. Then FPGA0_DataOut(1) is the output data of your first OutPort of FPGA0 and

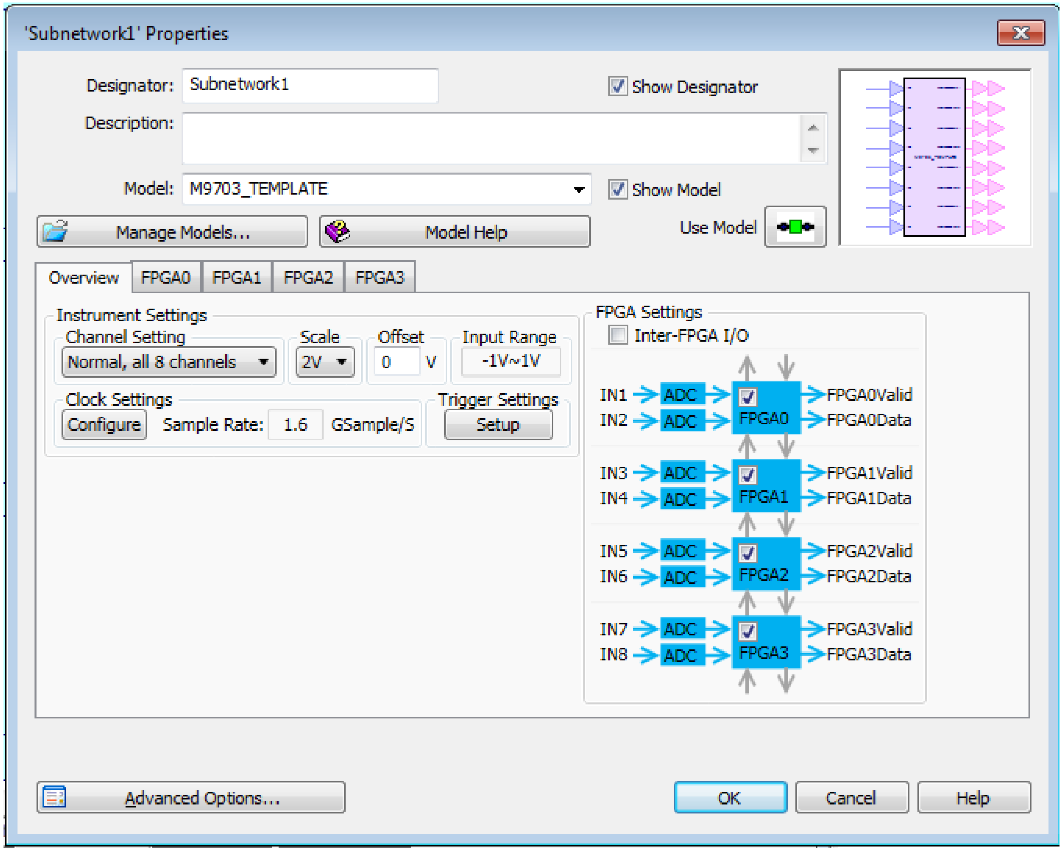
FPGA0_DataOut(2) is the output data of your second OutPort of FPGA0. The output of FPGA0_ValidOut is the valid indicator of both FPGA0_DataOut(1) and FPGA0_DataOut(2). When FPGA0_ValidOut is 1, it indicates the corresponding data sample of FPGA0_DataOut(1) and FPGA0_DataOut(2) are valid.

All output ports are SystemVue fixed-point data type. The fixed-point data format (wordlength, integer wordlength and signed) of FPGA*x*_DataOut is defined in M9703 Design Template GUI. The FPGA*x*_ValidOut is one-bit logic output, so its wordlength and integer wordlength are 1 and unsigned.

All output pair of FPGA*x*_DataOut and FPGA*x*_ValidOut are optional. For example, if you just use FPGA0, there will not be output signal from FPGA*x*_DataOut and FPGA*x* _ValidOut of FPGA1 ~ FPGA3.

After drag-and-drop M9703_TEMPLATE subnet from workspace tree to a Schematic, you can double click the M9703 Design Template model in Schematic to open its GUI. You can use this GUI to configure the M9703 DPU FPGAs interfaces.

Overview Tab:



'Subnetwork1' Properties

Designator: Subnetwork1

☑ Show Designator

Description:

Model: M9703_TEMPLATE

☑ Show Model

Instrument Settings

Channel Setting: Normal, all 8 channels

Scale: 2V

Offset: 0 V

Input Range: -1V~1V

Clock Settings

Sample Rate: 1.6 GSample/S

Trigger Settings

FPGA Settings

□ Inter-FPGA I/O

The first tab of the GUI is Overview. It is used to configure the top level attributions of M9703 Design Template.

Channel Setting

It has three options:

- Normal, all 8 channels: Normal Mode, 8 channels

- Interleaved, 1+3+5+7: Interleaved Mode, 4 combined channels, sample rate x2
- Interleaved, 2+4+6+8 : Interleaved Mode, 4 combined channels, sample rate x2

Scale

It has two options: 1V and 2V. It's the full voltage range for input signal. This parameter in M9703 Design Template is used to mimic the ADC input full scale. There is a corresponding parameter for controlling the real M9703 instrument in M9703ACosimBus model. The input value exceeding this full scale will be limited to the maximum or minimum value of this scale.

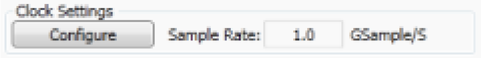
Offset

This parameter is used to mimic the offset adjust of M9703's input amplifier. There is a corresponding parameter for controlling the real M9703 instrument. The valid range is from -2*Scale to 2*Scale.

Input Range

It is a read-only item to show the valid range of input signal based on specified Scale and Offset parameters.

Clock Settings

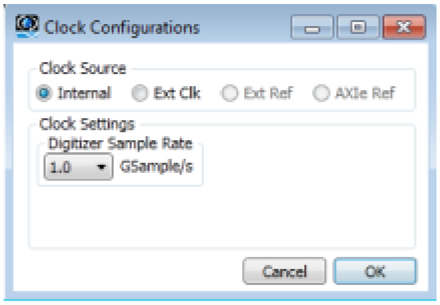


Sample Rate: 1.0 GSample/S

You can click Configure button to open a new GUI to set the clock and the final sample rate for your current configuration will be shown on the right side.

After clicking Configure button, a new GUI will be opened. You can select to use Internal or External (External Clock Source) clock mode.

When you select Internal clock mode, you have two clock options that can result in two different sample rates. When Channel Setting is set as Normal, all 8 channels, you can set the sample rate as 1.0GS/s for SR1 option or 1.6GS/s for SR2 option. While when Channel Setting is set as Interleaved, 1+3+5+7 or Interleaved, 2+4+6+8 , you can set the sample rate as 2.0GS/s for SR1 option or 3.2GS/s for SR2 option.



Clock Source: ● Internal ○ Ext Clk ○ Ext Ref ○ AXIe Ref

Clock Settings

Digitizer Sample Rate: 1.0 GSample/s

When you select Ext Clk clock mode, you can set the sample rate in a range. When you use an actual M9703 instrument, you need to provide an external clock input with the corresponding frequency.

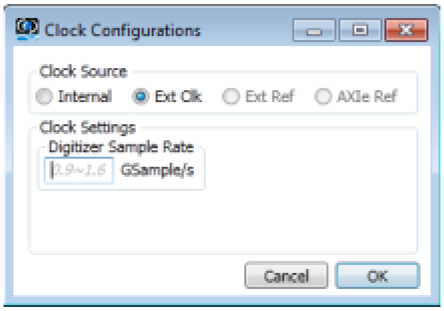
When Channel Setting is set as Normal, all 8 channels, the valid sample rate range is 0.9 ~ 1.6 GSample/s. But you need pay attention to the SR1/SR2 option of your M9703 instrument. If your M9703 instrument has SR1 option, the valid sample rate range is only 0.9 ~ 1.0GSample/s, while valid sample rate range is 0.9 ~ 1.6 GSample/s for SR2 option. The M9703 Design Template subnet is only used for FPGA design entry and software simulation, so you can use the full sample rate range, but when you use real M9703 instrument, you need pay attention to its SR1 /SR2 option.

In Normal, all 8 channels mode, the real external clock frequency that you need input to M9703 instrument is 2 times of the sample rate.

When Channel Setting is set as Interleaved, 1+3+5+7 or Interleaved, 2+4+6+8, the valid sample rate range is 1.8 ~ 3.2 GSample/s. But you need pay attention to the SR1/SR2 option of your M9703 instrument. If your M9703 instrument has SR1 option, the valid sample rate range is only 1.8 ~ 2.0GSample/s, while valid sample rate range is 1.8 ~ 3.2GSample/s for SR2 option. The M9703 Design Template subnet is only used for FPGA design entry and software simulation, so you can use the full sample rate range, but when you use real M9703 instrument, you need pay attention to its SR1/SR2 option.

In Interleaved, 1+3+5+7 or Interleaved, 2+4+6+8 mode, the real external clock frequency that you need input to M9703 instrument is the same as the sample rate.

The GUI gives you a hint for the valid range according to SR2 option case.


Clock Configurations

Clock Source

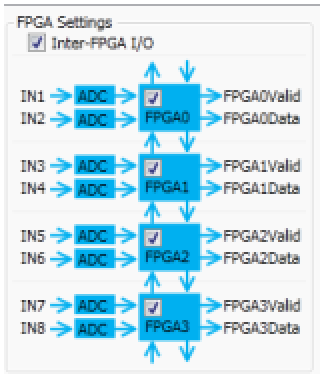
○ Internal ● Ext Clk ○ Ext Ref ○ AXIe Ref

Clock Settings

Digitizer Sample Rate: 0.9~1.6 GSample/s

The clock setting configuration doesn't affect the M9703 DPU FPGA programming file generation. So if your M9703 instrument is used with different clock setting that results in different sample rate. Your generated FPGA programming files also works. But a warning will be shown in the GUI of M9703CosimBus model to ask you to notice this difference, because it may cause different results in comparison to original software simulation.

FPGA Settings



FPGA Settings

☑ Inter-FPGA I/O

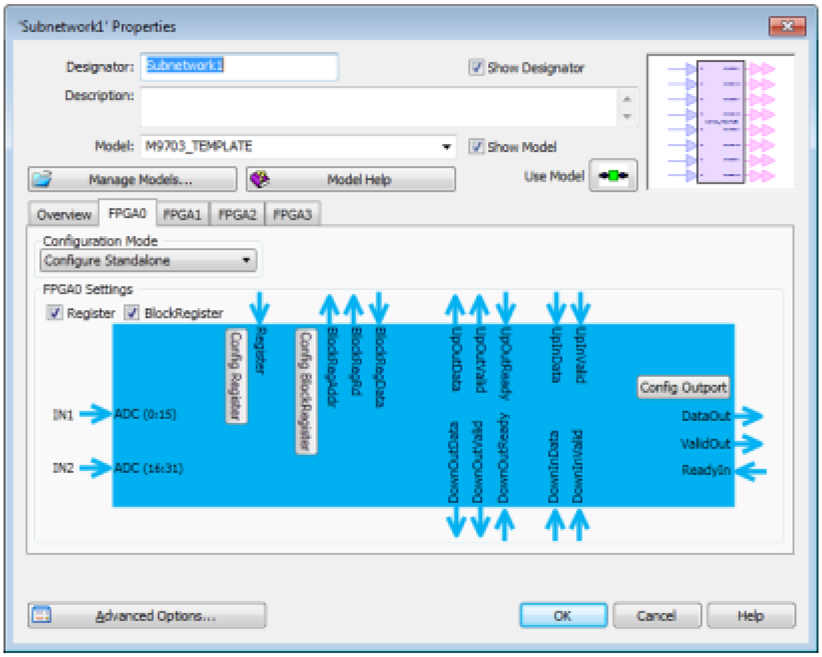
This figure shows your current DPU FPGAs configuration of M9703. You can select at least one DPU FPGA on this figure for FPGA design entry and software simulation. The input ports will also be adjusted on this figure according to the specified Channel Setting parameter.

You can also enable or disable Inter-FPGA I/O in this area. Each DPU FPGA has four Inter-FPGA I/O ports – UpOut, UpIn, DownOut and DownIn. If one DPU FPGA is selected and Inter-FPGA I/O is enabled, all of the four Inter-FPGA I/O ports of this DPU FPGA are enabled.

As shown in this area, FPGA0 that receives IN1 and IN2 is the uppermost DPU FPGA and FPGA3 that receive IN7 and IN8 is the lowermost DPU FPGA. So FPGA0's DownOut is connected with FPGA1's UpIn and FPGA0's DownIn is connected with FPGA1's UpOut. And so on, for the rest DPU FPGAs' Inter-FPGA I /O. So the four DPU FPGAs in one M9703 instrument can be connected in a chain.

The UpOut/UpIn of FPGA0 and DownOut/DownIn of FPGA3 are not used so far. They will be reserved to link with the adjacent M9703 instruments that are inserted in the same AXIe chassis as the current M9703 instrument in future.

FPGAx Tab:



'Subnetwork1' Properties

Designator: Subnetwork1 ☑ Show Designator

Description:

Model: M9703_TEMPLATE ☑ Show Model

Configuration Mode: Configure Standalone

FPGA0 Settings

☑ Register ☑ BlockRegister

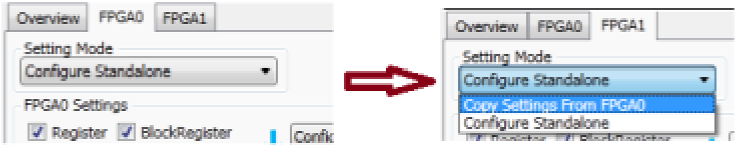
For each selected DPU FPGA in "FPGA Settings" in "Overview" tab, there will be a corresponding "FPGAx" tab in this M9703 Design Template GUI. And all tab UI of the selected FPGA are the same. In FPGAx tab, you can define FPGA interface ("Register", "BlockRegister" and "OutPort") for the current FPGA.

Configuration Mode

This parameter is used to specify how to configure the current FPGA interface, specify FPGA interface in the current GUI or copy interface configuration from another FPGA. The capability of copying from another FPGA setting will be useful when you want to apply the same FPGA interface settings for multiple FPGAs.

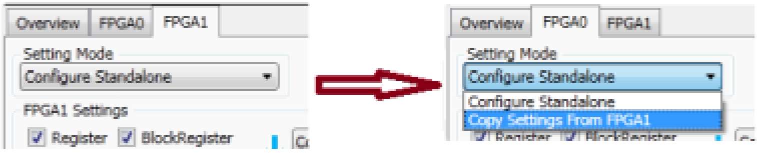
The default option is "Configure Standalone". When this option is selected, you use the current tab GUI to configure the corresponding FPGA interface. As long as you select "Configure Standalone" for FPGA0, you can find the option "Copy Settings From FPGA0" in the other FPGA tabs. And so on, for the rest FPGAs. It means that only the FPGA that is configured in its own tab GUI can be copied by the other FPGA tabs.

For example, if you enable FPGA0 and FPGA1. In FPGA0 tab, you set parameter "Configuration Mode" as "Configure Standalone" and configure FPGA0 interface in FPGA0 tab. Then in FPGA1 tab, you will have two options "Configure Standalone" and "Copy Settings From FPGA0" for parameter "Configuration Mode".


Setting Mode: Configure Standalone
FPGA0 Settings
☑ Register ☑ BlockRegister

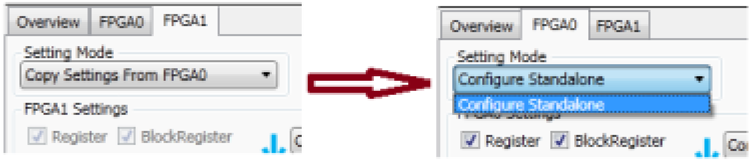
Setting Mode: Configure Standalone
Copy Settings From FPGA0
Configure Standalone

Similarly, if you select "Configure Standalone" in FPGA1 tab, then you will have two options "Configrue Standalone" and "Copy Settings From FPGA1" in FPGA0 tab.


Setting Mode: Configure Standalone
FPGA1 Settings
☑ Register ☑ BlockRegister

Setting Mode: Configure Standalone
Configure Standalone
Copy Settings From FPGA1

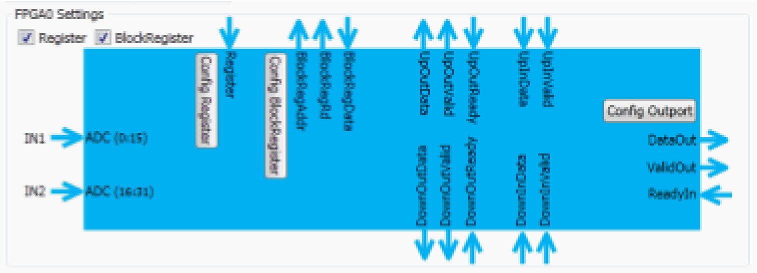
Otherwise, if you select "Copy Settings From FPGA0" in FPGA1 tab, then there will be no option "Copy Settings From FPGA1" in FPGA0 tab.


Setting Mode: Copy Settings From FPGA0
FPGA1 Settings
☑ Register ☑ BlockRegister

Setting Mode: Configure Standalone
Configure Standalone
☑ Register ☑ BlockRegister

FPGAx Settings

In FPGA*x* Settings area, a diagram of FPGA*x* interface is shown.



FPGA0 Settings

☑ Register ☑ BlockRegister

It shows the interface that you configure for FPGA*x*. The M9703_FPGAx subnet in that you design the custom real-time processing for DPU FPGA will have the corresponding I/O ports according to your configuration in this area. Each DPU FPGA has the following interface types:

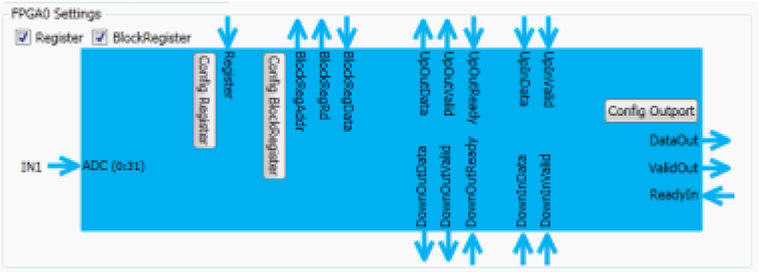
- ADC Input
- Register
- BlockRegister
- OutPort
- Inter-FPGA I/O

Register, BlockRegister and Inter-FPGA I/O are optional interface types.

ADC Input

Each DPU FPGA receives the corresponding ADC input. In normal 8 channels mode, each DPU FPGA receives two ADC channels input. In interleaved 4 channels mode, each DPU FPGA receives one ADC channel input. In M9703_FPGAx subnet, there is an input bus ADC(0:31) that you can connect with your real-time processing part to get the parallel ADC input streams. The interface diagram shows how the ADC input parallel streams correspond to ADC(0:31) bus.

For example, if you set Normal, all 8 channels mode, IN1 and IN2 are inputted to FPGA0. You can see IN1 is connected to ADC(0:15) and IN2 is connected to ADC (16:31) in the interface diagram. It means that input from IN1 is converted to 16 parallel input streams and you can get these parallel streams from ADC(0:15), where ADC(0) is the oldest sample and ADC(15) is the newest sample. Input from IN2 is converted to 16 parallel input streams and you can get these parallel streams from ADC(16:31), where ADC(16) is the oldest sample and ADC(31) is the newest sample.



FPGA0 Settings

☑ Register ☑ BlockRegister

If you set Interleaved, 1+3+5+7 mode, IN1 is inputted to FPGA0. You can see IN1 is connected to ADC(0:31) in the interface diagram. It means that input from IN1 is converted to 32 parallel input streams and you can get these parallel streams from ADC(0:31), where ADC(0) is the oldest sample and ADC(31) is the newest sample.

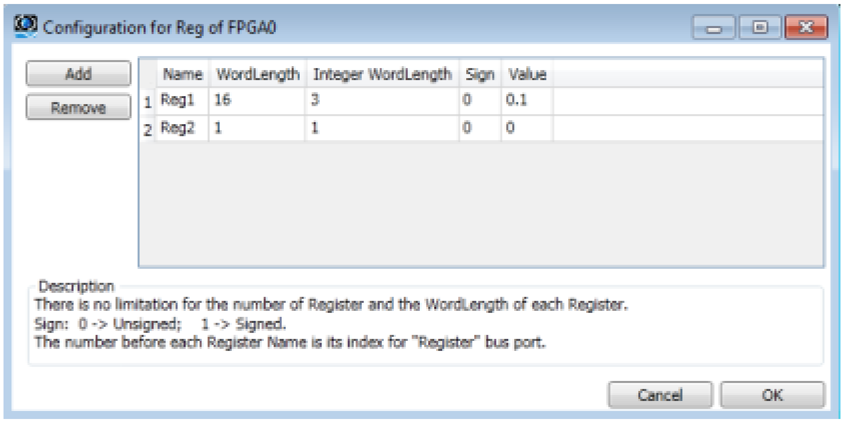
Register

Register Checkbox

Check/uncheck this box to enable/disable the "Register" interface for the current FPGA. When the box is checked, the corresponding Register interface will be shown in the FPGA interface diagram below the checkbox, and a button "Config Register" is also shown.
Note that this box is disabled (grey) when the current FPGA tab is set as "Copy Settings From Another FPGA" in parameter "Configuration Mode". The status of the box will be copied from another FPGA and can't be edited in the current FPGA tab.

Config Register Button

Push this button to open a new GUI to define the Register interface. This button is only shown when Register checkbox is checked.



| | Name | WordLength | Integer WordLength | Sign | Value |
|---|---|---|---|---|---|
| 1 | Reg1 | 16 | 3 | 0 | 0.1 |
| 2 | Reg2 | 1 | 1 | 0 | 0 |

Description
There is no limitation for the number of Register and the WordLength of each Register.
Sign: 0 -> Unsigned; 1 -> Signed.
The number before each Register Name is its index for "Register" bus port.

Click "Add" button to add a new row to define a new Register.
Click "Remove" button to remove a selected row (Register).
There is no limitation of the number of Registers, as long as the FPGA has enough area to implement them.

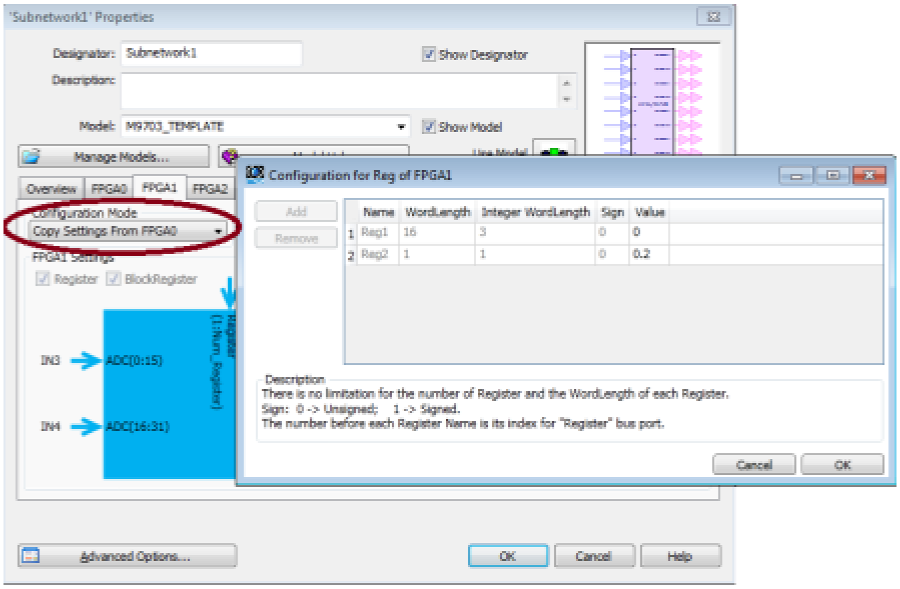
For the definition of a Register, you need input:

- Name: a string to represent the name of the Register.
- WordLength: Word Length (bits number) to represent the Register. Register is a fixed-point data. *About SystemVue fixed-point data type, please refer to SystemVue documentation: Home > Simulation > Data Flow > Fixed Point Simulation .*
- Integer WordLength: Integer Word Length (bits number to represent integer).
- Sign: 0->unsigned; 1->signed.

- Value: The value of current Register. You can input a floating-point value or equation variable name. The floating-point value will be converted to fixed-point type automatically (There may be overflow and quantization for the conversion. The default rule of overflow is Wrap and default rule of quantization is Truncate).

Note that if the current FPGA tab is set as "Copy Settings From Another FPGA", whether the current "Register" checkbox and "Config Register" button are enabled depends on the settings on the source FPGA tab. If the "Register" checkbox is checked in source FPGA tab, the checkbox is also checked and is disabled to edit on the current FPGA tab. In this case, "Config Register" button in the current FPGA tab is enabled and you can click it to open the Register definition GUI. But the definition of Register is copied from the source FPGA setting, so you can't add and remove Register and can't edit the name the fixed-point data format (WordLength, Integer WordLength and Sign) of Register. But you can edit the "Value" of defined Register.

For example, we can define FPGA interface for FPGA0 in its own FPGA tab GUI. And we copy the FPGA interface settings from FPGA0 to FPGA1 tab. Then when you click "Config Register" button in FPGA1 tab, you can see that the Register definition is copied from FPGA0 and you can only edit the "Value" of defined Register.



'Subnetwork1' Properties

Designator: Subnetwork1

☑ Show Designator

Description:

Model: M9703_TEMPLATE

☑ Show Model

Configuration Mode: Copy Settings From FPGA0

FPGA1 Settings

☑ Register ☑ BlockRegister

Configuration for Reg of FPGA1

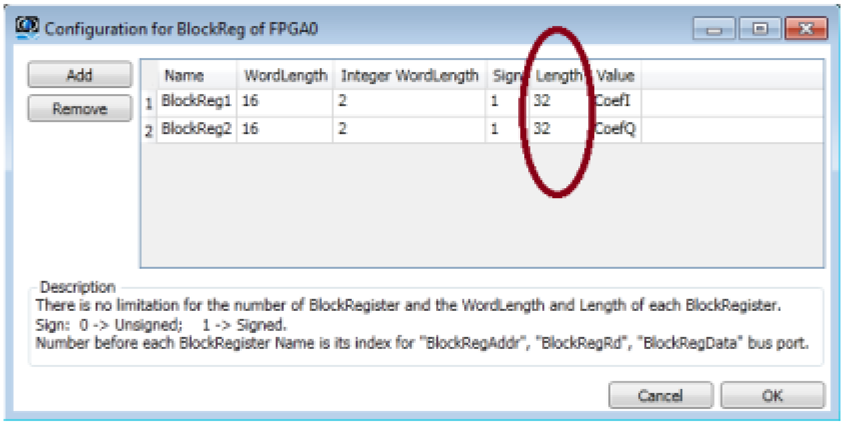
| | Name | WordLength | Integer WordLength | Sign | Value |
|---|---|---|---|---|---|
| 1 | Reg1 | 16 | 3 | 0 | 0 |
| 2 | Reg2 | 1 | 1 | 0 | 0.2 |

Description
There is no limitation for the number of Register and the WordLength of each Register.
Sign: 0 -> Unsigned; 1 -> Signed.
The number before each Register Name is its index for "Register" bus port.

**NOTE** The **Value** of each Register must be a **scalar variable or a constant.**

## BlockRegister

For BlockRegister, it has "BlockRegister" checkbox and "Config BlockRegister" button, which is similar as the setting of Register. The difference is that when you click "Config BlockRegister" to open BlockRegister definition GUI, BlockRegister has an additional attribute "Length" that specified the depth of each BlockRegister. The value of Length must be a scalar variable or a constant integer.



|  | Name | WordLength | Integer WordLength | Sign | Length | Value |
|---|---|---|---|---|---|---|
| 1 | BlockReg1 | 16 | 2 | 1 | 32 | CoefI |
| 2 | BlockReg2 | 16 | 2 | 1 | 32 | CoefQ |

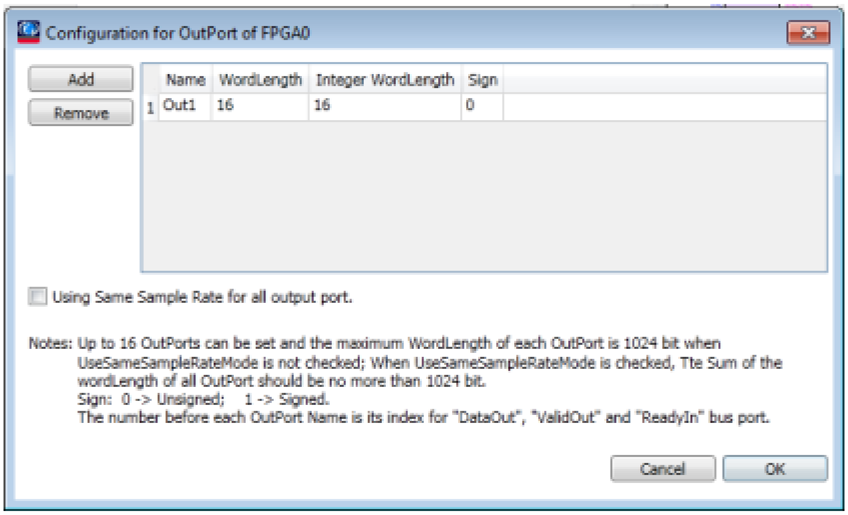
Description
There is no limitation for the number of BlockRegister and the WordLength and Length of each BlockRegister.
Sign: 0 -> Unsigned; 1 -> Signed.
Number before each BlockRegister Name is its index for "BlockRegAddr", "BlockRegRd", "BlockRegData" bus port.

There is no limitation of the number of BlockRegisters and the depth of each BlockRegister, as long as the FPGA has enough area to implement them.

**NOTE** The **Value** of each BlockRegister should be a **1xN vector**, where N is the value of **Length** for the corresponding BlockRegister. For example, in above figure, The **Value** of BlockReg1 **CoefI** must be a 1x32 vector, as **Length** of BlockReg1 is 32.

OutPort

OutPort is the mandatory FPGA interface. So it is unlike Register and BlockRegister, you have to define OutPort and there is no checkbox to disable OutPort.
When you click "Config Output" button, you can open a new GUI to define the OutPort.



|  | Name | WordLength | Integer WordLength | Sign |
|---|---|---|---|---|
| 1 | Out1 | 16 | 16 | 0 |

□ Using Same Sample Rate for all output port.

Notes: Up to 16 OutPorts can be set and the maximum WordLength of each OutPort is 1024 bit when UseSameSampleRateMode is not checked; When UseSameSampleRateMode is checked, Tte Sum of the wordLength of all OutPort should be no more than 1024 bit.
Sign: 0 -> Unsigned; 1 -> Signed.
The number before each OutPort Name is its index for "DataOut", "ValidOut" and "ReadyIn" bus port.

You can use Add or Remove button to create a new OutPort or remove a selected OutPort.

When Using Same Sample Rate for all output port is unchecked, each defined OutPort has its own ValidOut signal to indicate whether the current DataOut is valid. So the valid sample rates of OutPorts may be different. In this mode, the maximum number of OutPort is 16 and the maximum WordLength is 1024 for each OutPort.

When Using Same Sample Rate for all output port is checked, all defined OutPorts share the same ValidOut signal to indicate whether the current DataOut is valid. So the valid samples are generated in the same rate for all OutPorts. In this mode, the sum of all OutPorts' WordLength can't be larger than 1024.

The OutPort data type is also SystemVue fixed-point. So you can define the follow attributes of an OutPort:

- Name: a string to represent the name of an OutPort.
- WordLength: Word Length (the bits number) of an OutPort.
- Integer WordLength: Integer Word Length (the bits number to represent integer) of an OutPort.
- Sign: 0->unsigned; 1->signed.

Note that if the current FPGA tab is set as "Copy Settings From Another FPGA", the OutPort definition will be copied from the source FPGA OutPort definition. When you click "Config Output" button, you can view the copied OutPort definition, but can't edit anything.

Inter-FPGA I/O

When you enable Inter-FPGA I/O in Overview tab, the Inter-FPGA I/O interface will be shown in all enabled FPGA*x* tab. And you need to connect Inter-FPGA I/O ports in the corresponding M9703_FPGA*x* subnet. Please refer to Inter-FPGA I/O description. Otherwise, if you don't enable Inter-FPGA I/O in Overview tab, the Inter-FPGA I/O interface will not be displayed in FPGA*x* tab and you don't need connect these Inter-FPGA I/O ports in the corresponding M9703_FPGA*x* subnet.

### Software Simulation Behavior Description

After finishing the design entry in M9703 Design Template, users can create test waveforms and input them to the corresponding input ports of M9703 Design Template model. The test waveform can be floating point, so you can use all SystemVue models to create the waveforms. Note that the sample rate of input waveform must be the same as the one that you specified in GUI of M9703 Design Template. Then you can run SystemVue software simulation to debug and verify your M9703 FPGAs design.

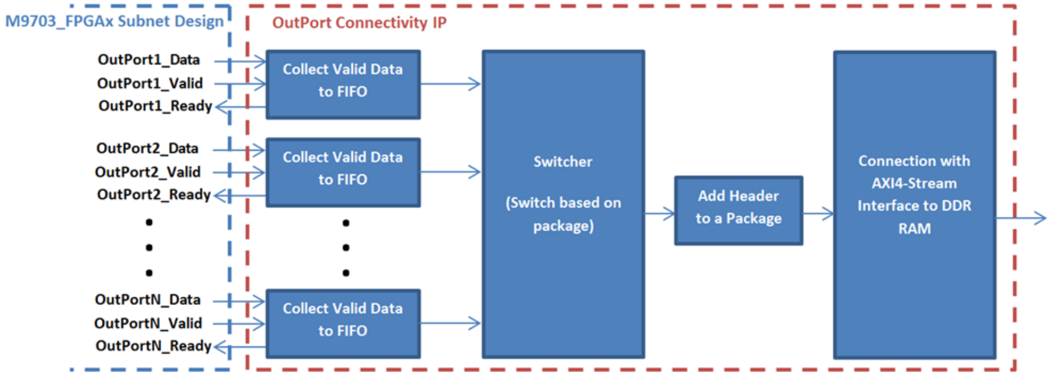
The OutPort of M9703 Design Template model have been described in this tutorial . The simulation behavior of output signals will be described in this part.

The output signals of four FPGAs are independent and the simulation behaviors of the four FPGAs are the same. So we can just study the simulation behavior of one FPGA.

The FPGA interfaces – ADC input, Register, BlockRegister and Inter-FPGA I/O, are simple and clear. Their behaviors have been described in previous parts. We will emphasis on the simulation behavior of OutPort in this part.

For Non-Same Sample Rate mode, we can define up to 16 OutPorts for one FPGA and the maximum word length can be 1024 bits for each OutPort. In the automatic FPGA programming file generation flow, SystemVue will create an HDL wrapper to connect an OutPort Connectivity IP core with the defined OutPorts automatically. The connectivity IP will package and switch the data from all OutPorts to the AXI4-

stream interface with data width 256 bits. The AXI4-stream interface provides a data path to send data stream from FPGA to on-board DDR3 SDRAM. The diagram implemented in FPGA programming file is shown below:



The connectivity IP has a FIFO for each OutPort. The FIFO only saves valid data from its corresponding OutPort and generates Ready to its corresponding OutPort. The outputs of all FIFOs are connected to the N-to-1 switcher, where N is the number of defined OutPorts. When the FIFO is full, it outputs Ready as 0 to its corresponding OutPort and isn't written in the new valid data. So only when both Valid and Ready are 1, the OutPort Data can be transferred to DDR3 SDRAM. Current data width of the switch is 128 bit.

In the connectivity IP, the data switching is based on package. It means that every time the switcher arbitrates and selects an OutPort, it will read a package from the selected OutPort FIFO. It will not arbitrate until the current package is transferred completely. The connectivity IP will also add a header at the beginning of each package to indicate which OutPort the current package comes from. The data length of a package is 64 times of data width of DDR3 SDRAM AXI4-stream interface (64*256 bits). The header length of a package is 2 times of data width of DDR3 SDRAM AXI4-stream interface (2*256 bits). So the total length of a package is 66 times of data width of DDR3 SDRAM AXI4-stream interface (66*256 bits). As the word length of an OutPort is an arbitrary number less than 1024, the connectivity IP will combine the valid data into a package and it may add some zero-padding bits at the end of a package.

So the total effective throughput (discarding header) to DDR3 SDRAM is (128 * 64 / 66) bit X FPGA working frequency, where 128 is N-to-1 switch data width.

In M9703 Design Template subnet, M9703AMemoryBus model is used to mimic the behavior of the connectivity IP. OutPorts of M9703_FPGA*x* user design subnet are connected to M9703AMemoryBus model, and this model generates Ready input of M9703_FPGA*x* subnet.

The connectivity IP is transparent to users and it's generated automatically according to the OutPort definition. The purpose of the connectivity IP description is to help you to understand the following rules for your M9703_FPGA*x* subnet design:

1. In real FPGA running, only when both ValidOut and ReadyIn of an OutPort are 1, the corresponding DataOut is transferred to DDR3 SDRAM. Only the data buffered in DDR3 SDRAM can be read back to SystemVue during M9703 instrument co-simulation.
2. In software simulation, DataOut output samples of an OutPort of M9703_FPGA*x* subnet are not the same as the corresponding DataOut output samples of M9703 Design Template top level subnet. But if you extract all DataOut output samples transferred to DDR RAM (the corresponding ValidOut and ReadyIn are both 1) of an M9703_FPGAx subnet's OutPort as the results sequence of this OutPort, and extract all DataOut output samples that their corresponding ValidOut are 1 for an M9703 Design Template subnet' OutPort as this OutPort's results sequence. The two results sequences are the exactly same.
3. In M9703 instrument co-simulation, DataOut output samples of an M9703ACosimBus model's OutPort are not the same as DataOut output samples of the corresponding M9703 Design Template subnet's OutPort in software simulation. But if you extract DataOut output samples that their corresponding ValidOut are 1 as results sequences, the results sequences from M9703 instrument co-simulation and software simulation are the exactly same.
4. As we just care about the valid output samples, above 2 and 3 can guarantee the valid output samples of an OutPort are the same between software simulation and M9703 instrument co-simulation.
5. If the sum of all OutPorts' word lengths is not larger than (128 * 64 / 66) ≈ 124 bits (the effect throughput to DDR3 SDRAM), even ValidOut of all OutPorts are always 1, the AXI4-Stream switch has enough bandwidth to transfer DataOut of all OutPorts to DDR3 SDRAM. So when the sum of all OutPorts' word lengths is not larger than 128 bits, ReadyIn of all OutPorts are always 1. You can use this condition to simply your M9703_FPGA*x* subnet design.

When the sum of all OutPorts' word lengths is larger than 124 bits, it may cause the connectivity IP outputs 0 to ReadyIn of an OutPort. Then you need add some logics in M9703_FPGAx subnet design to buffer valid DataOut samples, if you don't want to break the continuous data stream.

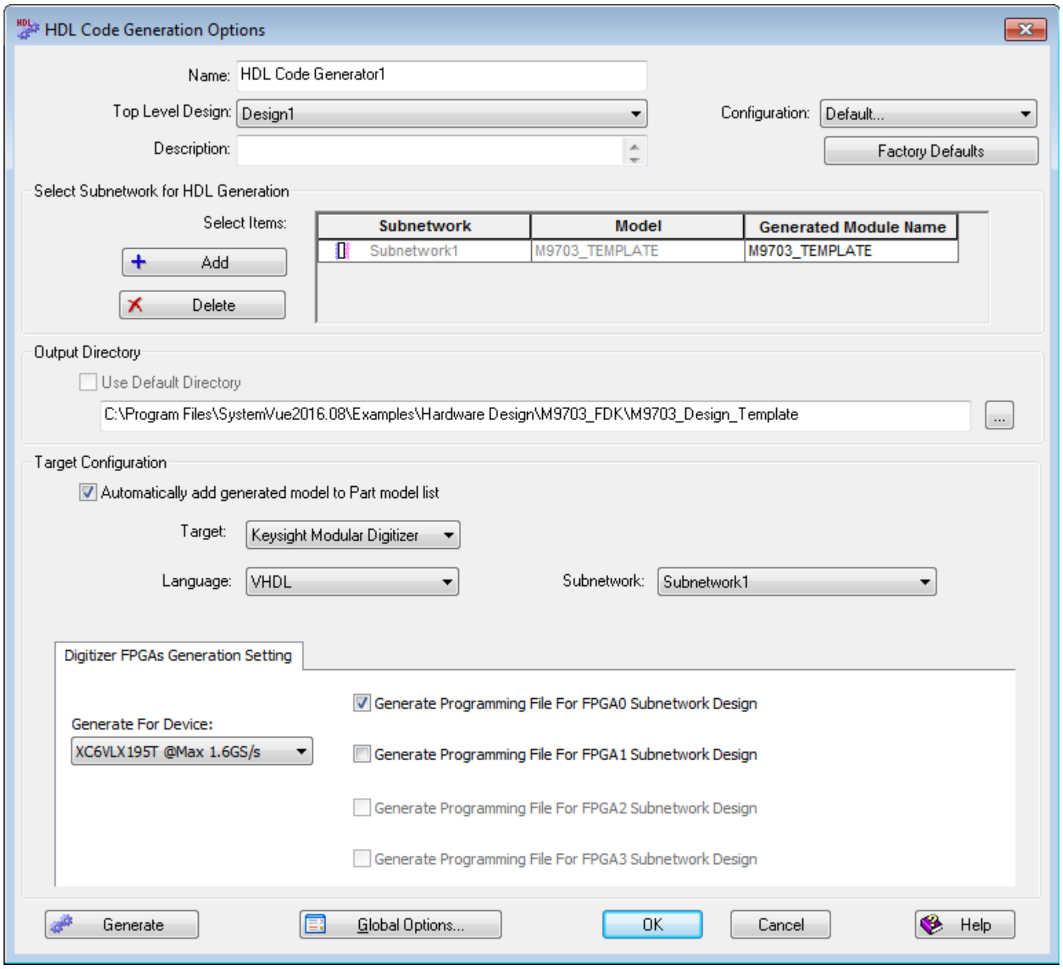
In SystemVue Example workspace M9703_FDK_Design_Template.wsv, there is a subnet "ValidExtract".You can use this subnet to only extract valid data samples, but it uses Dynamic Data Flow so need notice the limitation of Dynamic Data Flow.

For Same Sample Rate mode, it's actually a special case of Non-Same Sample Rate mode. Because all OutPort has the same ValidOut, SystemVue combines them together in background before switcher and just use a 1-to-1 switcher.

### M9703 FPGA Programming File Generation

After the verification of SystemVue software simulation, you can generate M9703 FPGA programming file with an automatic flow in SystemVue HDL Code Generator. For information on how to add a HDL Code Generator, refer HDL Code Generation. In HDL Code Generator, you can select the Target as Keysight Modular Digitizer. Then Click Add to select the M9703 Design Template in your current schematic to generate FPGA programming file.

You can select to generate FPGA programming file according which M9703_FPGA*x* subnet. Only enabled M9703_FPGA*x* subnets in M9703 Design Template GUI can be selected here.



HDL Code Generation Options

Name: HDL Code Generator1

Top Level Design: Design1

Configuration: Default...

Description:

Select Subnetwork for HDL Generation

Select Items:

| Subnetwork | Model | Generated Module Name |
|---|---|---|
| Subnetwork1 | M9703_TEMPLATE | M9703_TEMPLATE |

Output Directory

□ Use Default Directory

C:\Program Files\SystemVue2016.08\Examples\Hardware Design\M9703_FDK\M9703_Design_Template

Target Configuration

☑ Automatically add generated model to Part model list

Target: Keysight Modular Digitizer

Language: VHDL

Subnetwork: Subnetwork1

Digitizer FPGAs Generation Setting

Generate For Device: XC6VLX195T @Max 1.6GS/s

☑ Generate Programming File For FPGA0 Subnetwork Design

□ Generate Programming File For FPGA1 Subnetwork Design

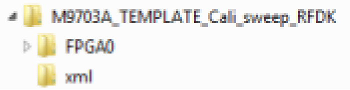
□ Generate Programming File For FPGA2 Subnetwork Design

□ Generate Programming File For FPGA3 Subnetwork Design

For M9703, you have two options of Generate For Device. For most M9703A and M9703B, they have XC6VLX195T DPU FPGA. So the option XC6VLX195T @Max 1.6 GS/s is used. Only for M9703B with -B01 bundle, it has XC6VSX315T DPU FPGA and -SR1 option (1.0 GS/s sample rate). So the option XC6VSX315T @Max 1.0GS /s (-B01 Bundle) will be used.

Then just click Generate. An automatic flow will be launch until you get the final M9703 FPGA programming files.

You can specify Generated Module Name in HDL Code Generator GUI. Then SystemVue will generate a folder with the name of specified Generated Module Name in the Output Directory. The generated file structure is shown below:



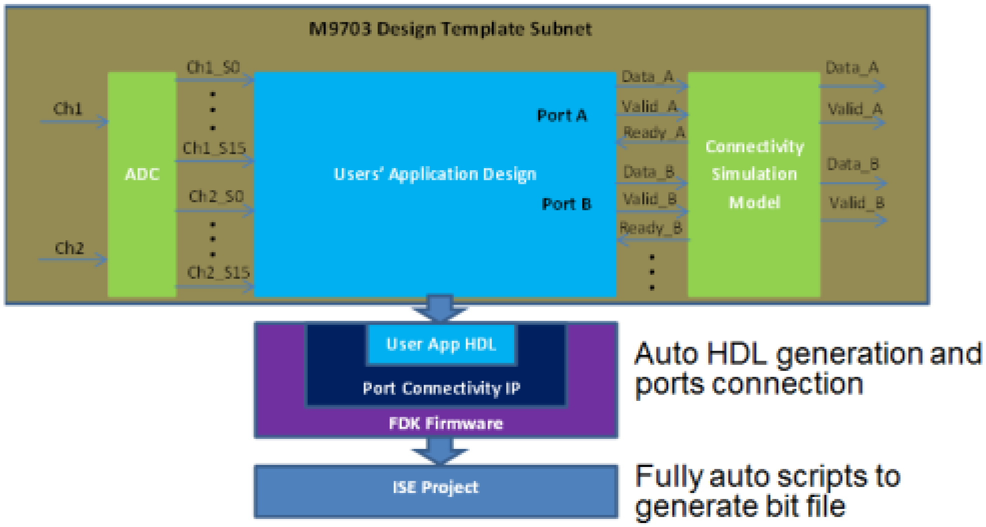
M9703A_TEMPLATE_Cali_sweep_RFDK
FPGA0
xml

For example, if you specify Generated Module Name as M9703_TEMPLATE_Cali_sweep_RFDK, then a folder M9703_TEMPLATE_Cali_sweep_RFDK will be generated in the Output Directory. There will be an xml folder and the corresponding FPGA*x* folders under the M9703_TEMPLATE_Cali_sweep_RFDK folder. For example, if you just generate FPGA programming file for FPGA0, only FPGA0 folder is generated here. While if you generate FPGA programming file for FPGA0~FPGA3, four folders FPGA0, FPGA1, FPGA2 and FPGA3 will be generated here. The generated FPGA programming file and ISE project are under the corresponding FPGA*x* folder. Two xml files are generated in xml folder to describe the generated FPGA configuration for the whole M9703 instrument.

In GUI of M9703ACosimBus model, there is a parameter FPGA Images Path. You can select the top level folder, such as M9703_TEMPLATE_Cali_sweep_RFDK folder, for this parameter. Then the GUI of M9703ACosimBus can parse the folder structure and xml file to understand your generated FPGA configuration.

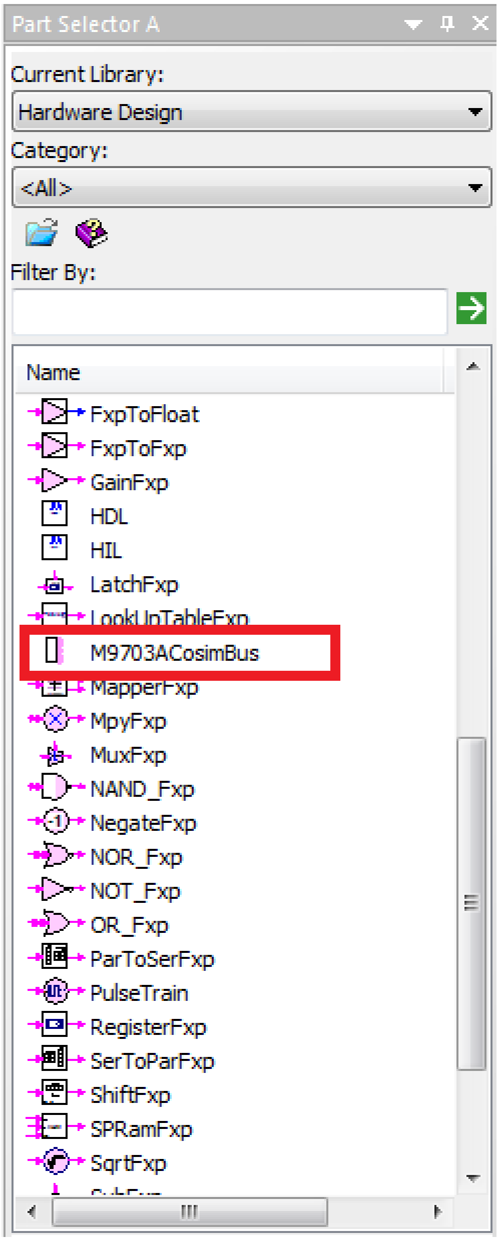
**CAUTION** Please don't modify the generated FPGA folder manually. It may cause that SystemVue can't pares it correctly and work with M9703 instrument correctly.

After clicking Generate button, SystemVue generates HDL codes for selected subnet application design at first. Then SystemVue generates an HDL wrapper file to connect with the generated application HDL codes automatically. The HDL wrapper file contains some connectivity IP to link your defined FPGA interface with M9703 FPGA infrastructure design that implements some foundational functions, such as link with ADC data stream, DDR3 SDRAM and PCI Express on backplane. Then SystemVue will create an ISE project and run this project to get the final FPGA programming file. The whole flow is automatic. A diagram of the generation flow is shown below.



## M9703 Instrument Co-simulation with SystemVue

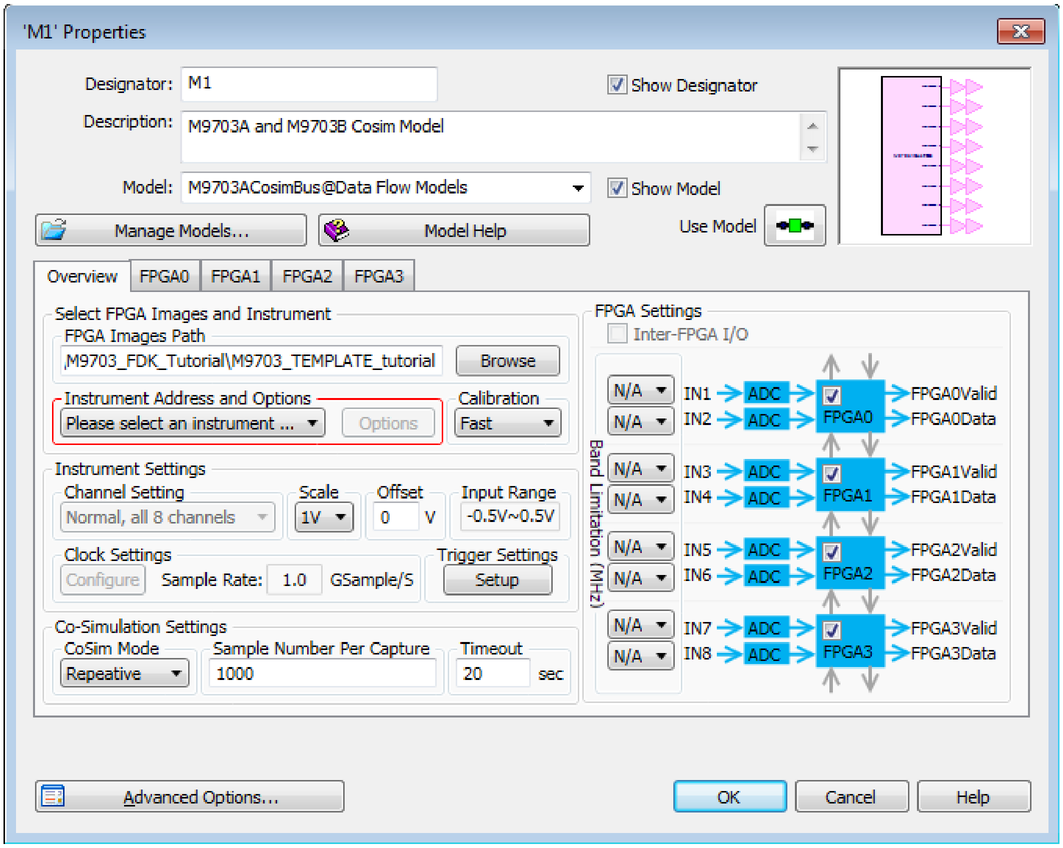
After the FPGA programming file is generated, you can use M9703ACosimBus model to download your FPGA programming file to M9703 instrument and capture the output results of FPGA to SystemVue simulation environment. You can find M9703ACosimBus model in Part Selector under Hardware Design.



M9703 Co-Simulation Model GUI

Place one M9703ACosimBus model in a schematic to instantiate one M9703 instrument, and you can place multiple M9703ACosimBus models to ask multiple M9703 instruments to work together in one SystemVue simulation.
Double click the M9703ACosimBus model, you can open its GUI. You can use this GUI to configure a M9703 instrument. The GUI of M9703ACosimBus has a similar architecture of the GUI of M9703 Design Template.

Overview Tab:



'M1' Properties

Designator: M1 ☑ Show Designator

Description: M9703A and M9703B Cosim Model

Model: M9703ACosimBus@Data Flow Models ☑ Show Model

Use Model

Overview | FPGA0 | FPGA1 | FPGA2 | FPGA3

Select FPGA Images and Instrument

FPGA Images Path: M9703_FDK_Tutorial\M9703_TEMPLATE_tutorial

Instrument Address and Options: Please select an instrument ...

Calibration: Fast

Instrument Settings

Channel Setting: Normal, all 8 channels

Scale: 1V

Offset: 0 V

Input Range: -0.5V~0.5V

Clock Settings: Sample Rate: 1.0 GSample/S

Co-Simulation Settings

CoSim Mode: Repeative

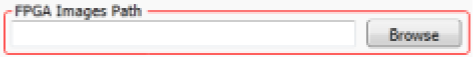
Sample Number Per Capture: 1000

Timeout: 20 sec

FPGA Settings

□ Inter-FPGA I/O

## FPGA Images Path

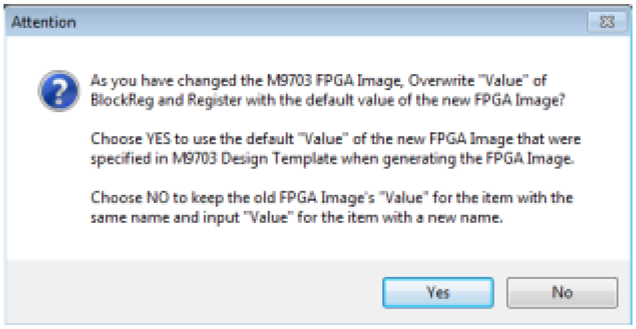


FPGA Images Path

Specify the generated M9703 FPGA image folder. You should specify the top level folder of a generated M9703 FPGA image.

Then the GUI can parse the generated FPGA image and extract the configuration that was implemented in the FPGA programming files. It can also find the generated FPGA programming files. Then in FPGAx tab, you can specify which programming file is used to program the corresponding FPGAx.

When you change FPGA Images Path, the below window will pop-up to ask you how to specify the values for the defined Register and BlockRegister.



Attention

As you have changed the M9703 FPGA Image, Overwrite "Value" of BlockReg and Register with the default value of the new FPGA Image?

Choose YES to use the default "Value" of the new FPGA Image that were specified in M9703 Design Template when generating the FPGA Image.

Choose NO to keep the old FPGA Image's "Value" for the item with the same name and input "Value" for the item with a new name.

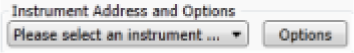
For a generated FPGA image, the configuration of Register and BlockRegister, such as WordLength, Integer WordLength, Sign and Length, are fixed and you can't modify them any more. But you can modify the values for the defined Register and BlockRegister. So if you change FPGA Images Path from an old FPGA image to a new FPGA image, you can specify whether you use the default values of new FPGA image or remain the values you specified for the old FPGA image.

The default values of the new FPGA image are specified in M9703 Design Template custom UI and recorded in the automatic FPGA image generation flow.

If you select to use the values of old FPGA image, all Register and BlockRegister items that have the same names between the new and old FPGA images will remain their values. For every new FPGA image's Register and BlockRegister item that has not the same name in the old FPGA image, the GUI will prompt you to input its value.

For example, if the old FPGA image has Registers with names Reg1 and Reg2 and the new FPGA image has Registers with name Reg2 and Reg3. You have selected the old FPGA image and specified Reg1=1 and Reg2=2 in the GUI of M9703ACosimBus model. Then you change FPGA Images Path to select the new FPGA image. The pop-up window will show. If you select Yes to use the default values of new FPGA image, the values of Reg2 and Reg3 of the new FPGA image will be loaded according to their default values. Otherwise, you select No to remain the values of the old FPGA image, Reg2 will keep its value 2 because there is a Register with the same name in the old FPGA image. And the GUI will prompt you to input a new value for Reg3 because there is no Register with the same name in the old FPGA image.

Instrument Address and Options

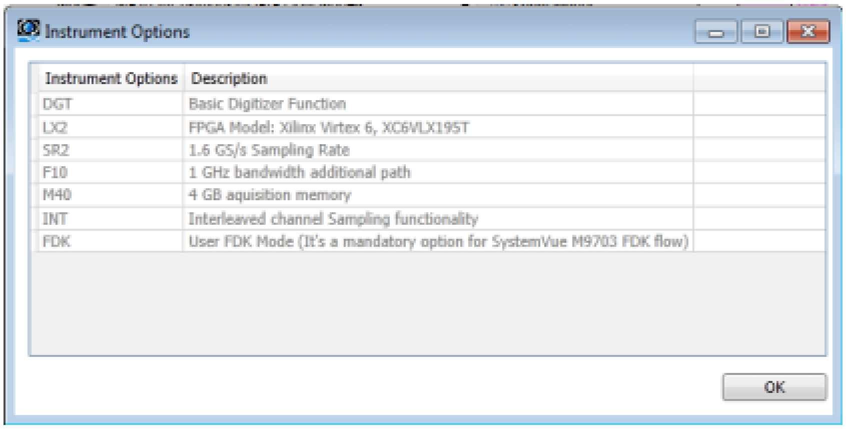

Instrument Address and Options
Please select an instrument ...
Options

Specify an M9703 instrument according to its address. When you open the GUI, it will detect all M9703 instruments connected to this computer and list the detected M9703 instruments addresses. Then you can select an M9703 instrument.

The default value of instrument address is a blank option and shows Please select an instrument ... in GUI. When you select a detected real M9703 instrument address, the GUI will initialize the selected M9703 instrument and extract its hardware options. It usually takes several seconds to initialize a M9703 instrument.

If you have specified a valid M9703 instrument address and click Ok to save the parameters of M9703ACosimBus model, then when you open the GUI next time, the GUI will detect all connected M9703 instruments again and check whether the address that was specified last time is still in current connected instruments list. If so, the GUI will initialize the previously configured M9703 instrument, otherwise, it will switch to the blank option.

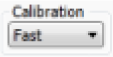
The GUI will extract M9703 instrument hardware options during the instrument initialization. After an M9703 instrument is initialized successfully, you can view all its hardware options by clicking Option button.



| Instrument Options | Description |
|---|---|
| DGT | Basic Digitizer Function |
| LX2 | FPGA Model: Xilinx Virtex 6, XC6VLX195T |
| SR2 | 1.6 GS/s Sampling Rate |
| F10 | 1 GHz bandwidth additional path |
| M40 | 4 GB aquisition memory |
| INT | Interleaved channel Sampling functionality |
| FDK | User FDK Mode (It's a mandatory option for SystemVue M9703 FDK flow) |

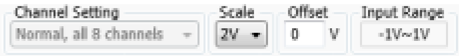
In addition, the GUI will also check whether the selected M9703 instrument has any conflict with the generated FPGA programming files. It will check whether the selected M9703 instrument has FDK option (the M9703 instrument has to have FDK option to use a customized FPGA programming file). If not, the M9703 instrument can't be selected and the M9703 address will be switched to blank option automatically.

Calibration


Calibration: Fast

It has three options: None, Fast and Full.

- None: It will not do M9703 instrument self-calibration at the beginning of simulation.
- Fast: It will do M9703 instrument self-calibration for the current instrument parameters setting at the beginning of simulation. It usually takes several seconds for the fast self-calibration.
- Full: It will do M9703 instrument self-calibration for all instrument parameter setting at the beginning of simulation. It usually takes about one minute for the full self-calibration.


Channel Setting: Normal, all 8 channels
Scale: 2V
Offset: 0 V
Input Range: -1V~1V

Channel Setting

It has three options: Normal, all 8 channels, Interleaved, 1+3+5+7 and Interleaved, 2+4+6+8.

It's read-only for M9703ACosimBus model. The Channel Setting value is gotten from the specified FPGA image file, so it shows the channel setting that you configured for your own FPGA programming file.

Note: If you configured channel setting as Interleaved mode and generate the FPGA programming file, you have to use an M9703 instrument with INT option in order to use your FPGA programming file. The GUI will check your selected M9703 instrument. If the selected M9703 instrument has no INT option, it will pop-up an error window and ask you to change to another M9703 FPGA image or change to another M9703 instrument.

Scale

It has two options: 1V and 2V. It's the full voltage range for input signal. This parameter is used to control the ADC input full scale of M9703 instrument.

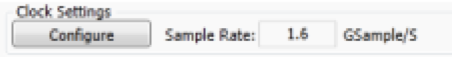
Offset

This parameter is used to control the offset adjust of M9703's input amplifier. The valid range is from -2*Scale to 2*Scale. When the input signals ride on a DC value, you can use the offset adjustment to make the input signals be in the center of valid input range of M9703 instrument, which can get the optimal quantization results.

Input Range

It is a read-only item to show the valid range of input signal based on specified Scale and Offset parameters.

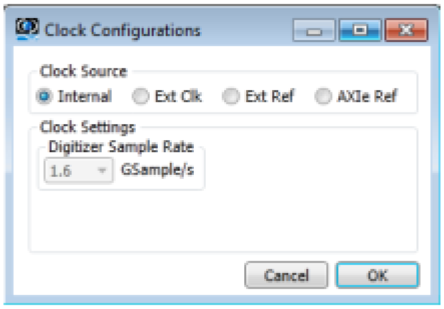
Clock Settings



Sample Rate: 1.6 GSample/S

In the Clock Settings area, there is a Configure button and a read-only Sample Rate display. You can click Configure button to set clock options. The final sample rate that the M9703 instrument works on is related to M9703 hardware option, channel setting and clock configuration and will be displayed in the GUI.

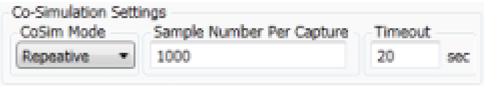
After clicking Configure button, a new GUI will be opened to configure the clock of M9703 instrument.



Clock Source: ● Internal ○ Ext Clk ○ Ext Ref ○ AXIe Ref

Digitizer Sample Rate: 1.6 GSample/s

There are four clock options. Please refer to Overview of M9703 High-Speed Digitizer for the details of clock options.



CoSim Mode: Repeative

Sample Number Per Capture: 1000

Timeout: 20 sec

Cosim Mode

There are two options: Single Pass and Repeative.

Single Pass:

It only asks M9703 digitizer to do data capture once.

When M9703 instrument starts to capture the data, M9703 FPGAs will get raw data samples from ADCs and then the raw ADC input samples go through your own FPGA design logics to generate your defined OutPort format output samples. M9703 digitizer will buffer a number of continuous your defined OutPort format output samples in on-board DDR3 RAM without any data loss. The number is specified in next parameter "Sample Number Per Capture".

After the "Sample Number Per Capture" samples of the first data capture are outputted, M9703ACosimBus model will not output valid samples any more, that means that output ports FPGA0_Valid ~ FPGA3_Valid are always 0 after the first data capture.

Repeative:

After the "Sample Number Per Capture" samples of a data capture are outputted, M9703ACosimBus model will do next data capture and output "Sample Number Per Capture" samples for next data capture until the end of simulation.

Sample Number Per Capture

It's the number of valid output samples for each data capture. It's the same for all four FPGAs.

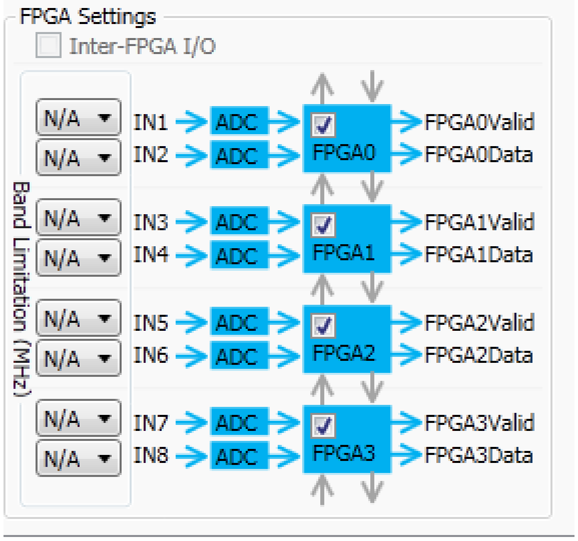
If there are multiple OutPorts defined for FPGAx, M9703ACosimBus model will output valid samples for each OutPorts alternatively. The number specified in "Sample Number Per Capture" is the sum of all OutPorts' valid samples for each data capture. You can't specify the valid output samples number for each OutPort, because it depends on the actual valid samples generation rate for each OutPort. For a given buffer size, you can't forecast whether the buffer size is enough to save a specified number of valid samples for an OutPort.

For example, given the buffer size is fixed and you have two OutPorts, OutA and OutB. If the actual valid samples generation rate for OutA and OutB is similar, the buffer size is enough to capture NumX valid samples for both OutA and OutB. But if the actual valid samples generation rate of OutA is large and that of OutB is very small, the buffer size may not be enough to capture NumX valid samples for OutB. So you can only specify the sum number of all OutPorts' valid samples, then SystemVue can forecast whether M9703 instrument has enough buffer size for your data capture. If M9703 instrument hasn't enough buffer size for your specified "Sample Number Per Capture", SystemVue will post an error during simulation and tell you the maximum number for "Sample Number PerCapture" according to your current configuration.

TimeOut

It specifies the maximum time limitation for an M9703 instrument to buffer "Sample Number Per Capture" valid output samples to DDR3 SDRAM for each data capture. If M9703 instrument can't save enough valid output samples within this TimeOut time, SystemVue simulation will abort.

It is usually used to avoid SystemVue simulation hang because your FPGA design doesn't generate valid output samples. Sometimes, you FPGA design only generates valid output samples under a specific condition, such as detecting a synchronization signal. If your input analog signals to M9703 instrument don't contain the synchronization signal, your FPGA design will not generate valid output samples and your SystemVue simulation will hang. So you can use this parameter to avoid the simulation hang.



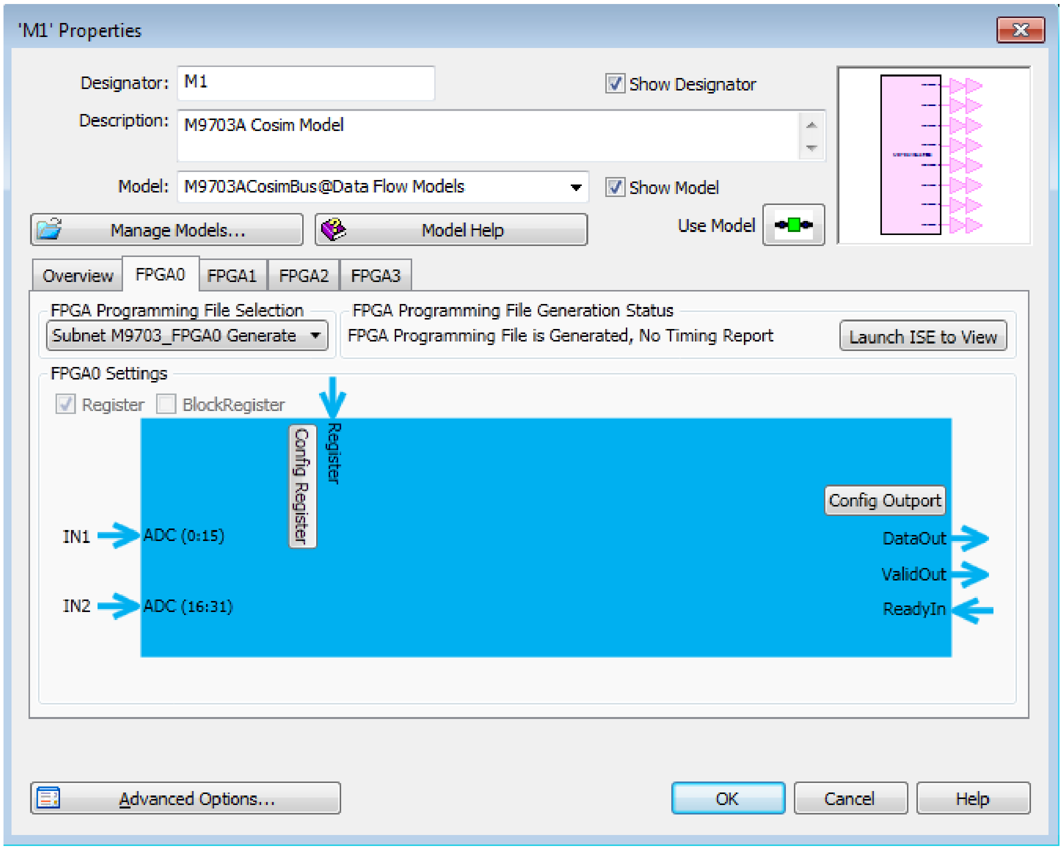
FPGA Settings

FPGA Settings area displays the FPGA architecture according to your current configuration. You can use the checkboxes in this area to select which FPGAs on M9703 instrument you want to enable and use. There is the corresponding FPGAx tab for each selected FPGA. After specifying a valid FPGA image, you can configure each selected FPGA in its corresponding FPGAx tab.

Bandwidth Limitation

It specifies whether to use or bypass an analog filter before each ADC on M9703 instrument.
If M9703 instrument has F05 option, it's mandatory to use the 600MHz bandwidth analog filter.
If M9703 instrument has F10 option, you can select "N/A" to bypass the analog filter or select "600MHz" to use the filter.

FPGAx Tab



'M1' Properties

Designator: M1

☑ Show Designator

Description: M9703A Cosim Model

Model: M9703ACosimBus@Data Flow Models

☑ Show Model

Use Model

Overview | FPGA0 | FPGA1 | FPGA2 | FPGA3

FPGA Programming File Selection: Subnet M9703_FPGA0 Generate

FPGA Programming File Generation Status: FPGA Programming File is Generated, No Timing Report

FPGA0 Settings

☑ Register ☐ BlockRegister

FPGAx tab is used to configure the real FPGAx on M9703 instrument.

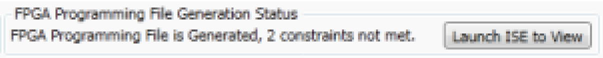
FPGA Programming File Selection

It specifies which FPGA programming file will be programmed to the real FPGAx on M9703 instrument.

The GUI will parse your specified FPGA image folder and extract all generated FPGA programming files to list here. At most you can generate four FPGA programming files for subnet design M9703_FPGA0 ~ M9703_FPGA3 in M9703 Design Template. You can also just select one or some subnets design for FPGA programming files generation in HDL Code Generator.

You can specify a programming file for different FPGAs on M9703 instrument. For example, you can just generate one FPGA programming file using M9703_FPGA0 subnet design, and specify to use this programming file in all four FPGAx tabs, which means that you can program this FPGA programming file to four FPGAs on M9703 instrument. Then the four FPGAs will have the same function. (But you can still specify different values of Register and BlockRegister for the four FPGAs programmed by the same programming file.)

FPGA Programming File Generation Status



FPGA Programming File Generation Status

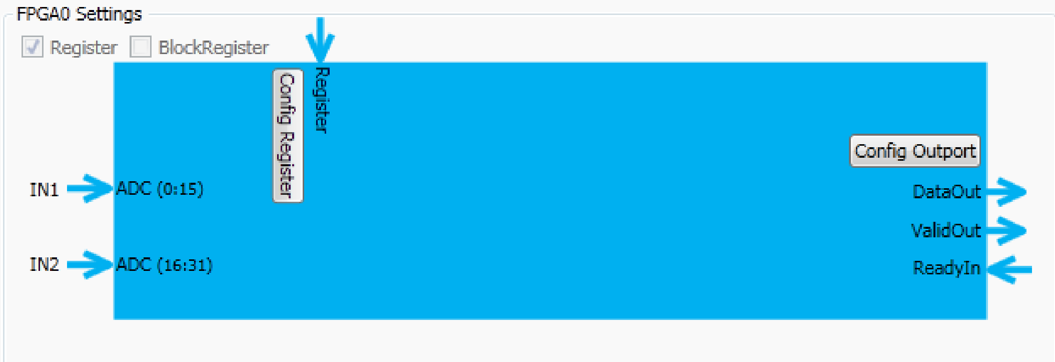
FPGA Programming File is Generated, 2 constraints not met.

This area can help you to know the generation result of FPGA programming file that you specified at FPGA Programming File Selection.

The left part of this area will display some read-only information to tell you whether the FPGA programming file is generated successfully. If it's generated, whether there is any timing constraints violation.

The right part of this area is a button Launch ISE to View. You can click this button to open Xilinx ISE GUI and load the ISE project for your specified programming file generation. Then you can view all generation reports in ISE environment. Especially you can view timing report when the read-only information on the left shows some timing constraints violations.

FPGAx Settings



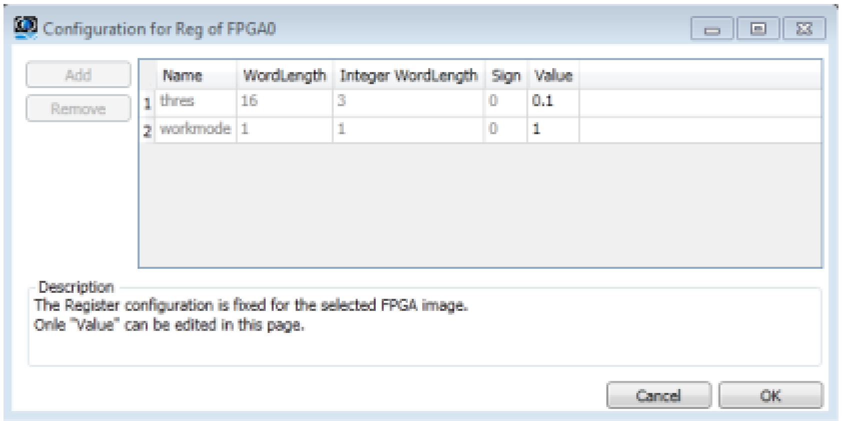
This area displays the FPGAx interface. The M9703 Design Template has a similar area in its custom UI. You can define the FPGAx interface of Register, BlockRegister and OutPort in M9703 Design Template custom UI. And in the custom UI of M9703ACosimBus model, you can view the generated FPGA interface of Register, BlockRegister and OutPort in the specified FPGA programming file. You can modify the values of Register and BlockRegister in this GUI, but you can't modify the interface format, such as WordLength, because the interface format is fixed for generated FPGA programming file.

Register and BlockRegister checkboxes

The two checkboxes are read-only in this UI. If Register or BlockRegister interface was defined in your specified FPGA programming file, the corresponding checkbox will be shown checked and the corresponding FPGA interface will be shown in the FPGA interface figure in the GUI.

Config Register Button

If Register interface was defined when generating the specified FPGA programming file, this button will be shown in the FPGA interface figure. After you click this button, a new UI will be opened.



| | Name | WordLength | Integer WordLength | Sign | Value |
|---|---|---|---|---|---|
| 1 | thres | 16 | 3 | 0 | 0.1 |
| 2 | workmode | 1 | 1 | 0 | 1 |

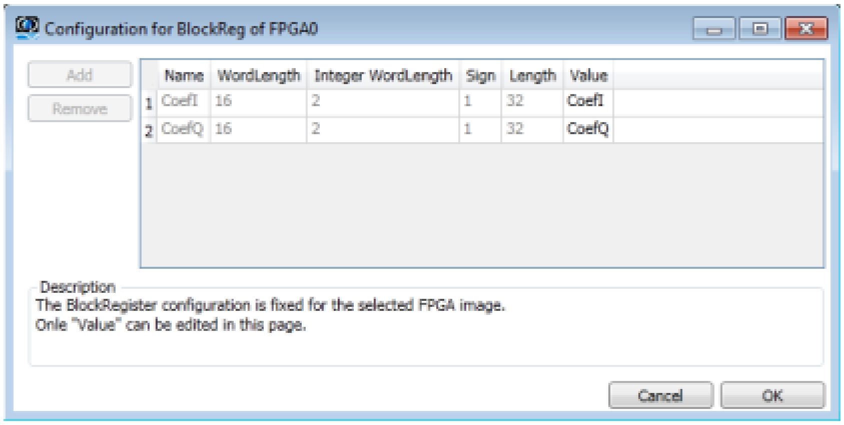
The Register configuration is fixed for the selected FPGA image.
Onle "Value" can be edited in this page.

You can view "Name", "WordLength", "Integer WordLength" and "Sign" of the defined Register in your specified FPGA programming file, but you can't modify them because they are fixed for a generated FPGA programming file. But you can modify "Value" of Register to re-configure the Register values for M9703 instrument co-simulation.

You can use a float-point number as the value of a Register or use a variable that is defined in equation.

Config BlockRegister Button

If BlockRegister interface was defined when generating the specified FPGA programming file, this button will be shown in the FPGA interface figure. After you click this button, a new UI will be opened.



| | Name | WordLength | Integer WordLength | Sign | Length | Value |
|---|---|---|---|---|---|---|
| 1 | CoefI | 16 | 2 | 1 | 32 | CoefI |
| 2 | CoefQ | 16 | 2 | 1 | 32 | CoefQ |

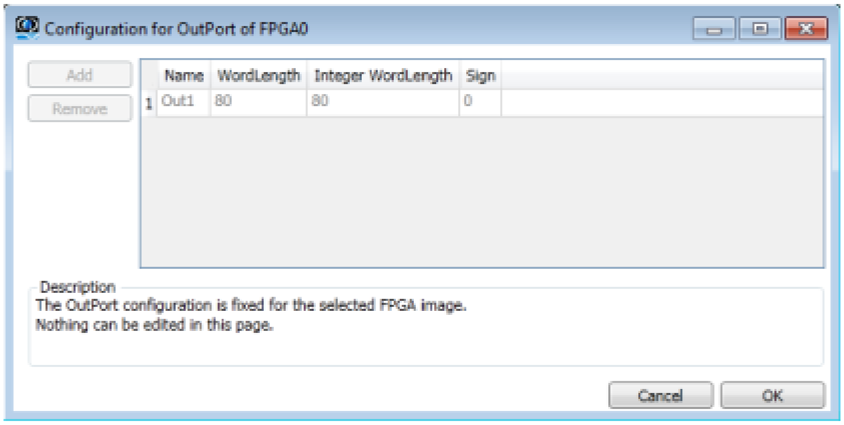
The BlockRegister configuration is fixed for the selected FPGA image.
Onle "Value" can be edited in this page.

You can view "Name", "WordLength", "Integer WordLength", "Sign" and "Length" of the defined BlockRegister in your specified FPGA programming file, but you can't modify them because they are fixed for a generated FPGA programming file. But you can modify "Value" of BlockRegister to re-configure the BlockRegister values for M9703 instrument co-simulation.

You can use a vector with the corresponding "Length" as the value of a BlockRegister or use a variable that is defined in equation.

Config OutPort Button

OutPort is a mandatory FPGA interface, so Config OutPort button and the OutPort interface is always shown in the FPGA interface figure.
After you click this button, a new UI will be opened.



|   | Name | WordLength | Integer WordLength | Sign |
|---|---|---|---|---|
| 1 | Out1 | 80 | 80 | 0 |

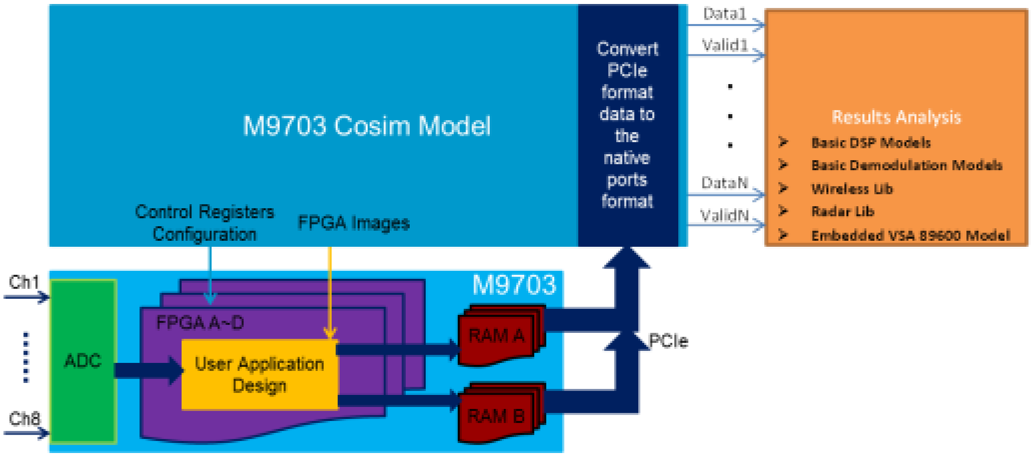
Description
The OutPort configuration is fixed for the selected FPGA image.
Nothing can be edited in this page.

You can view "Name", "WordLength", "Integer WordLength" and "Sign" of the defined OutPort in your specified FPGA programming file, but you can't modify them because they are fixed for a generated FPGA programming file. Nothing can be modified for the defined OutPort interface in a generated FPGA programming file.

### M9703 Co-Simulation Model Simulation Behavior Description

M9703ACosimBus model has the same output ports as M9703 Design Template subnet, but it hasn't any input port. Usually, you can connect the output ports of M9703ACosimBus model to the same bus wires as M9703 Design Template subnet to reuse the data analysis design of software simulation for M9703 instrument co-simulation.

When you have connected M9703ACosimBus model well in a schematic, you can run SystemVue simulation. Then SystemVue will do the following steps to link with real M9703 instrument and capture the results data back SystemVue.



SystemVue On PC

1. SystemVue leverage MD1 driver to download specified FPGA programming files to enabled FPGAs, configure the enabled FPGAs according to your settings in the custom UI of M9703ACosimBus and do self-calibration for all enabled FPGAs.
2. If Register or BlockRegister is defined in the FPGA programming file, SystemVue writes initial values for the defined Register or BlockRegister on enabled FPGAs.
3. After writing the initial values of Register and BlockRegister for all enabled FPGAs, all enabled FPGAs are ready to capture data. Then SystemVue asks M9703 to send a "Start" signal to all enabled FPGAs, the "Start" signal is guaranteed to reach all enabled FPGAs at the same time. Then all enabled FPGAs can capture ADC input data from the same starting time.
4. The enabled FPGAs process the real-time ADC input data stream as designed in user application design and generate OutPort data streams.
5. The OutPort data streams are packaged and switched and then buffered into M9703 on-board DDR RAM.
6. In the GUI of M9703ACosimBus model, there is a parameter "Samples Number Per Capture". When SystemVue detects that DDR RAM has buffered enough OutPort data for specified "Samples Number Per Capture", SystemVue will stop ADC data capture and read back the buffered OutPorts data via PCI Express connection between M9703 instrument and PC.
7. SystemVue will automatically convert PCI Express data format to user defined OutPort data format and output the OutPort data via the output ports of M9703ACosimBus model. As the OutPort data streams are buffered in DDR RAM based on packages, they will also be outputted based on packages from the output ports of M9703ACosimBus model.
8. After all enabled FPGAs output "Sample Number Per Capture" samples, if you're using "Single Pass" co-sim mode, SystemVue will not ask M9703 instrument to capture data and ValidOut of all enabled FPGAs will be 0 until the end of simulation. If you're using "Repeative" co-sim mode, SystemVue will go to step (3) and ask M9703 instrument to capture data again.

For Non-Same Sample Rate mode, the data buffered in DDR3 RAM is based on the OutPort package. One package just includes valid output data from one OutPort, and the package header will have the information of OutPort index. So when SystemVue read data from DDR3 RAM, it is also OutPort package based. The number of samples in one package is not fixed, and it's variable as the word length of OutPort.

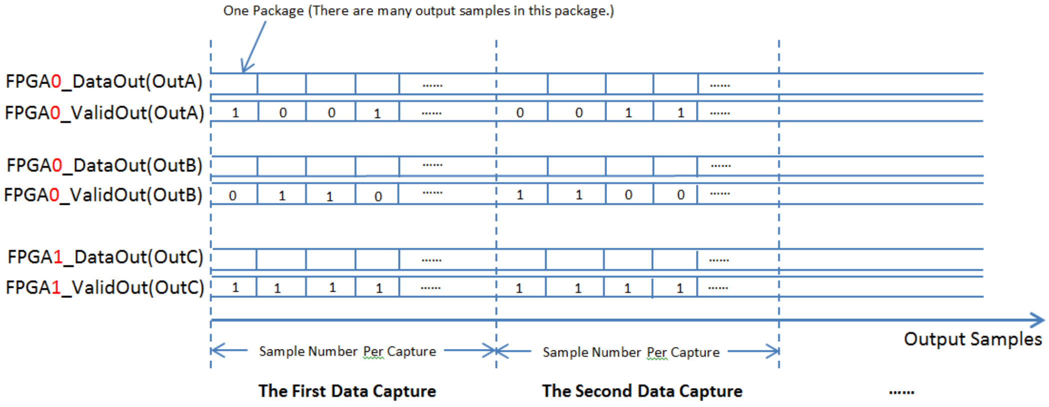
When you just define one OutPort, all packages are for this OutPort and all physical bandwidth between FPGA and DDR3 RAM is used for this OutPort data. So the valid output of the OutPort will be always 1.

When you define multiple OutPorts, M9703ACosimBus model will output valid samples for each OutPorts alternatively. It means that ValidOut of OutPorts will be 1 alternatively. The number of "Sample Number Per Capture" is the sum of all OutPorts' valid samples for each data capture. You can't specify the valid output samples number for each OutPort, because it depends on the actual valid samples generation rate for each OutPort. For a given buffer size, you can't forecast whether the buffer size is enough to save a specified number of valid samples for an OutPort.

For example, we defined two OutPorts, OutA and OutB for FPGA0 and define one OutPort OutC for FPGA1. When SystemVue read back OutA package, DataOut of OutA will output valid data, and ValidOut of OutA is 1; while ValidOut of OutB is 0 during the time. When SystemVue read back OutB package, DataOut of OutB will output valid data, and ValidOut of OutB is 1; while ValidOut of OutA is 0 during the time. OutA and OutB will output valid data alternatively, and for each output sample, only one OutPort data is valid.
While for FPGA1, ValidOut of OutC will always be 1 and DataOut of OutC will always output valid data.

For both FPGA0 and FPGA1, every data capture generates "Sample Number Per Capture" output samples. The OutPort output samples within one data capture are gotten by processing a span of continuous ADC data stream. While when a new data capture starts, the data stream is not continuous with the previous data capture.

It is shown in below figure:



For Same Sample Rate mode, it's a special case of Non-Same Sample Rate mode, it actually just have one combined output branch. So in this mode, FPGA*x*_ValidOut is always 1.

## Tutorial of SystemVue M9703 FPGA Design Flow

We will use a simple FIR example to go through the whole M9703 FPGA design flow. You can find the tutorial example from "Help > Open Examples > Hardware Design > M9703_FDK > M9703_FDK_Tutorial".

In normal 8 channels mode, two ADC input channels are connected to a FPGA. In this FIR example, we want to connect IQ signal of a baseband signal to the two channels and ask the IQ signals to go through the FIR filters with the same coefficients. We implemented two set of FIR coefficients, one is for Low Pass Filter and the other is for High Pass Filter. We defined a Register to select which set of coefficients are used.

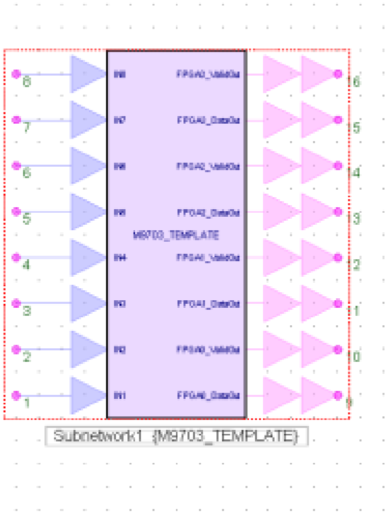
We can use FPGA0 and FPGA1 of M9703 to do software simulation. Then we can run the automatic flow to generate FPGA programming file and use the generated FPGA programming file for M9703 instrument co-simulation.

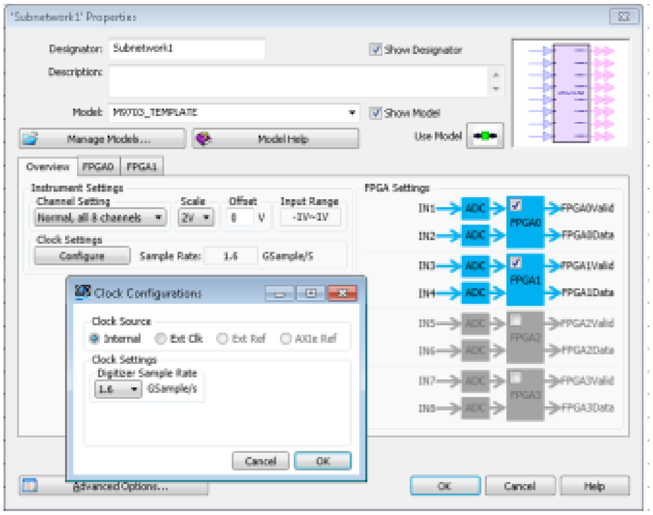
### Design the FIR Example for M9703 FPGA

We need start from a blank M9703 Design Template. So you can open "Help > Open Examples > Hardware Design > M9703_FDK > M9703_Design_Template > M9703_Design_Template.wsv" and save it to a writable directory.

Then you can drag-and-drop a M9703_TEMPLATE subnet to a schematic and double click it to open its custom UI.

In the custom UI, you need to uncheck FPGA2 and FPGA3 in FPGA Settings area, because we just use FPGA0 and FPGA1 for software simulation in this example. Besides, you need click "Configure" button in Clock Settings area to open the Clock Configuration GUI and select to use "Internal" clock mode at 1.6 GSample/S sample rate.




'Subnetwork1' Properties

Designator: Subnetwork1 ☑ Show Designator

Description:

Model: M9703_TEMPLATE ☑ Show Model

Overview | FPGA0 | FPGA1

Instrument Settings

Channel Setting: Normal, all 8 channels; Scale: 2V; Offset: 0 V; Input Range: -1V~1V

Clock Settings: Sample Rate: 1.6 GSample/S

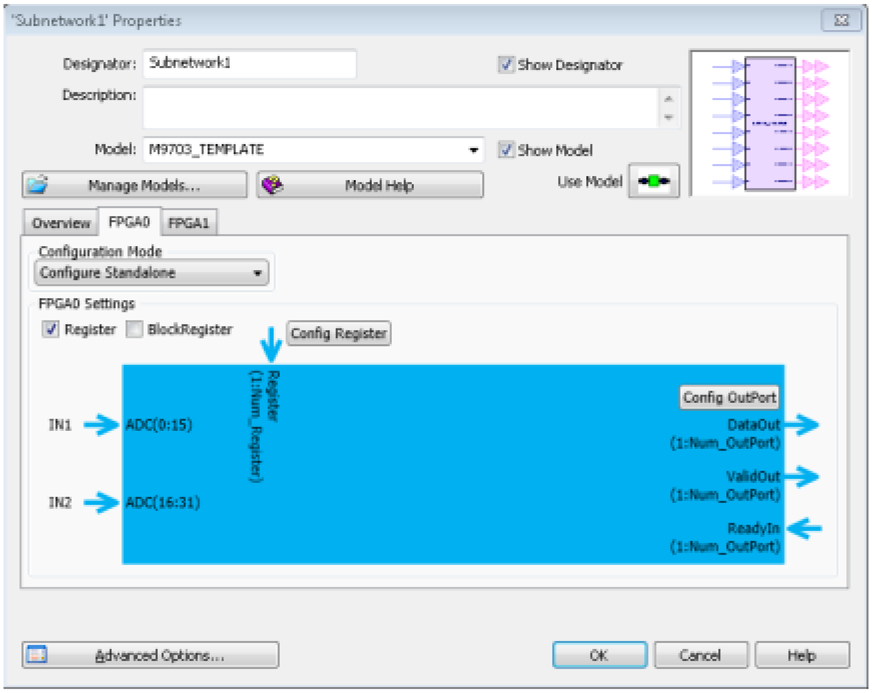
FPGA Settings: FPGA0 ☑, FPGA1 ☑, FPGA2 □, FPGA3 □

Clock Configurations

Clock Source: ● Internal ○ Ext Clk ○ Ext Ref ○ AXIe Ref

Clock Settings: Digitizer Sample Rate 1.6 GSample/s

Then you can click FPGA0 tab to configure the FPGA interface for FPGA0. As we just use Register in this example, you need to uncheck BlockRegister checkbox in this UI. Then you can click "Config Register" and "Config OutPort" buttons to configure Register and OutPort individually.



'Subnetwork1' Properties

Designator: Subnetwork1

Description:

Model: M9703_TEMPLATE

☑ Show Designator

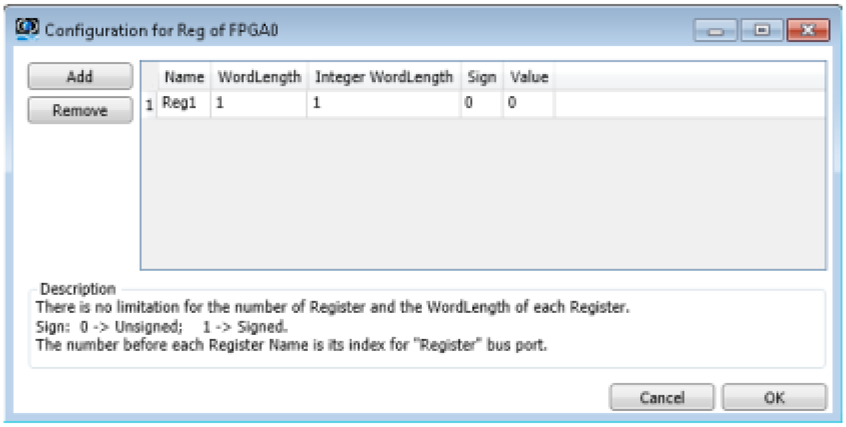
☑ Show Model

Configuration Mode: Configure Standalone

FPGA0 Settings

☑ Register □ BlockRegister

For Register configuration, we just need 1 bit to switch two FIR coefficients, so we can configure Register as below. We can give Value as 0 to switch to LPF FIR filter for FPGA0.

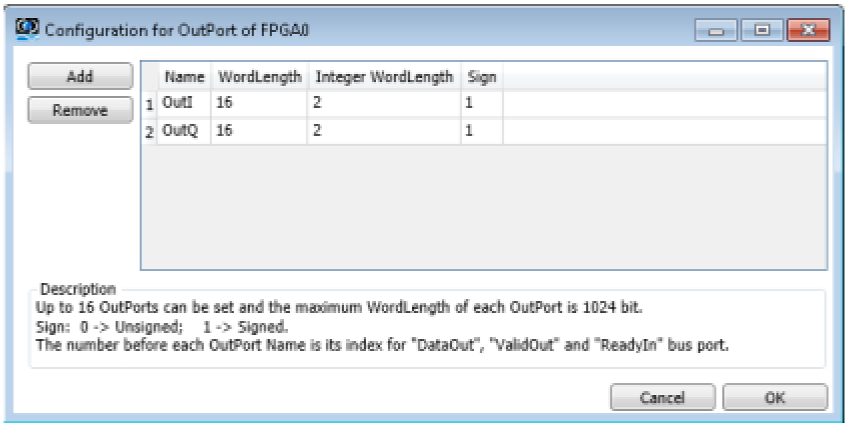


Configuration for Reg of FPGA0

| | Name | WordLength | Integer WordLength | Sign | Value |
|---|---|---|---|---|---|
| 1 | Reg1 | 1 | 1 | 0 | 0 |

Description
There is no limitation for the number of Register and the WordLength of each Register.
Sign: 0 -> Unsigned; 1 -> Signed.
The number before each Register Name is its index for "Register" bus port.

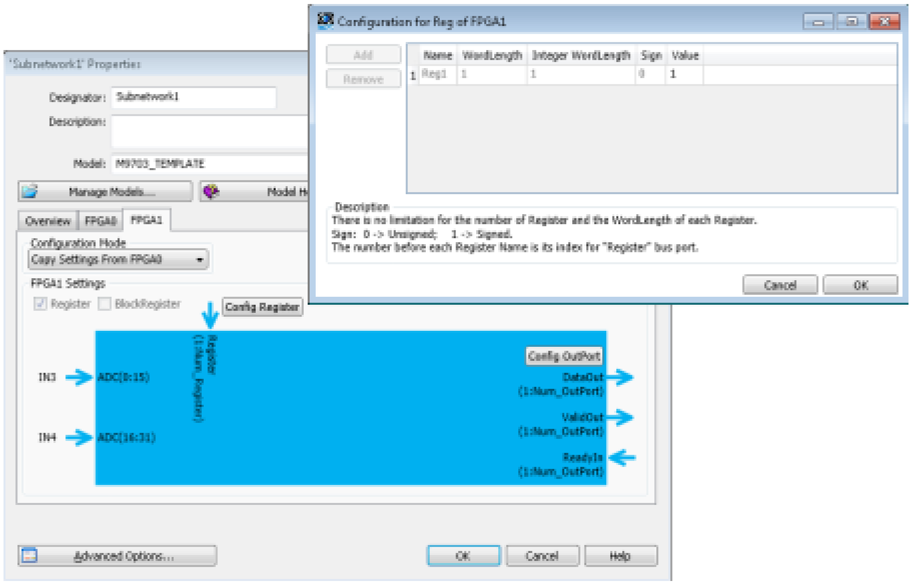
For OutPort configuration, we can define OutI and OutQ for IQ data output from the FIR filters. And the fixed-point data format is defined as Signed, wordlength = 16, integer wordlength = 2.



|   | Name | WordLength | Integer WordLength | Sign |
|---|---|---|---|---|
| 1 | OutI | 16 | 2 | 1 |
| 2 | OutQ | 16 | 2 | 1 |

Description
Up to 16 OutPorts can be set and the maximum WordLength of each OutPort is 1024 bit.
Sign: 0 -> Unsigned; 1 -> Signed.
The number before each OutPort Name is its index for "DataOut", "ValidOut" and "ReadyIn" bus port.

Then you can click FPGA1 tab to configure FPGA interface of FPGA1. You need select "Copy Settings From FPGA0" for Configuration Mode, as we want to use the same FPGA design for both FPGA0 and FPGA1. Then all configurations are copied from FPGA0 tab, you just need click "Config Register" in FPGA1 tab to give Value of FPGA1 Register as 1 to select HPF FIR for FPGA1.



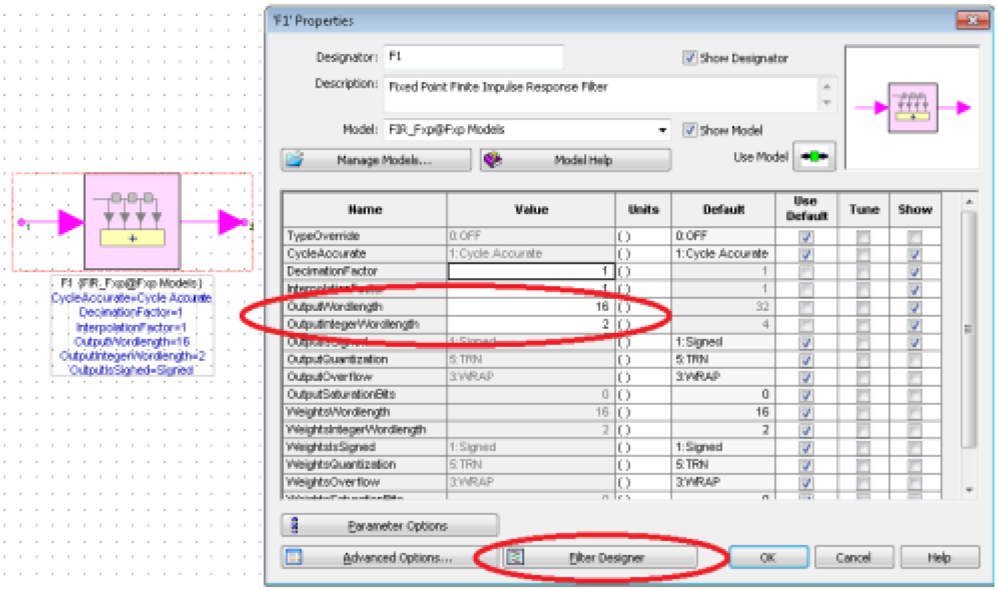
Designator: Subnetwork1
Description:
Model: M9703_TEMPLATE
Configuration Mode: Copy Settings From FPGA0
FPGA1 Settings

|   | Name | WordLength | Integer WordLength | Sign | Value |
|---|---|---|---|---|---|
| 1 | Reg1 | 1 | 1 | 0 | 1 |

Description
There is no limitation for the number of Register and the WordLength of each Register.
Sign: 0 -> Unsigned; 1 -> Signed.
The number before each Register Name is its index for "Register" bus port.

You have used the custom UI to configure the FPGA interfaces for M9703 FPGAs. Then you need create FPGA design for M9703 FPGA. Open M9703_FPGA0 subnet schematic, all interface ports are pre-configured, you need create FIR design in this schematic and connect the FIR design with the FPGA interfaces.

Go to "Hardware Design Library" in Part Selector and drag a FIR_Fxp model to M9703_FPGA0. Double click this model to open its parameters UI. You need modify its OutputWordLength, OutputIntegerWordLength and OutputIsSigned as <16, 2, Signed> to match your OutPort data format definition. Then click "Filter Designer" to open FIR coefficients design UI.



F1 {FIR_Fxp@Fxp Models}
CycleAccurate=Cycle Accurate
DecimationFactor=1
InterpolationFactor=1
OutputWordlength=16
OutputIntegerWordlength=2
OutputIsSigned=Signed

'F1' Properties

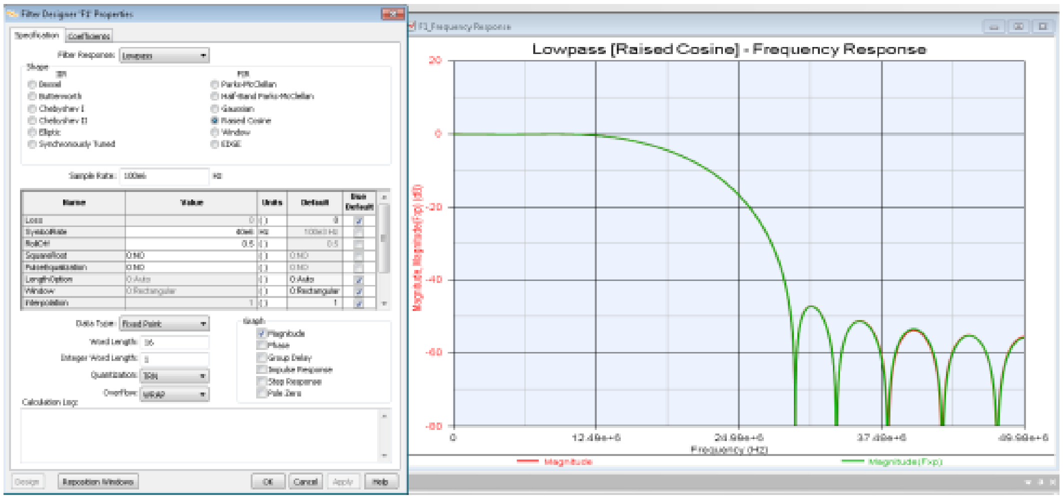
Designator: F1 ☑ Show Designator

Description: Fixed Point Finite Impulse Response Filter

Model: FIR_Fxp@Fxp Models ☑ Show Model

| Name | Value | Units | Default | Use Default | Tune | Show |
|---|---|---|---|---|---|---|
| TypeOverride | 0:OFF | ( ) | 0:OFF | ☑ | ☐ | ☐ |
| CycleAccurate | 1:Cycle Accurate | ( ) | 1:Cycle Accurate | ☑ | ☐ | ☑ |
| DecimationFactor | 1 | ( ) | 1 | ☐ | ☐ | ☑ |
| InterpolationFactor | 1 | ( ) | 1 | ☐ | ☐ | ☑ |
| OutputWordlength | 16 | ( ) | 32 | ☐ | ☐ | ☑ |
| OutputIntegerWordlength | 2 | ( ) | 4 | ☐ | ☐ | ☑ |
| OutputIsSigned | 1:Signed | ( ) | 1:Signed | ☑ | ☐ | ☑ |
| OutputQuantization | 5:TRN | ( ) | 5:TRN | ☑ | ☐ | ☐ |
| OutputOverflow | 3:WRAP | ( ) | 3:WRAP | ☑ | ☐ | ☐ |
| OutputSaturationBits | 0 | ( ) | 0 | ☑ | ☐ | ☐ |
| WeightsWordlength | 16 | ( ) | 16 | ☑ | ☐ | ☐ |
| WeightsIntegerWordlength | 2 | ( ) | 2 | ☑ | ☐ | ☐ |
| WeightsIsSigned | 1:Signed | ( ) | 1:Signed | ☑ | ☐ | ☐ |
| WeightsQuantization | 5:TRN | ( ) | 5:TRN | ☑ | ☐ | ☐ |
| WeightsOverflow | 3:WRAP | ( ) | 3:WRAP | ☑ | ☐ | ☐ |
| WeightsSaturationBits | 0 | ( ) | 0 | ☑ | ☐ | ☐ |

In filer coefficients design UI, configure the parameters as shown below for low pass filter:



Filter Designer 'F1' Properties

Filter Response: Lowpass

Shape

IIR
○ Bessel
○ Butterworth
○ Chebyshev I
○ Chebyshev II
○ Elliptic
○ Synchronously Tuned

FIR
○ Parks-McClellan
○ Half-Band Parks-McClellan
○ Gaussian
● Raised Cosine
○ Window
○ EDGE

Sample Rate: 100e6 Hz

| Name | Value | Units | Default | Use Default |
|---|---|---|---|---|
| Loss | 0 | ( ) | 0 | ☑ |
| SymbolRate | 40e6 | Hz | 100e3 Hz | ☐ |
| RollOff | 0.5 | ( ) | 0.5 | ☐ |
| SquareRoot | 0:NO | ( ) | 0:NO | ☐ |
| PulseEqualization | 0:NO | ( ) | 0:NO | ☐ |
| LengthOption | 0:Auto | ( ) | 0:Auto | ☑ |
| Window | 0:Rectangular | ( ) | 0:Rectangular | ☑ |
| Interpolation | 1 | ( ) | 1 | ☑ |

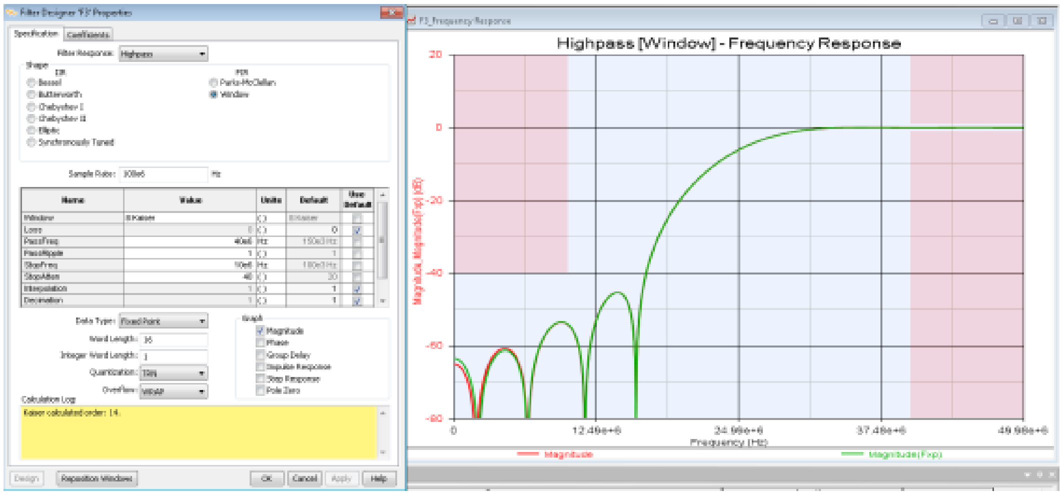
Data Type: Fixed Point
Word Length: 16
Integer Word Length: 1
Quantization: TRN
Overflow: WRAP

Graph
☑ Magnitude
☐ Phase
☐ Group Delay
☐ Impulse Response
☐ Step Response
☐ Pole Zero

Calculation Log:

After clicking OK, you can save the coefficients for FIR_Fxp model. You can copy the FIR_Fxp model and paste it to M9703_FPGA0, then you have the FIR filters with the same coefficients for I and Q data paths.

In the same way, you can design coefficients for high pass filter as below parameter configuration:



Filter Designer 'F3' Properties

Specification | Coefficients

Filter Response: Highpass

Shape

IIR

○ Bessel
○ Butterworth
○ Chebyshev I
○ Chebyshev II
○ Elliptic
○ Synchronously Tuned

FIR

○ Parks-McClellan
● Window

Sample Rate: 100e6 Hz

| Name | Value | Units | Default | Use Default |
|---|---|---|---|---|
| Window | 8 Kaiser | ( ) | 8 Kaiser | ☐ |
| Loss | 0 | ( ) | 0 | ☑ |
| PassFreq | 40e6 | Hz | 150e3 Hz | ☐ |
| PassRipple | 1 | ( ) | 1 | ☐ |
| StopFreq | 10e6 | Hz | 180e3 Hz | ☐ |
| StopAtten | 40 | ( ) | 20 | ☐ |
| Interpolation | 1 | ( ) | 1 | ☑ |
| Decimation | 1 | ( ) | 1 | ☑ |

Data Type: Fixed Point

Word Length: 16

Integer Word Length: 1

Quantization: TRN

Overflow: WRAP

Graph

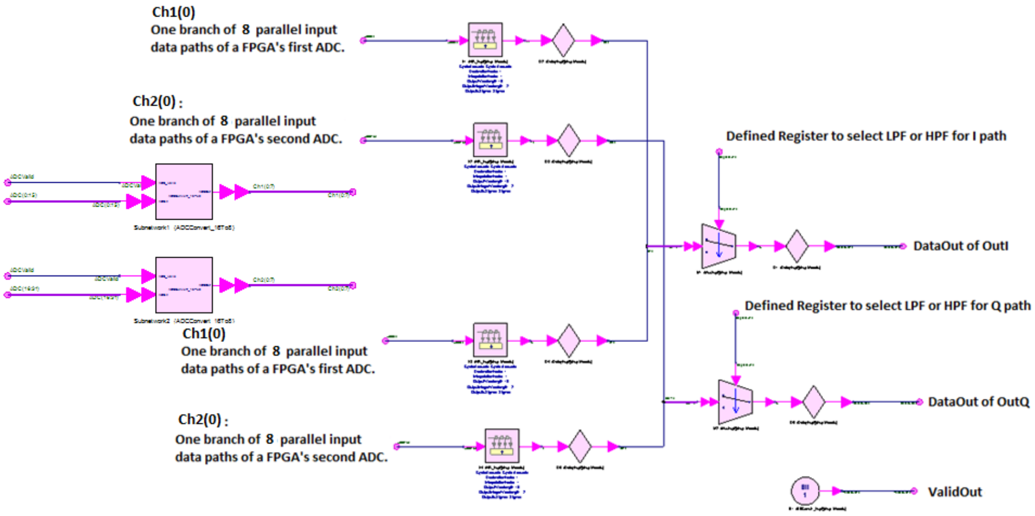
☑ Magnitude
☐ Phase
☐ Group Delay
☐ Impulse Response
☐ Step Response
☐ Pole Zero

Calculation Log

Kaiser calculated order: 14.

Design | Reposition Windows | OK | Cancel | Apply | Help

You can refer to M9703_FPGA0 in "Help > Open Examples > Hardware Design > M9703_FDK > M9703_FDK_Tutorial > tutorial.wsv" for the whole FPGA design.

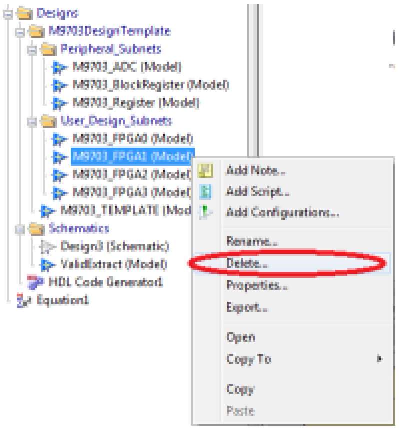


Ch1(0)
One branch of 8 parallel input
data paths of a FPGA's first ADC.
Ch2(0) :
One branch of 8 parallel input
data paths of a FPGA's second ADC.
Defined Register to select LPF or HPF for I path
DataOut of OutI
Defined Register to select LPF or HPF for Q path
Ch1(0)
One branch of 8 parallel input
data paths of a FPGA's first ADC.
Ch2(0) :
One branch of 8 parallel input
data paths of a FPGA's second ADC.
DataOut of OutQ
ValidOut

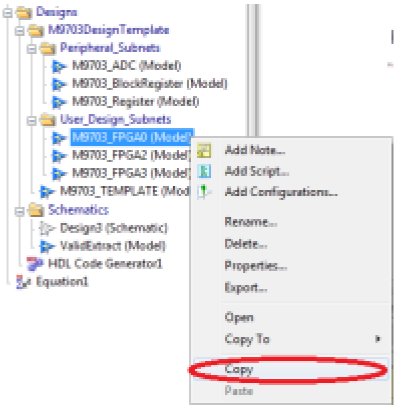
Please note:

1. Subnet ADCConvert_16To8 is used to convert 16 parallel input streams to 8 parallel streams and make all 8 parallel data valid at every clock cycle. This design just process one of 8 parallel data, it's a 8 times decimation.
2. Because the wordlength of both OutA and OutB are 16 bits, the total bit width of all OutPorts is 32 bits, which is less than 124 bits (the effect throughput to DDR3 SDRAM). So ReadyIn of OutA and OutB will always be 1 and we can simply the design without connecting ReadyIn input ports.

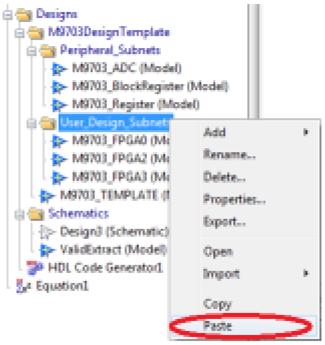
After the design of FPGA0, we can copy the design for FPGA1, as we want to use the same design for both FPGA0 and FPGA1.
In workspace tree, you can delete the existing M9703_FPGA1 at first.



Then copy M9703_FPGA0 in workspace tree.



Then paste to User_Design_Subnets folder in workspace tree.

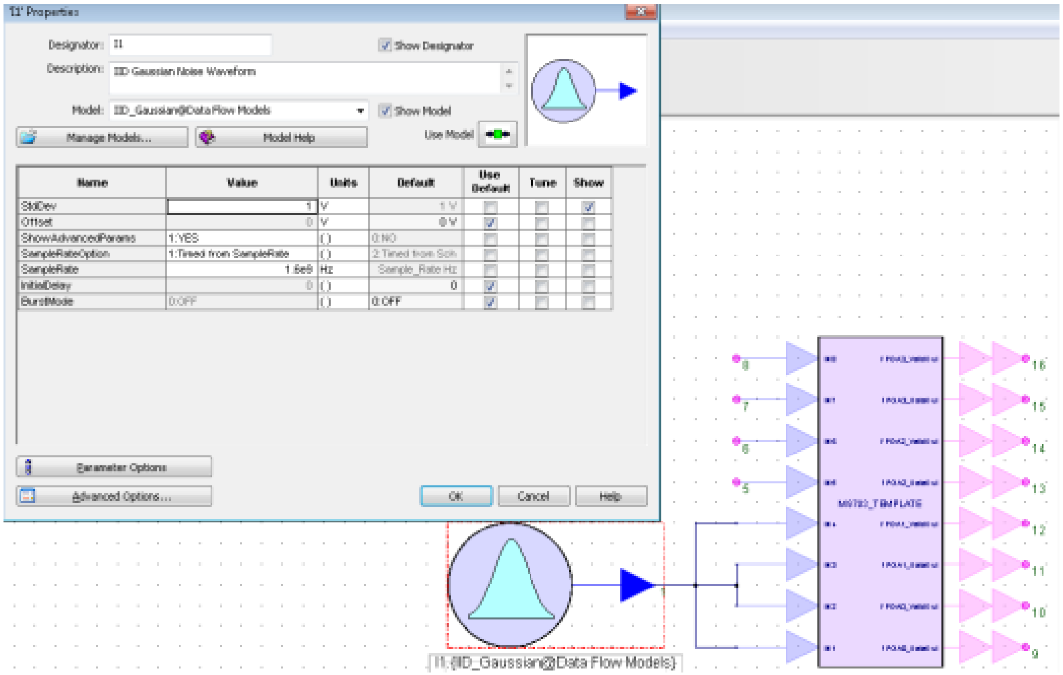


At last, you need rename the new pasted subnet as "M9703_FPGA1".
Then you have finished the FPGA design in M9703 Design Template.

## Software Simulation

After FPGA design entry in M9703 Design Entry, we can create a testbench in SystemVue top level schematic to test the FIR design by software simulation.

We need find IID_Gaussian model in "Algorithm Design" library in part selector and drag it to top level schematic. Then we can set its parameters and connect its output to IN1~IN4 of M9703 Design Template subnet to get a Gaussian white noise test signal for our FPGA design. Please note that you have to set the SampleRate of the IID_Gaussian model as 1.6GHz to match your configuration in the custom UI of M9703 Design Template.

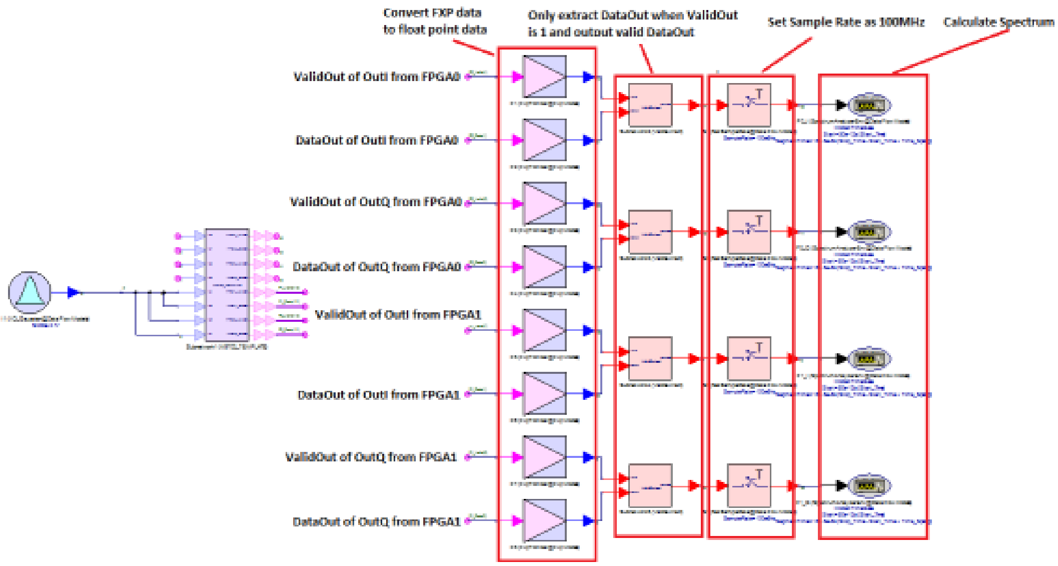


'I1' Properties

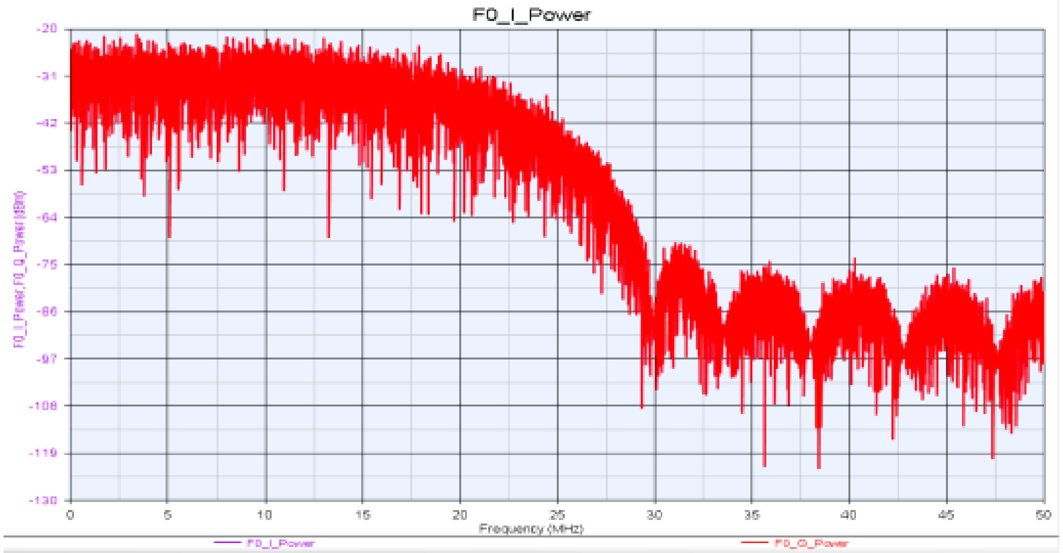
Designator: I1 ☑ Show Designator
Description: IID Gaussian Noise Waveform
Model: IID_Gaussian@Data Flow Models ☑ Show Model

| Name | Value | Units | Default | Use Default | Tune | Show |
|---|---|---|---|---|---|---|
| StdDev | 1 | V | 1 V | □ | □ | ☑ |
| Offset | 0 | V | 0 V | ☑ | □ | □ |
| ShowAdvancedParams | 1:YES | () | 0:NO | □ | □ | □ |
| SampleRateOption | 1:Timed from SampleRate | () | 2:Timed from Sch | □ | □ | □ |
| SampleRate | 1.6e9 | Hz | Sample_Rate Hz | □ | □ | □ |
| InitialDelay | 0 | () | 0 | ☑ | □ | □ |
| BurstMode | 0:OFF | () | 0:OFF | ☑ | □ | □ |

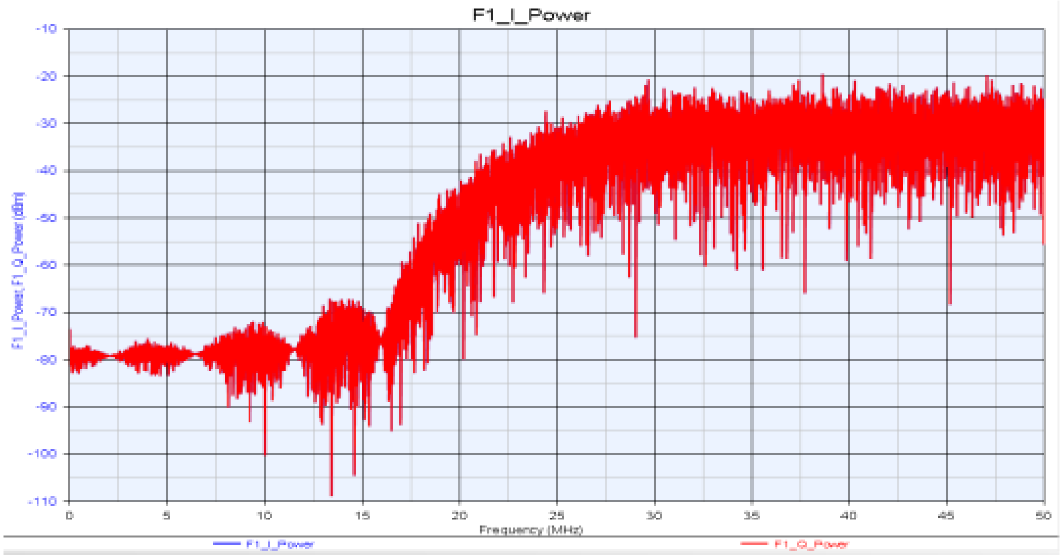
Then you need refer to the top level schematic of "Help > Open Examples > Hardware Design > M9703_FDK > M9703_FDK_Tutorial > tutorial.wsv" to connect models like below to get the spectrum of IQ signals from both FPGA0 and FPGA1.


Convert FXP data to float point data
Only extract DataOut when ValidOut is 1 and outout valid DataOut
Set Sample Rate as 100MHz
Calculate Spectrum
ValidOut of OutI from FPGA0
DataOut of OutI from FPGA0
ValidOut of OutQ from FPGA0
DataOut of OutQ from FPGA0
ValidOut of OutI from FPGA1
DataOut of OutI from FPGA1
ValidOut of OutQ from FPGA1
DataOut of OutQ from FPGA1

Please be careful for the net labels of wires, so that you can get the correct wires connection. Besides, because the actual data rates of filtered IQ signals are 100MHz, you need use "SetSampleRate" models to set their sample rate before Spectrum models. When you connect the top level testbench well, you can run SystemVue simulation. Then you can draw the spectrums of I and Q filtered signals from FPGA0 in figure "DF1_Design3_F0_I_Power". As you have set Register of FPGA0 as 0 to select Low Pass Filter, you can see the results as below figure.

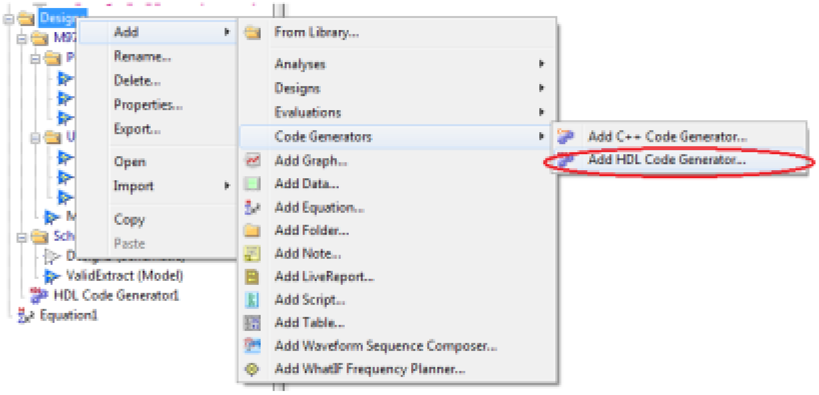


You can draw the spectrums of I and Q filtered signals from FPGA1 in figure "DF1_Design3_F1_I_Power". As you have set Register of FPGA1 as 1 to select High Pass Filter, you can see the results as below figure.

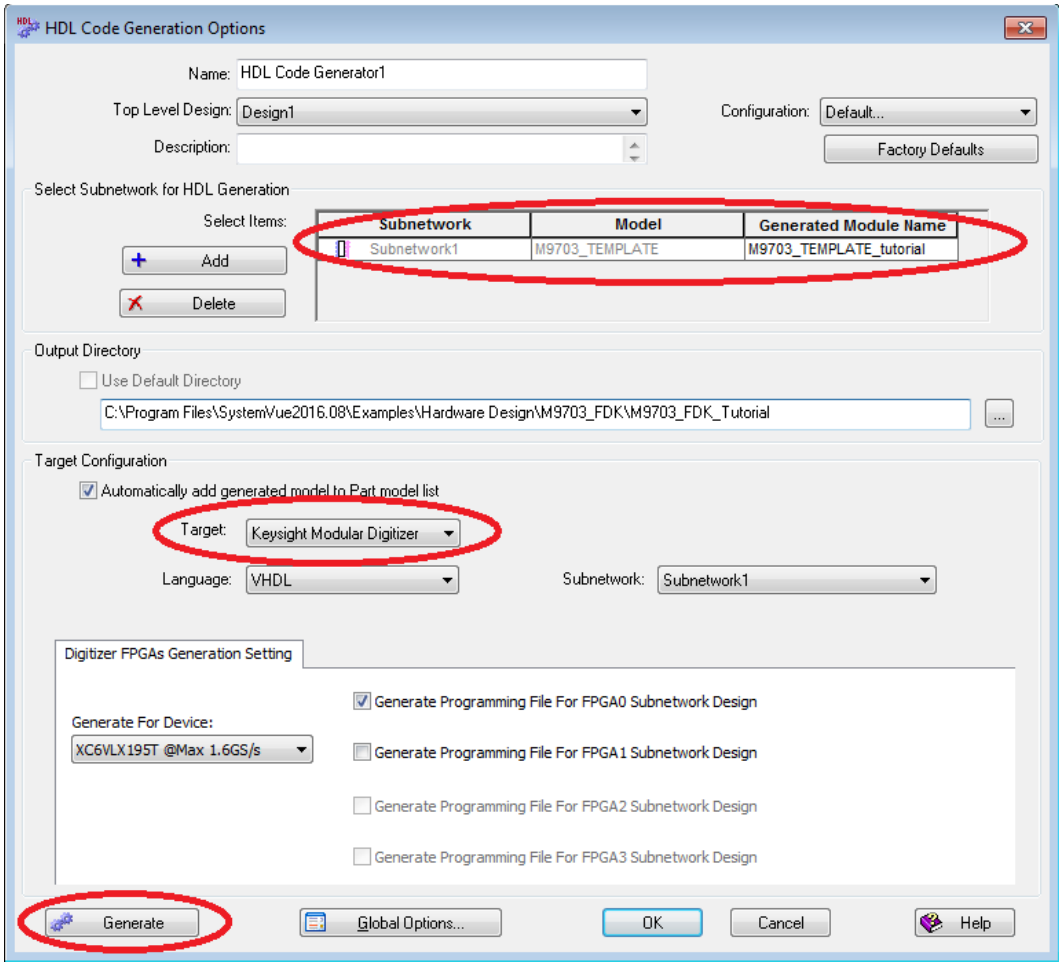


## Generate FPGA Programming File

After you finish software simulation to verify the function of your FPGA design, you can add a "HDL Code Generator" to launch the automatic FPGA programming file generation flow.
You can right click a folder in workspace tree and select to Add HDL Code Generator.



In added HDL Code Generator, you can click Add button to select your M9703_TEMPLATE subnet for FPGA image generation. Then select to use "Keysight Modular Digitizer" mode and click "Generate" button. SystemVue will launch an automatic flow to generate the final FPGA programming file.



Name: HDL Code Generator1

Top Level Design: Design1

Configuration: Default...

Description:

Select Subnetwork for HDL Generation

Select Items:

| Subnetwork | Model | Generated Module Name |
|---|---|---|
| Subnetwork1 | M9703_TEMPLATE | M9703_TEMPLATE_tutorial |

Output Directory

□ Use Default Directory

C:\Program Files\SystemVue2016.08\Examples\Hardware Design\M9703_FDK\M9703_FDK_Tutorial

Target Configuration

☑ Automatically add generated model to Part model list

Target: Keysight Modular Digitizer

Language: VHDL

Subnetwork: Subnetwork1

Digitizer FPGAs Generation Setting

Generate For Device: XC6VLX195T @Max 1.6GS/s

☑ Generate Programming File For FPGA0 Subnetwork Design

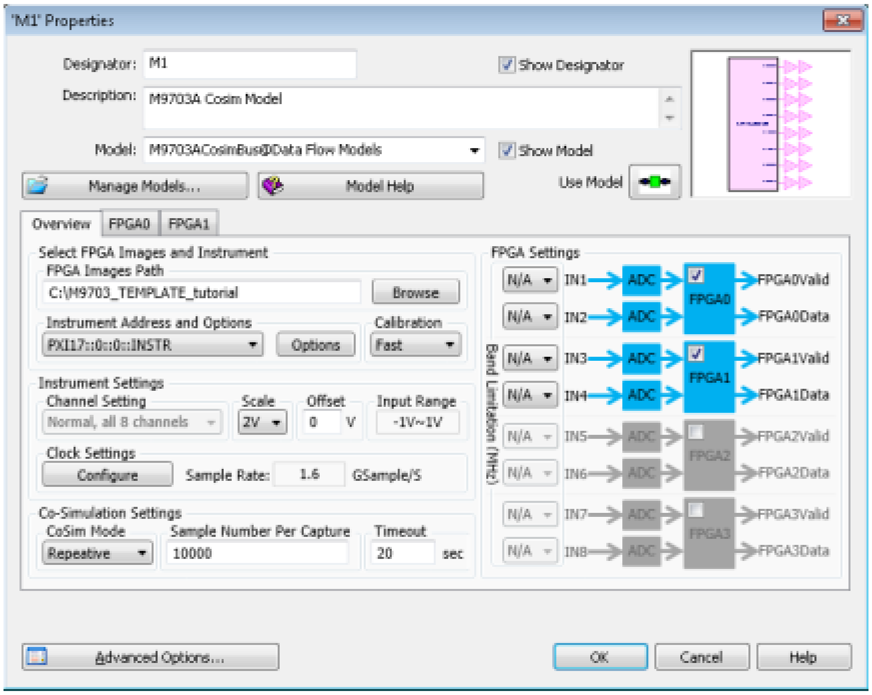
□ Generate Programming File For FPGA1 Subnetwork Design

□ Generate Programming File For FPGA2 Subnetwork Design

□ Generate Programming File For FPGA3 Subnetwork Design

## M9703 Instrument Co-Simulation

After your FPGA programming file is generated, you can find M9703ACosimBus model in "Algorithm Design" library and drag it to the top level testbench schematic. Double click the model to open its GUI and configure as below:



'M1' Properties

Designator: M1

☑ Show Designator

Description: M9703A Cosim Model

Model: M9703ACosimBus@Data Flow Models

☑ Show Model

Select FPGA Images and Instrument

FPGA Images Path: C:\M9703_TEMPLATE_tutorial

Instrument Address and Options: PXI17::0::0::INSTR

Calibration: Fast

Instrument Settings

Channel Setting: Normal, all 8 channels

Scale: 2V

Offset: 0 V

Input Range: -1V~1V

Clock Settings

Sample Rate: 1.6 GSample/S

Co-Simulation Settings

CoSim Mode: Repeative

Sample Number Per Capture: 10000

Timeout: 20 sec

FPGA Settings

☑ FPGA0

☑ FPGA1

□ FPGA2

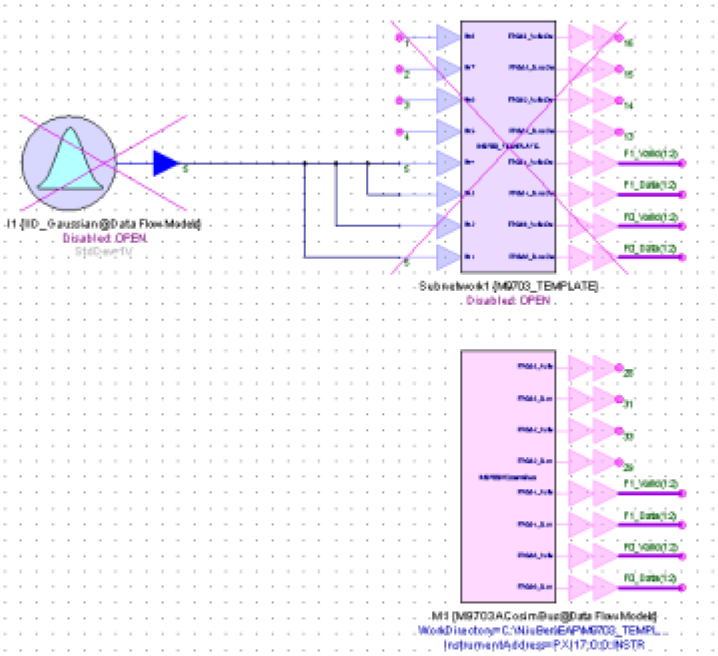
□ FPGA3

Uncheck FPGA2 and FPGA3 in FPGA Settings area, as we just use FPGA0 and FPGA1 in this example.

Select the path of your generated FPGA image in parameter "FPGA Image Path".

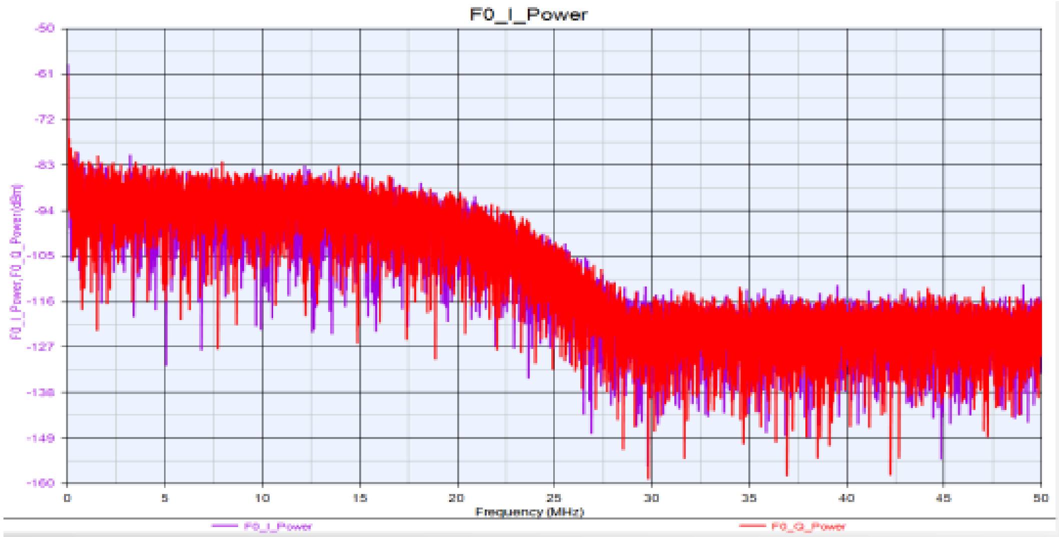
You have to install Agilent IO Library and MD1 driver and have M9703 instrument that powers on and connects with your computer. Otherwise, you can't see any instrument listed in this UI for selection.

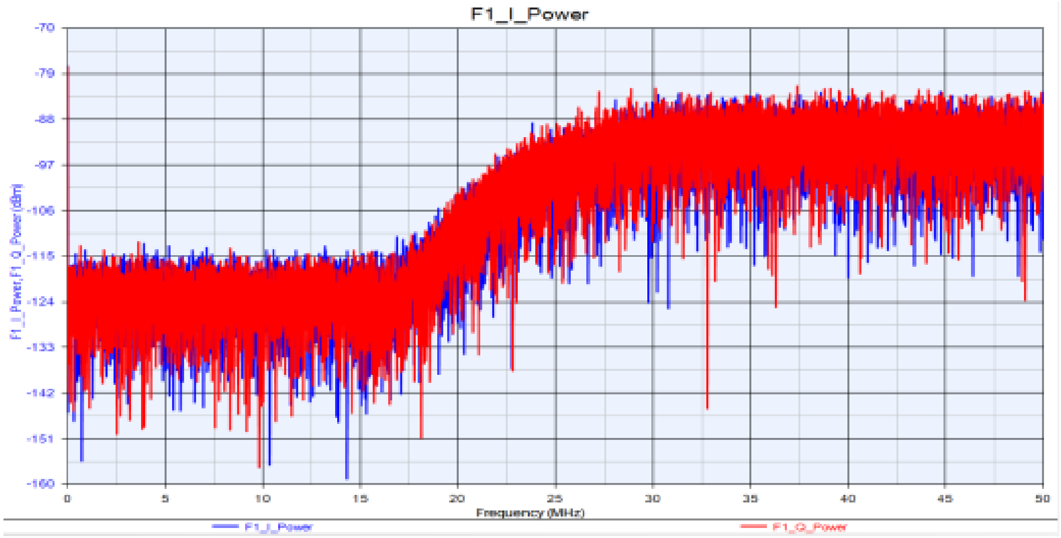
Then in the top level testbench, you can connect the M9703ACosimBus model with the same wires as M9703 Design Template and disable Gaussian noise model and M9703 Design Template subnet, as shown below:


I1 {IID_Gaussian@Data Flow Models}
Disabled: OPEN
StdDev=1V
Subnetwork1 {M9703_TEMPLATE}
Disabled: OPEN
F1_Valid(1:2)
F1_Data(1:2)
F0_Valid(1:2)
F0_Data(1:2)
M1 {M9703ACosimBus@Data Flow Models}
WorkDirectory=C:\NiuBen\EAP\M9703_TEMPL...
InstrumentAddress=PXI17::0::0::INSTR

When you connect the M9703ACosimBus model well, you can run the SystemVue simulation. M9703ACosimBus model will find the actual M9703 instrument and do the co-simulation. You can leave IN1~IN4 of M9703 instrument open, then it will get the filtering results for the floor noise.

When the co-simulation is finished, you can observe the spectrum in the same figures:





## SystemVue U5303A FPGA Design Flow

### Required Hardware and Software

Following is the list of required hardware and software.

#### Required Hardware

- U5303A PCIe 12-bit High-Speed Digitizer with On-Board Signal Processing
  The U5303A hardware should have -SR1 or -SR2 option for 1.0GS/s or 1.6 GS/s sample rate. The U5303A hardware with -SR0 option for 500MS/s sample rate is NOT supported by this flow, as it has a different hardware configuration.
  The U5303A must be configured with –FDK option to enable its FPGA programming capability. For U5303A hardware installation, see U5303A Startup Guide.
- A PC with PCIe x8 Gen2 slot. The U5303A instrument can be installed in PCIe x8 Gen2 slot directly.

#### Required Software

- SystemVue 2016.08 or later version (W1462 license bundle is required)
- Keysight IO Libraries Suite: version 17.2 update 2 or later version
- Keysight MD2 High-Speed Digitizer Instrument Drivers: 1.12 or later Windows version is required.
- Xilinx ISE: version 14.7 or later

**NOTE** SystemVue U5303A FPGA design flow consists of FPGA design entry and software simulation, FPGA programming file generation and U5303A instrument co-simulation.

For FPGA design entry and software simulation, only SystemVue 2016.08 or later version is required. And SystemVue can be installed on any PC no matter whether U5303A instrument is installed on it or not. Keysight IO Libraries Suite, Keysight MD2 High-Speed Digitizer Instrument Driver and Xilinx ISE are not required for this step.

For FPGA programming file generation, SystemVue and Xilinx ISE are required. SystemVue and Xilinx ISE can be installed on any PC no matter whether U5303A instrument is installed on it or not. Keysight IO Libraries Suite and Keysight MD2 High-Speed Digitizer Instrument Driver are not required for this step.

For U5303A instrument co-simulation, SystemVue, Keysight IO Libraries Suite and Keysight MD2 High-Speed Digitizer Instrument Driver are required. They must be installed on the same PC that U5303A instrument is installed on. Xilinx ISE is not required for this step.

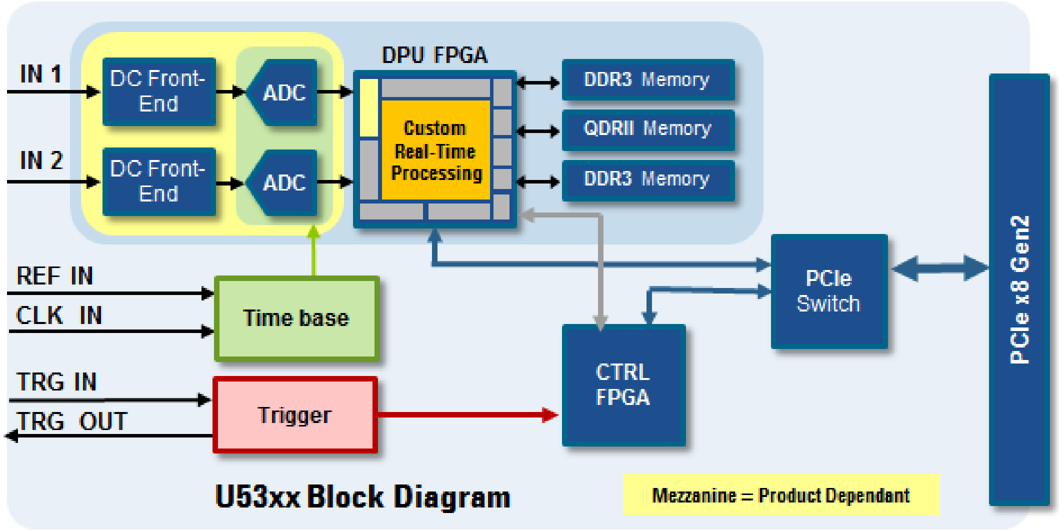
So you can do FPGA design entry, software simulation and FPGA programming file generation on any PC without U5303A instrument installation. After the FPGA programming file is generated, you can deploy the generated FPGA programming file to run U5303A instrument co-simulation on the PC with U5303A instrument installation.

## Overview of SystemVue U5303A FDK Design Flow

### Overview of U5303A High-Speed Digitizer

Based on the PCIe, the U5303A is an 2-channel, 12-bit wideband digital receiver /digitizer, able to capture signals from DC up to 2 GHz at 1.6GS/s. A channel interleaving capability allows waveform acquisition at up to 3.2 GS/s with exceptional measurement accuracy.

U5303A Hardware Diagram

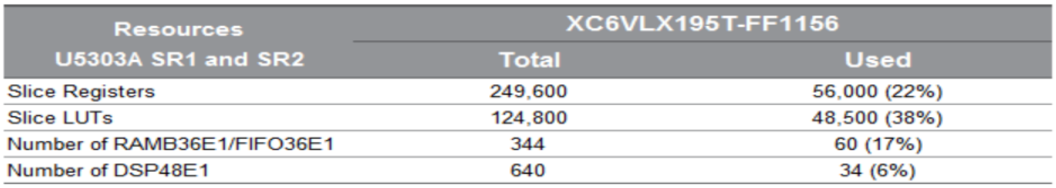


IN1 and IN2 are 2 input ports from U5303A front panel. They go through DC Front-End that are electable analog low pass filter (pass band frequency is 600MHz). Then the analog input signals are fed into respective ADCs to convert to digital signals.
The digitizer architecture could be simply understood by the following data stream scheme: For each analog channel, a front-end electronics converts the customer analog signal into a digital stream of data (ADC). These digital streams are captured by a processing FPGA (further called DPU FPGA). The custom real-time processing usually provides a data reduction scheme to only store processed results into a temporary memory buffer. Then the host application retrieves these processed data through the PCIe data and control bus whose sustained data bandwidth is significantly lower than the raw data bandwidth.

There is one DPU FPGA on a U5303A. It's Xilinx XC6VLX195T -FF1156 -2 FPGA.

- IN1 and IN2 are inputted into DPU FPGA0.

The DPU FPGA opens a partial area for custom real-time processing. Below table shows the FPGA resources are used for U5303A infrastructure, the remaining FPGA resources can be used for custom real-time processing.



| Resources | XC6VLX195T-FF1156 | |
|---|---|---|
| U5303A SR1 and SR2 | Total | Used |
| Slice Registers | 249,600 | 56,000 (22%) |
| Slice LUTs | 124,800 | 48,500 (38%) |
| Number of RAMB36E1/FIFO36E1 | 344 | 60 (17%) |
| Number of DSP48E1 | 640 | 34 (6%) |

And the DPU FPGA has the following physical interface:

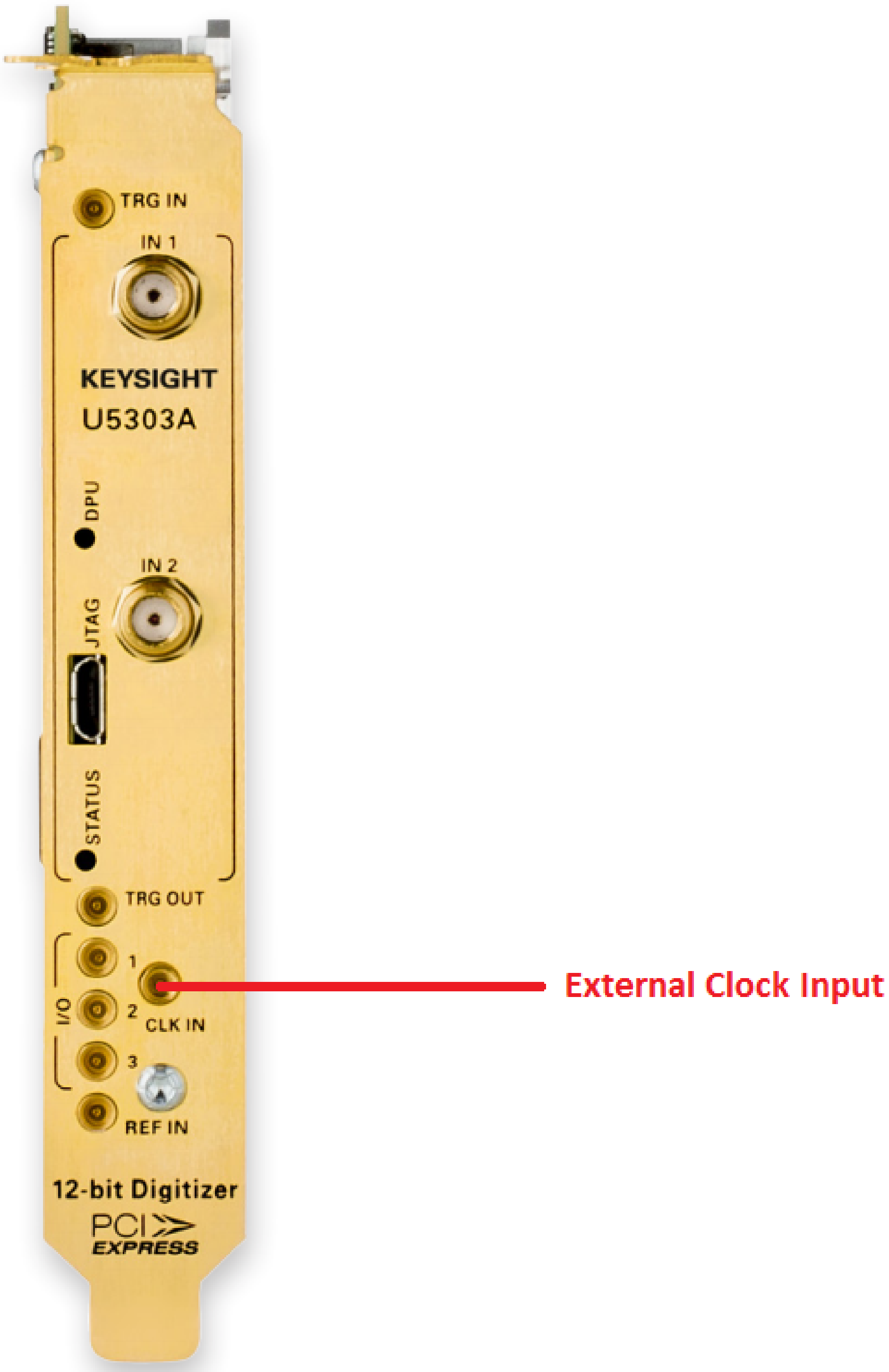
- ADC parallel data streams input
- Connectivity with two DDR3 memory and one QDRII memory
- Inter FPGAs data stream connectivity
- PCIe connectivity with backplane via PCIe switch
- Control signals from CTRL FPGA

In SystemVue U5303A FPGA design flow, these physical interfaces are transparent to users. SystemVue will provide some algorithm-level interfaces, such as OutPort, Register and BlockRegister, so that users can use these interfaces easily without caring about the format and protocol of the physical interfaces.
The DPU FPGA is connected with two DDR3 SDRAM memories. The DDR3 SDRAM memories can be used to buffer the real-time processed data for PCIe transfer with embedded controller or PC. The memory size of each DDR3 SDRAM memory is indicated by U5303A option:
Option
-M10: The size of each DDR3 SDRAM is 512M Byte. There are 2 DDR3 SDRAM for the DPU FPGA, so the total memory size of one U5303A is 1G Byte.
-M20: The size of each DDR3 SDRAM is 1G Byte. There are 2 DDR3 SDRAM for the DPU FPGA, so the total memory size of one U5303A is 2G Byte.
-M40: The size of each DDR3 SDRAM is 2G Byte. There are 2 DDR3 SDRAM for the DPU FPGA, so the total memory size of one U5303A is 4G Byte.

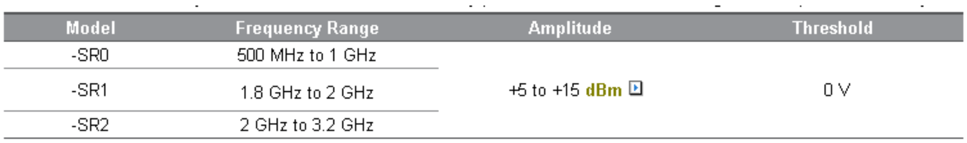
### U5303A DPU FPGA Clock

All custom real-time processing are in the same system clock domain. The system clock is synchronized with DPU FPGA input clock source. U5303A provides three kinds of clock sources for its DPU FPGA:

- Internal clock
  The internal clock is generated from U5303A module hardware.
- External clock
  The instruments external clock input connector (as shown in below figure) is selected as the source.

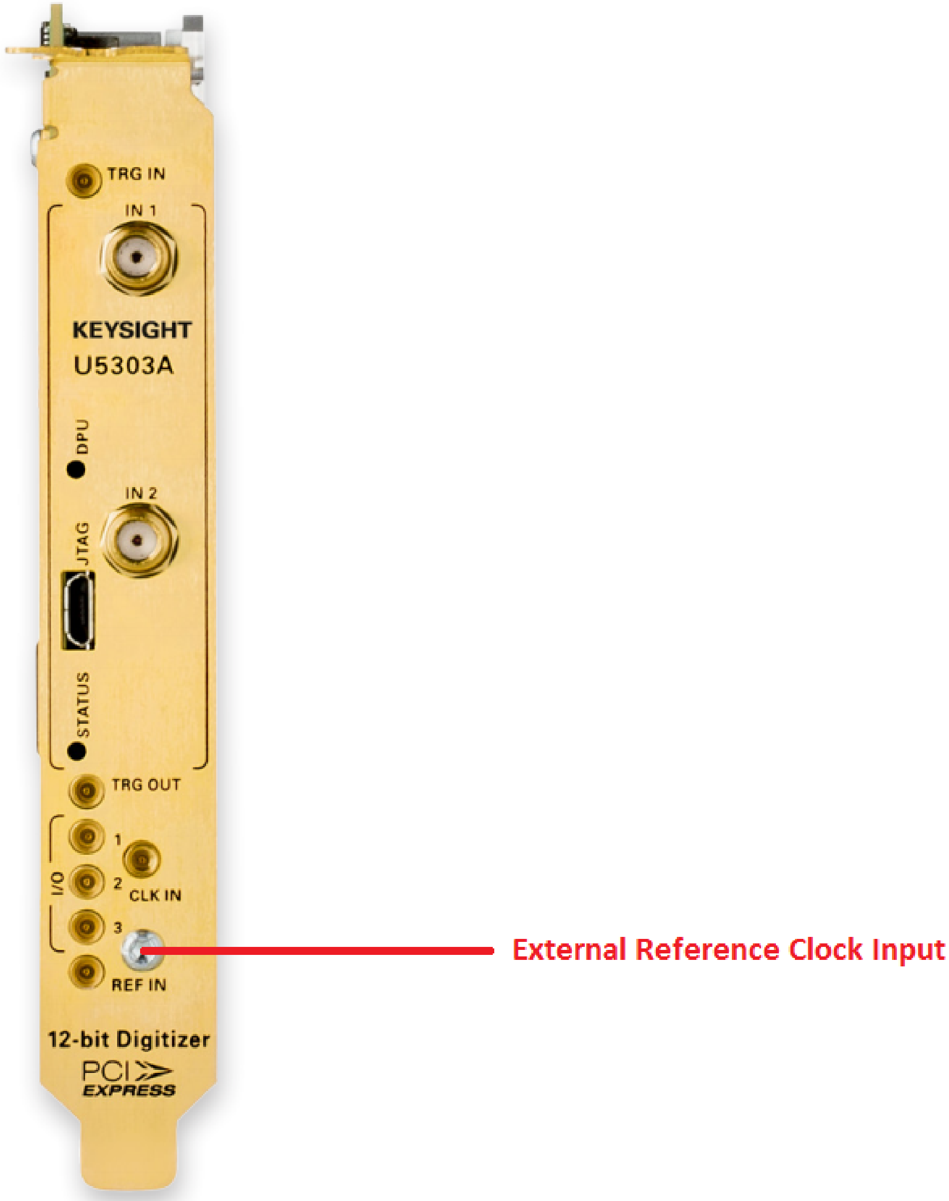


The External Clock may be used to vary the sampling rate of the digitizer, it must be continuously present if selected for the digitizer to operate correctly. The input is 50Ω AC coupled. The requirement of external clock signal is:



| Model | Frequency Range | Amplitude | Threshold |
|---|---|---|---|
| -SR0 | 500 MHz to 1 GHz | +5 to +15 dBm | 0 V |
| -SR1 | 1.8 GHz to 2 GHz | | |
| -SR2 | 2 GHz to 3.2 GHz | | |

- External reference clock
  The instruments external reference clock input connector (as shown in below figure) is selected as the source.



For applications that require greater timing precision and long-term stability than is obtainable from the internal clock, a 100 MHz Reference signal can be used. The input is 50 **Ω** terminated and AC coupled.

If your input is not at exactly the specified value, you must remember to compensate for the difference in your application since the digitizer and the driver have no way to know about such deviations.

| Parameter | Value | Tolerance |
|---|---|---|
| Nominal Frequency | 100 MHz | ±100 kHz |
| Amplitude | -3 to +3 dBm | |

**If synchronization between several digitizers is required, the reference signal should be applied to all of them.**

All custom real-time processing on DPU FPGA run on DPU FPGA system clock. When clock source is selected as Internal Clock or External Reference Clock, the DPU FPGA system clock frequency is fixed. When U5303A is configured as 1GS/s option (-SR1), the DPU FPGA system clock is 125MHz. All custom real-time processing and ADC captured data in DPU FPGAs run on the 125MHz DPU FPGA system clock. When U5303A is configured as 1.6GS/s option (-SR2), the DPU FPGA system clock is 200MHz. All custom real-time processing and ADC captured data in DPU FPGAs run on the 200MHz system clock.

When clock source is selected as External Clock, the DPU FPGA system clock frequency varies as the frequency of External Clock input. The DPU FPGA system clock frequency is equal to one sixteenth of the frequency of External Clock input. When U5303A is configured as 1GS/s option (-SR1), the frequency of External Clock input is in range 1.8GHz ~ 2.0GHz, so the DPU FPGA system clock frequency is in range 112.5MHz ~ 125MHz. When U5303A is configured as 1.6GS/s option (-SR2), the frequency of External Clock input is in range 2.0GHz ~ 3.2GHz, so the DPU FPGA system clock frequency is in range 125MHz ~ 200MHz.

DPU FPGA system clock frequency:

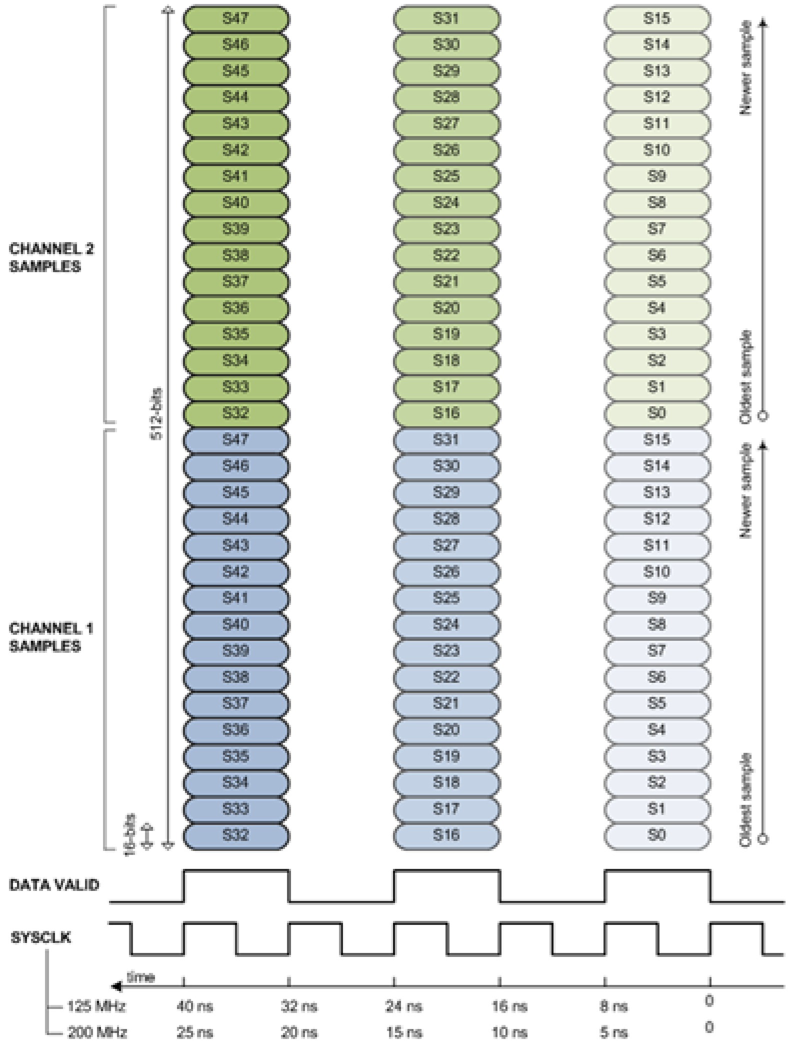
| | 1GS/s option (-SR1) | 1.6GS/s option (-SR2) |
|---|---|---|
| Clock Source:<br>**Internal Clock** Or<br>**External Reference Clock** | 125MHz | 200MHz |
| Clock Source:<br>**External Clock** | External Clock Frequency / 16<br>External Clock Range:<br>1.8GHz ~ 2.0GHz | External Clock Frequency / 16<br>External Clock Range:<br>2.0GHz ~ 3.2GHz |

### ADC Parallel Input Streams of U5303A DPU FPGA

For ADC input to DPU FPGA, the incoming ADC data streams are de-multiplexed into multiple parallel data streams at lower data rate and then perform some pre-correction of the sampling defaults (e.g. linearity correction, interleave mismatches correction). So for each sample, the 12-bit ADC data are extended to 16-bit after the pre-correction.

In DPU FPGA, the parallel ADC input streams are all sampled based on DPU FPGA system clock.

For normal sample mode (1.0GS/s for SR1 option and 1.6GS/s for SR2 option), each DPU FPGA receives two channels ADC input streams. Below figure shows how the ADC input streams are inputted into the custom real-time processing part.

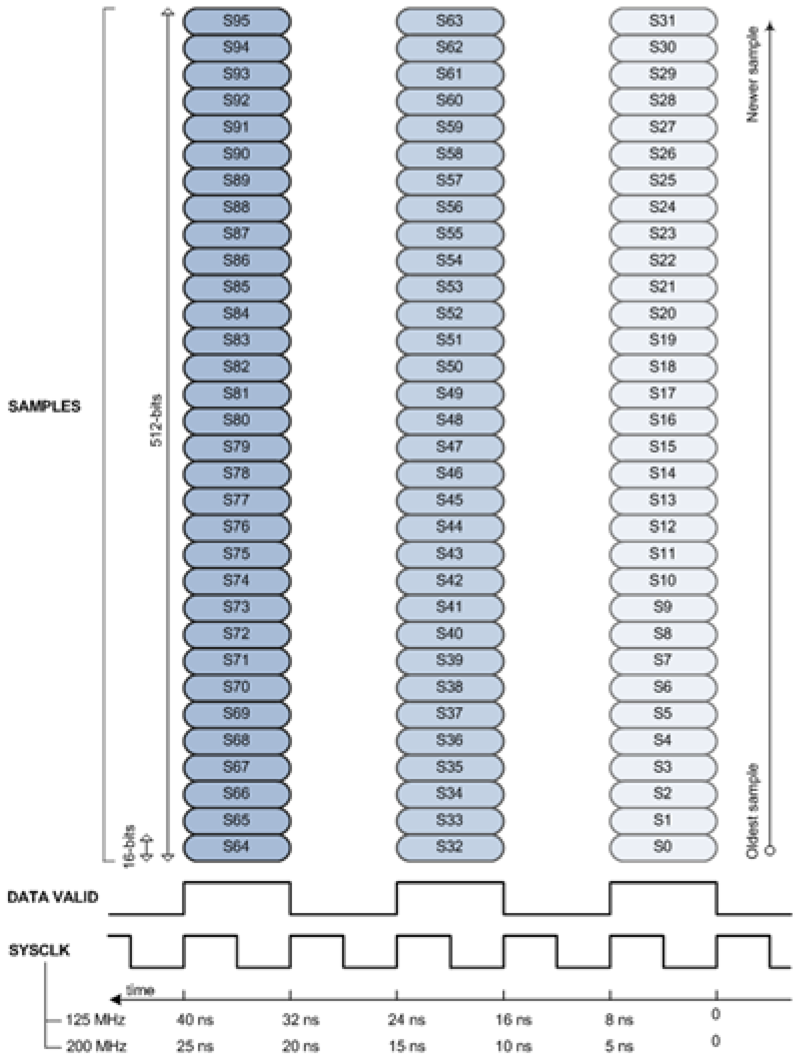


Each ADC channel has 16 parallel streams. Every two system clock cycles, there are valid ADC samples. The DATA VALID signal in above figure indicates when valid ADC samples arrive.

For 1.0GS/s SR1 option, its DPU FPGA system clock is 125MHz and valid samples arrive every two system clock cycles, so the total sample rate is: 16 parallel streams x 125MS / 2 = 1.0GS/s. For 1.6GS/s SR2 option, its DPU FPGA system clock is 200MHz and valid samples arrive every two system clock cycles, so the total sample rate is: 16 parallel streams x 200MS / 2 = 1.6GS/s.
Because each sample has 16-bit, every time the valid samples arrive, there are

(16 parallel samples/channel) x (16-bits/sample) x 2 channel = 512 bit

For interleaved sample mode (2.0GS/s for SR1 option and 3.2GS/s for SR2 option), each DPU FPGA receives one channel ADC input streams. Below figure shows how the ADC input streams are inputted into the custom real-time processing part.



Each ADC channel has 32 parallel streams. Every two system clock cycles, there are valid ADC samples. The DATA VALID signal in above figure indicates when valid ADC samples arrive.

For 2.0GS/s SR1 option, its DPU FPGA system clock is 125MHz and valid samples arrive every two system clock cycles, so the total sample rate is: 32 parallel streams x 125MS / 2 = 2.0GS/s. For 3.2GS/s SR2 option, its DPU FPGA system clock is 200MHz and valid samples arrive every two system clock cycles, so the total sample rate is: 32 parallel streams x 200MS / 2 = 3.2GS/s.
Because each sample has 16-bit, every time the valid samples arrive, there are

(32 parallel samples/channel) x (16-bits/sample) x 1 channel = 512 bit

### Overview of U5303A FPGA Design Flow

The SystemVue U5303A FPGA design flow consists of three steps: design entry and software simulation, U5303A FPGA programming file auto generation and U5303A instrument co-simulation with SystemVue.

- Design entry and software simulation
  SystemVue provides a hierarchical subnet template to ask users to configure U5303A four FPGAs and design FPGAs in a model-based environment. In

addition, SystemVue provides some software peripheral models to mimic the hardware behaviors of U5303A FPGA interface, so users can run a pure software simulation to debug and verify their FPGA design.

- U5303A FPGA programming file auto generation
  When the FPGA design in SystemVue schematic is verified by software simulation and is ready for generating FPGA programming file, SystemVue provides a "one-button-push" automatic flow to launch Xilinx ISE to generate final U5303A FPGA programming files in background.
- U5303A instrument co-simulation with SystemVue
  After the U5303A FPGA programming files are generated, SystemVue provides a U5303A instrument co-simulation model to connect SystemVue with U5303A hardware. This model can download U5303A FPGA programming file to DPU FPGAs, configure the user-defined FPGA registers and capture data output from U5303A DPU FPGA to SystemVue. Users can verify the custom real-time processing on real hardware in this step.

NOTE

After U5303A instrument co-simulation verification, users can use AgMD2 IVI-C or AgMD2 IVI-COM driver to create their own program to deploy the user-defined DPU FPGA programming file and control U5303A instrument without launching SystemVue.

AgMD2 IVI-C and AgMD2 IVI-COM driver are installed automatically when you install MD2 High-Speed Digitizer Instrument Drivers.

For details of programming using AgMD2 IVI-C and AgMD2 IVI-COM, please refer to the document of **IVI Driver Reference** . You will get this document after installing MD2 High-Speed Digitizer Instrument Drivers.

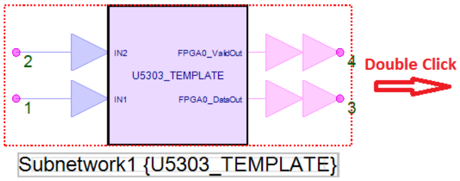
## Design Entry and Software Simulation

### U5303A Design Template

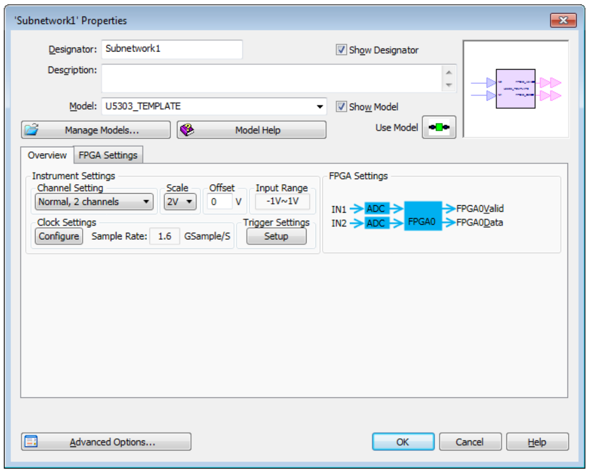
U5303A Design Template is a set of hierarchical subnets to model U5303A's hardware architecture. It mimics and simplifies the behavior of U5303A DPU FPGAs' interface, so that users can easily configure U5303A DPU FPGAs and enter model-based FPGA design under U5303A's hardware architecture. Then users can do software simulation with U5303A Design Template to debug and verify whether their real-time processing part works well under U5303A's hardware architecture before generating FPGA programming file.

When you drag-and-drop the top level subnet of U5303A Design Template from workspace tree to a schematic, you will get a U5303A Design Template model in the schematic. This model has a GUI to help you configure the U5303A interfaces and working mode easily.

U5303A Design Template Model



U5303A Design Template Model GUI



'Subnetwork1' Properties

Designator: Subnetwork1 ☑ Show Designator

Description:

Model: U5303_TEMPLATE ☑ Show Model

Manage Models... Model Help Use Model

Overview | FPGA Settings

Instrument Settings

Channel Setting: Normal, 2 channels; Scale: 2V; Offset: 0 V; Input Range: -1V~1V

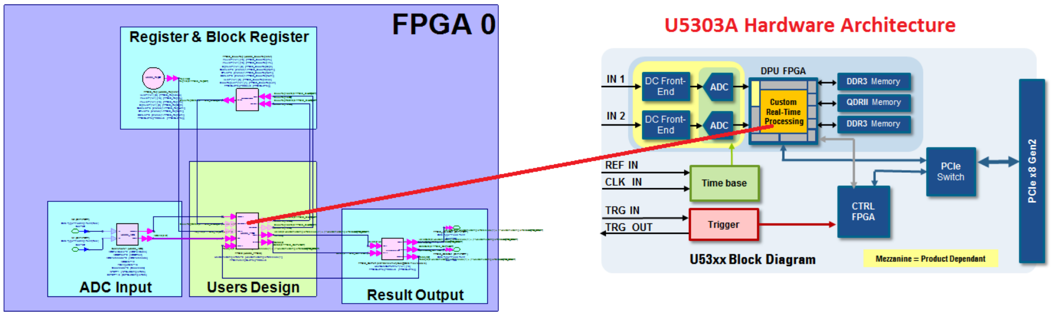
Clock Settings: Configure; Sample Rate: 1.6 GSample/S; Trigger Settings: Setup

FPGA Settings

Below figure shows the top-level subnet of U5303A Design Template. The DPU FPGA of U5303A is mimicked in the top-level subnet. For the mimicked DPU FPGA, there is a green area Users Design. This area is mapped to Custom real-time processing part on the corresponding DPU FPGA hardware. Users can enter their model-based FPGA design in the subnet of Users Design area.

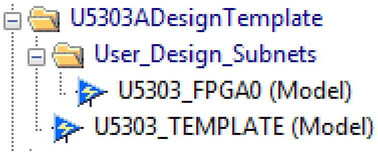
U5303A DPU FPGAs' physical interfaces, such as ADC Input, DDR3 Memory link and PCIe link, are simplified in the U5303A Design Template subnets. So users just need connect their own real-time processing part with these simplified interfaces in U5303A Design Template, instead of being familiar with the sophisticated physical interfaces of U5303A hardware. It will save a lot of development time. Then when you generate FPGA programming file, SystemVue will provide an automatic flow to connect users' real-time processing part design with the physical interfaces well and generate the final FPGA programming file.

Top-Level Subnet of U5303A Design Template


U5303A Hardware Architecture

## U5303A Design Template Hierarchy

You can find the subnets architecture of U5303A Design Template in SystemVue workspace tree from example workspace " *<SystemVue installation dir>* \Examples\Hardware Design\U5303_FDK\U5303_Design_Template\ U5303A_FDK_Design_Template.wsv", as shown below:

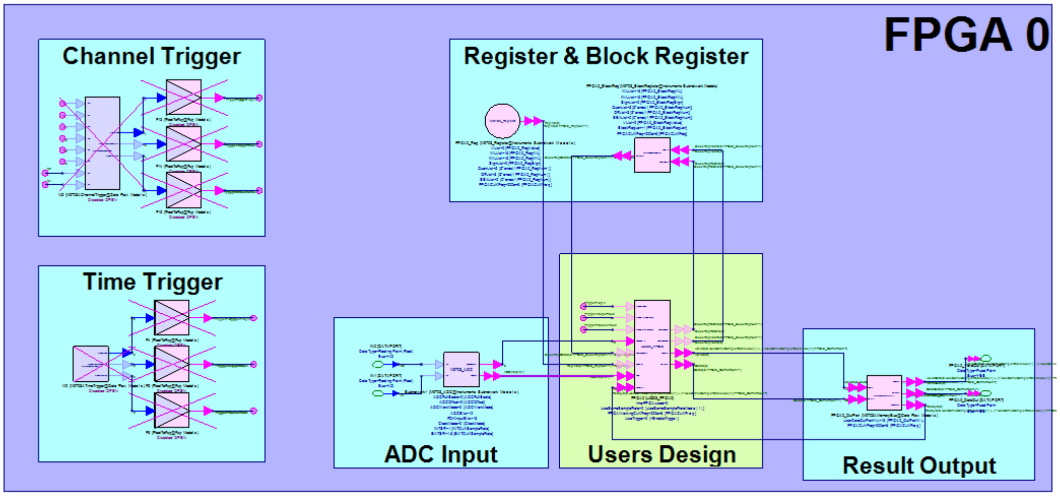


U5303ADesignTemplate
User_Design_Subnets
U5303_FPGA0 (Model)
U5303_TEMPLATE (Model)

### Top-level subnet: U5303_TEMPLATE

This subnet is the top level subnet of the U5303A Design Template. You can drag-and-drop it from the workspace tree to a schematic to instantiate a U5303A Design Template model in a schematic, and double click the model to open U5303A Design Template GUI to configure the U5303A DPU FPGA interfaces. All subnets and their parameters are all controlled by the GUI, so you can use the GUI to configure the U5303A DPU FPGA interface and working mode and all subnets under U5303A Design Template can be configured well automatically according to your input in the GUI.

If you double click the U5303_TEMPLATE subnet in workspace tree, you can open the subnet to view the implementation of the subnet. All things in this subnet implementation have be pre-configured well and controlled by U5303A Design Template GUI, so users need do nothing in this subnet.

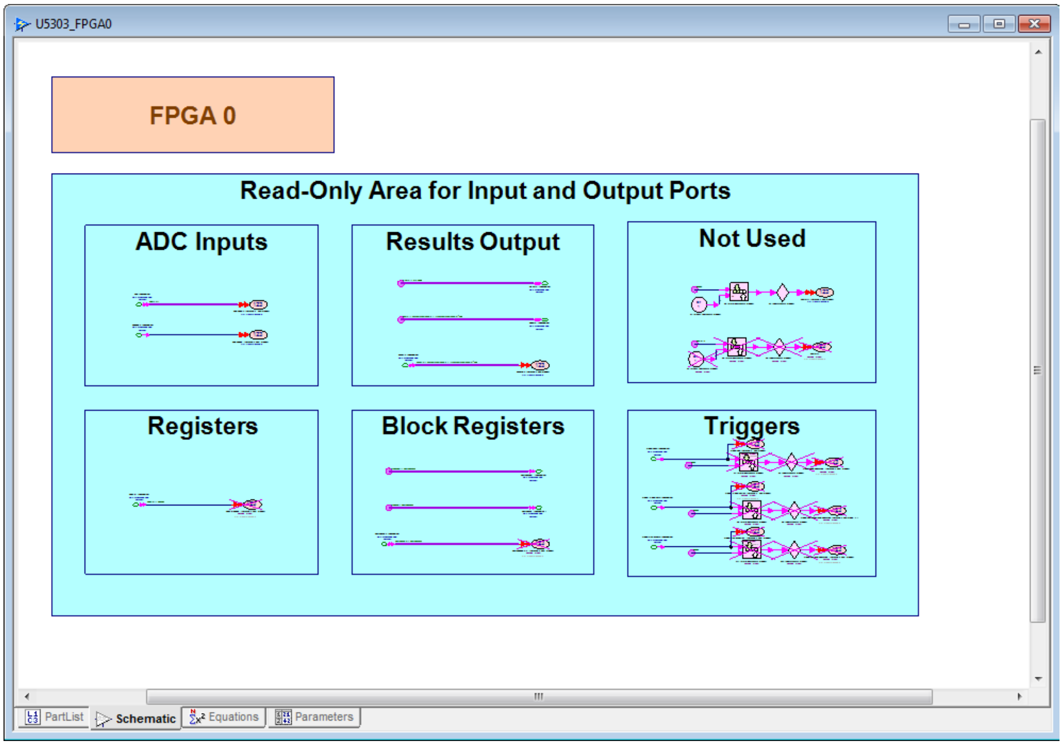


In U5303_TEMPLATE subnet, there is a subnet for custom real-time processing FPGA design entry in the green area Users Design of the DPU FPGA. The subnets and models in the other blue areas mimic U5303A DPU FPGA hardware architecture and provide the simplified interfaces for the custom real-time processing part.

In workspace tree of U5303A Design Template, the subnet in folder User_Design_Subnets – U5303_FPGA0, are in green Users Design area in top level U5303_TEMPLATE subnet. You can enter your model-based FPGA design in this subnet for the DPU FPGA.

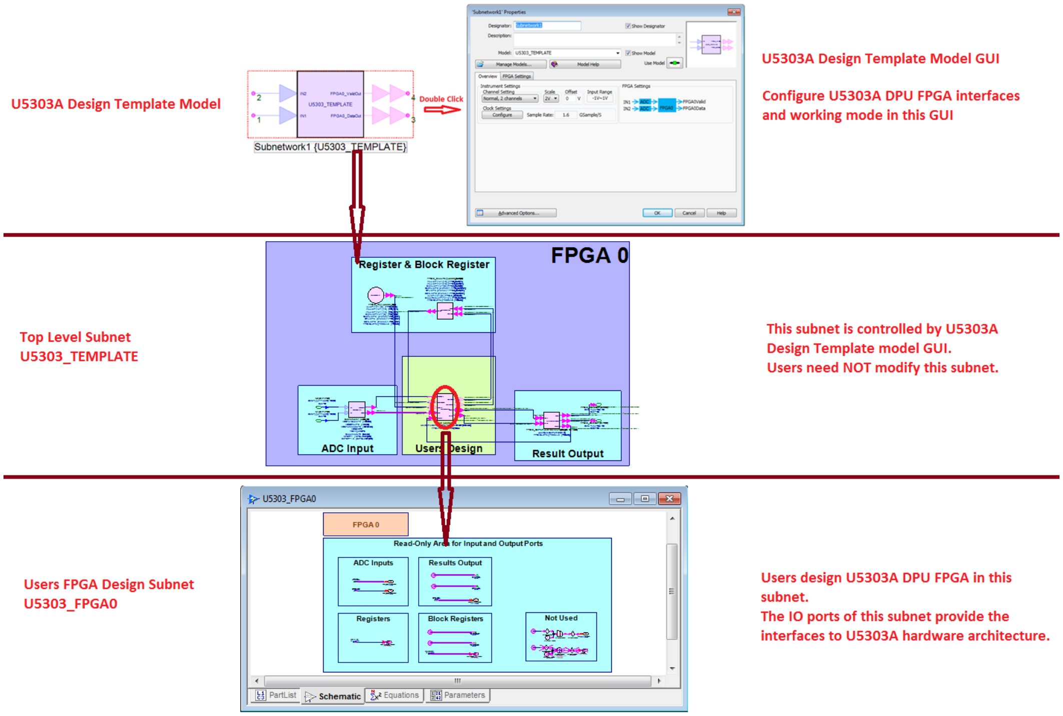
### Users FPGA Design subnets: U5303_FPGA0

You can double click subnet U5303_FPGA0 in workspace tree to look into the implementation of the subnet, as shown below.



Above figure shows a blank U5303_FPGA0 subnet and its interface models have been placed in this subnet in advance. These interface models are configured automatically according to your input in U5303A Design Template model GUI. Please don't modify anything in the blue "Read-Only" area in order to keep these pre-configured IO ports working correctly. Users can create their own model-based fixed-point design connecting with the defined interface. See Users FPGA Design Interfaces in U5303A Design Template to get details about how to connect model-based user application design with the defined FPGA interface.

In summary, the hierarchy of U5303A design template is shown in below figure.



U5303A Design Template Model

U5303A Design Template Model GUI

Configure U5303A DPU FPGA interfaces and working mode in this GUI

Top Level Subnet U5303_TEMPLATE

This subnet is controlled by U5303A Design Template model GUI. Users need NOT modify this subnet.

Users FPGA Design Subnet U5303_FPGA0

Users design U5303A DPU FPGA in this subnet. The IO ports of this subnet provide the interfaces to U5303A hardware architecture.

1. The top level is the U5303A Design Template model that you drag-and-drop from the workspace tree to schematic. You can double click this model in a schematic to open its GUI. You can configure U5303A interfaces and working mode in the GUI.
2. Look into U5303A Design Template model (subnet U5303_TEMPLATE), and you will go a lower level to view the implementation of the U5303A Design Template. You can find the peripheral subnets and user FPGA design subnet in this level. All things in this level are controlled by your input in U5303A Design Template model GUI automatically. Please don't modify anything in this level.
3. Look into the user FPGA design subnet, U5303_FPGA0, and you will go a lower level to create your own model-based design that you want to implement on the corresponding U5303A DPU FPGA in this level. The I/O ports have been placed in this subnet in advance and are also controlled by your input in U5303A Design Template model GUI automatically. So please don't modify anything in blue "Read-Only" area in this subnet.

Finally, you can follow the below steps to design U5303A DPU FPGA using U5303A Design Template:

1. Find U5303A Design Template from SystemVue example workspace: <*your SystemVue installation dir*>\Examples\Hardware Design\U5303_FDK\U5303_Design_Template\ U5303A_FDK_Design_Template.wsv. Save this example workspace to your working directory or copy folder U5303DesignTemplate from workspace tree of this example workspace to that of your own workspace.

2. Drag-and-drop subnet U5303_TEMPLATE from workspace to a schematic. Double click the U5303A Design Template model in your schematic to open its GUI. You can configure the interface (OutPort, Register and BlockRegister) of the DPU FPGA.
3. Find the U5303_FPGA0 subnet. Double click them in workspace tree to open the subnet. Then you can create your own model-based design and connect your design with your defined FPGA interfaces. Note that you can only use the models in SystemVue "Hardware Design Library", the models beyond this library can't generate HDL codes and can't be implemented on U5303A DPU FPGA finally.
4. Then after you finish your model-based design entry, you can go back to the top level schematic to create the input waveforms for the corresponding input ports of U5303A Design Template model and link the output ports of U5303A Design Template model to data sink models. Then you can run SystemVue simulation to verify your U5303A FPGA design via SystemVue software simulation.

### Users FPGA Design Interfaces in U5303A Design Template

In Overview of U5303A High-Speed Digitizer, U5303A DPU FPGAs' physical interfaces are introduced. The DPU FPGA has the following physical interface:

- ADC parallel data streams input
- Connectivity with two DDR3 memory and one QDRII memory
- PCIe connectivity with backplane via PCIe switch
- Control signals from CTRL FPGA

In order to make the custom real-time processing part implement under U5303A's hardware architecture easily, U5303A Design Template simplifies these physical interfaces. These simplified interfaces have less relationship with physical protocol, and they are all algorithm-level interfaces, so users can connect these simplified interfaces with their FPGA design easier. So users can save the time that is spent to study the details of U5303A DPU FPGAs' physical interfaces.

There are 5 types simplified interfaces in U5303A Design Template:

- ADC Input
- OutPort
- Register
- BlockRegister
- Trigger

You can configure these interfaces in U5303A Design Template model GUI for your selected U5303A DPU FPGAs and connect your configured interfaces with your model-based fixed-point design in the users FPGA design subnet (U5303_FPGA0 ).

Register, BlockRegister and Trigger are optional interfaces. You can configure whether to use them in U5303A Design Template model GUI.

These simplified interfaces have been placed in users FPGA design subnet U5303_FPGA0 in advance, and the I/O ports of U5303_FPGA0 subnet are controlled by your configuration in U5303A Design Template model GUI automatically. Net labels of lines are used to connect these ports. Users can use net labels to connect their own model-based design with these interfaces. For details of how to use net labels of connection line, please refer to Connection Line Net Labels.

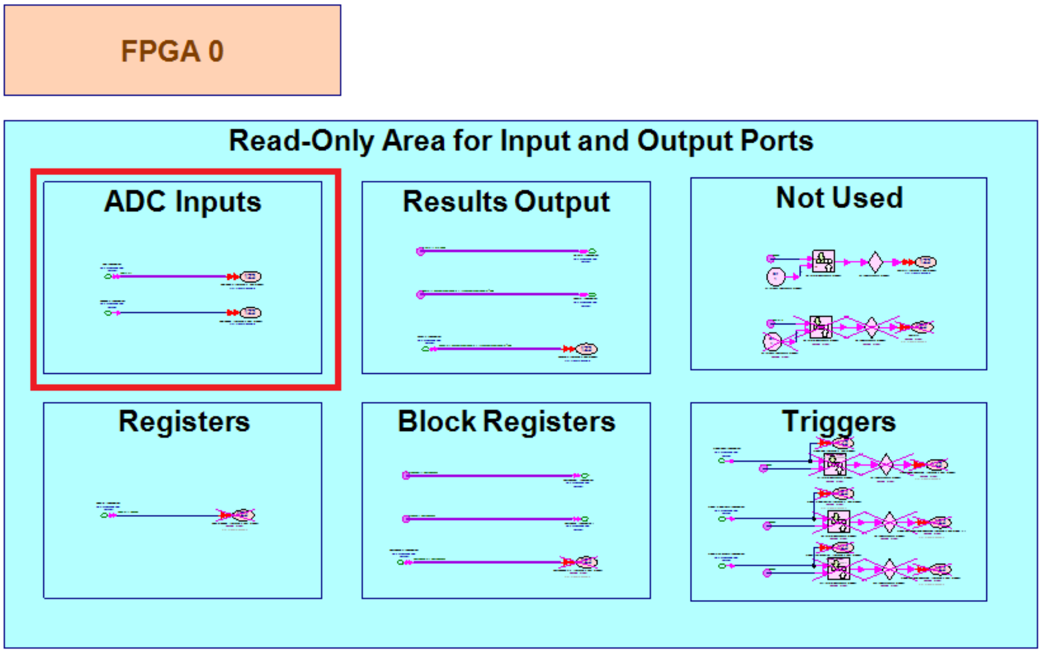
### ADC Input

ADC Input is corresponding to U5303A DPU FPGAs' physical interface ADC Input (see Overview of U5303A High-Speed Digitizer). Users can use this interface to connect ADC parallel input streams with their own FPGA design.

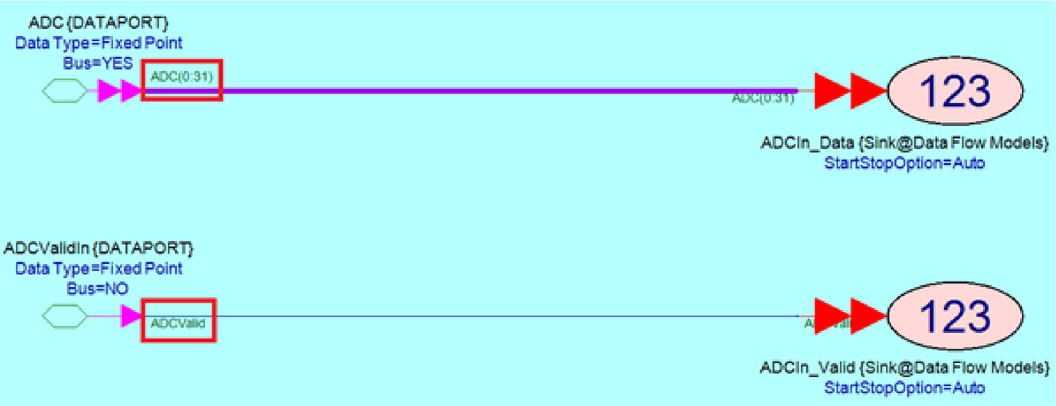
ADC Input in U5303A Design Template has the same behavior as it physical ADC Input parallel streams. It provides parallel ADC input streams and a valid signal to indicate whether the corresponding ADC parallel streams are valid. As described in Overview of U5303A High-Speed Digitizer, there are valid parallel ADC samples every two DPU FPGA system clock cycles.

Connect ADC input with your FPGA design in subnet U5303_FPGA0:

In user FPGA design subnet U5303_FPGA0, ADC parallel input streams are a input bus with width 32 and a valid input port. They are configured by U5303A Design Template model GUI automatically.



The input ports are connected to lines with Net Label. The Net Label of parallel ADC input data is ADC(0:31). Net Label of ADC input valid signal is ADCValid. So user can use Net Label name to connect the parallel ADC input with their own model-based FPGA design.



As described in Overview of U5303A High-Speed Digitizer, ADC input data ADC(0: 31) is related U5303A ADC sample mode:

For Normal, 2 channels mode:

- IN1 and IN2 connect with FPGA0;

ADC(0:15) are 16 parallel ADC samples of the first channel for current FPGA. ADC(0) is the oldest sample and ADC(15) is the newest sample.
ADC (16:31) are 16 parallel ADC samples of the second channel for current FPGA. ADC(16) is the oldest sample and ADC(31) is the newest sample.

For Interleaved, channel 1 mode:

- IN1 connects with FPGA0;

ADC(0:31) are 32 parallel samples of the connected channel of the current FPGA. ADC(0) is the oldest sample and ADC(31) is the newest sample.

For Interleaved, channel 2 mode:

- IN2 connects with FPGA0;

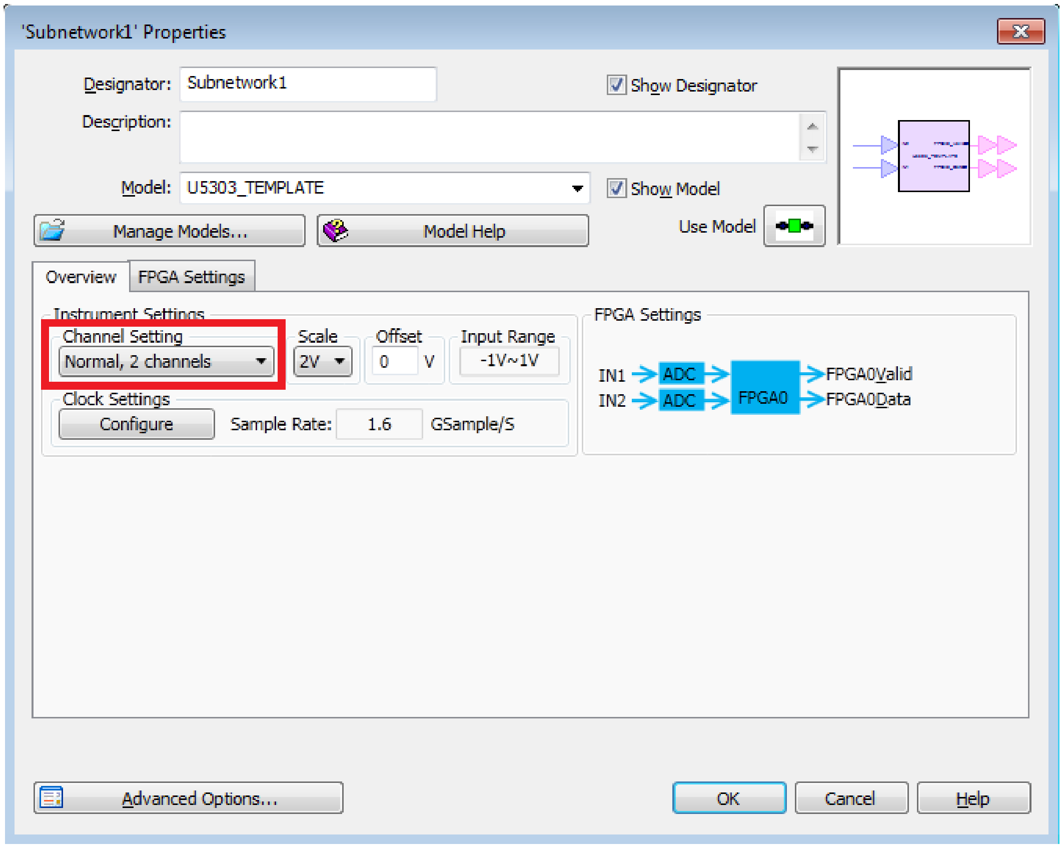
ADC(0:31) are 32 parallel samples of the connected channel of the current FPGA. ADC(0) is the oldest sample and ADC(31) is the newest sample.

For example, if IN2 connects with FPGA0 for interleaved mode, in subnet U5303_FPGA0, ADC(0:31) are 32 parallel samples of IN2.

Configure ADC input mode in GUI of U5303A Design Template:

ADC sample mode can be configured in U5303A Design Template model GUI. Then in FPGA Settings tab of the GUI, the connection relationship between IN1, IN2 and ADC(0:31) is also shown.

For example, Normal, 2 channels mode and FPGA Settings tab:



'Subnetwork1' Properties

Designator: Subnetwork1 ☑ Show Designator

Description:

Model: U5303_TEMPLATE ☑ Show Model

Instrument Settings

Channel Setting: Normal, 2 channels

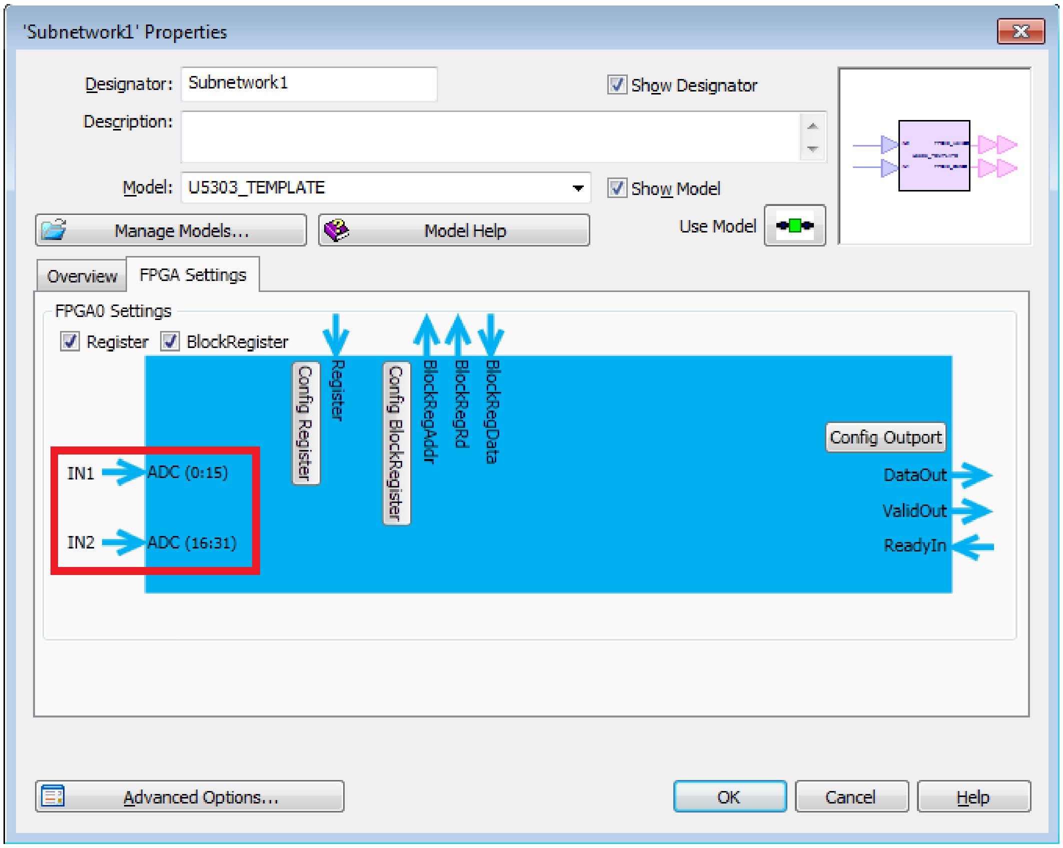
Scale: 2V

Offset: 0 V

Input Range: -1V~1V

Clock Settings

Sample Rate: 1.6 GSample/S



'Subnetwork1' Properties

Designator: Subnetwork1 ☑ Show Designator

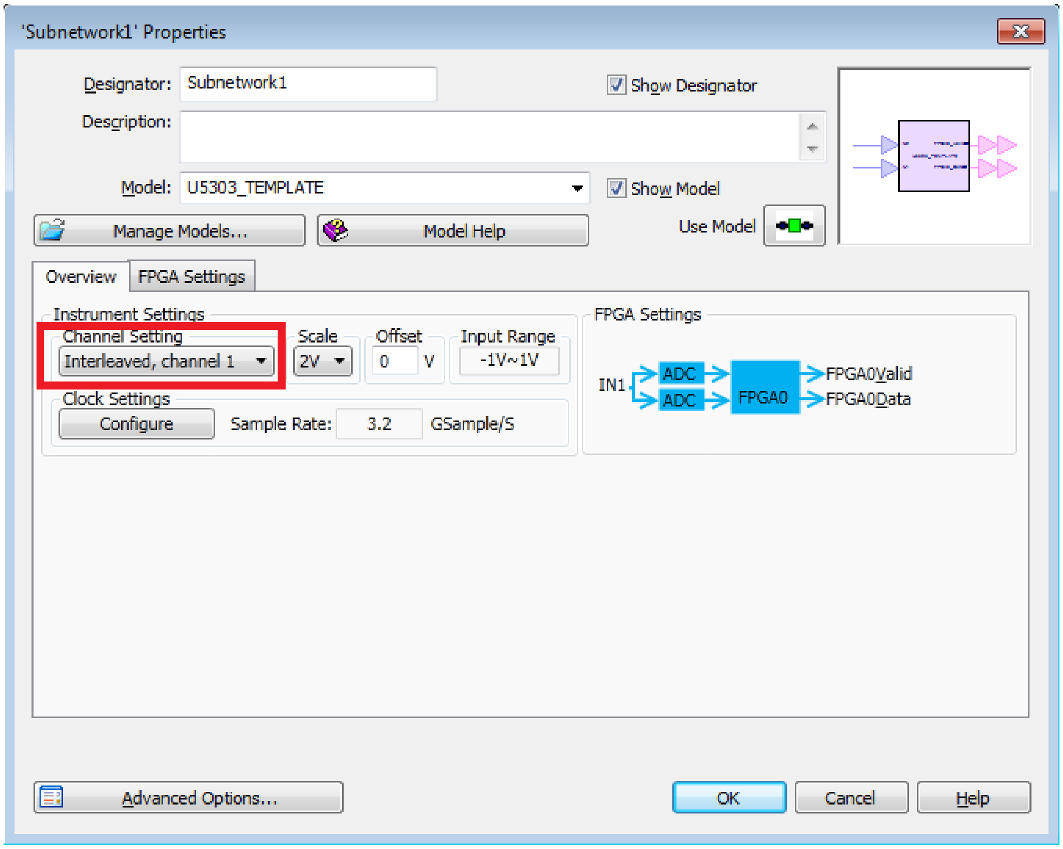
Description:

Model: U5303_TEMPLATE ☑ Show Model

FPGA0 Settings

☑ Register ☑ BlockRegister

For example, Interleaved, channel 1 mode and FPGA Settings tab:



'Subnetwork1' Properties

Designator: Subnetwork1

☑ Show Designator

Description:

Model: U5303_TEMPLATE

☑ Show Model

Use Model

Overview | FPGA Settings

Instrument Settings

Channel Setting: Interleaved, channel 1

Scale: 2V

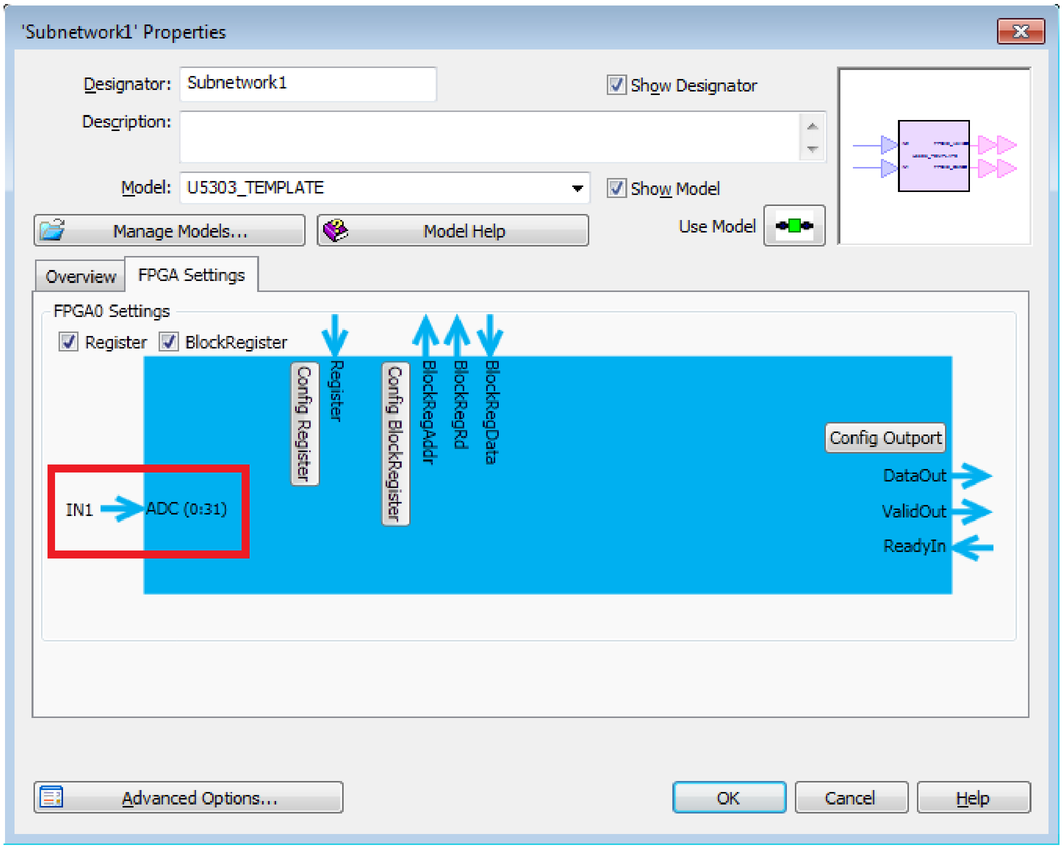
Offset: 0 V

Input Range: -1V~1V

Clock Settings

Sample Rate: 3.2 GSample/S

FPGA Settings



'Subnetwork1' Properties

Designator: Subnetwork1

☑ Show Designator

Description:

Model: U5303_TEMPLATE

☑ Show Model

Use Model

Overview | FPGA Settings

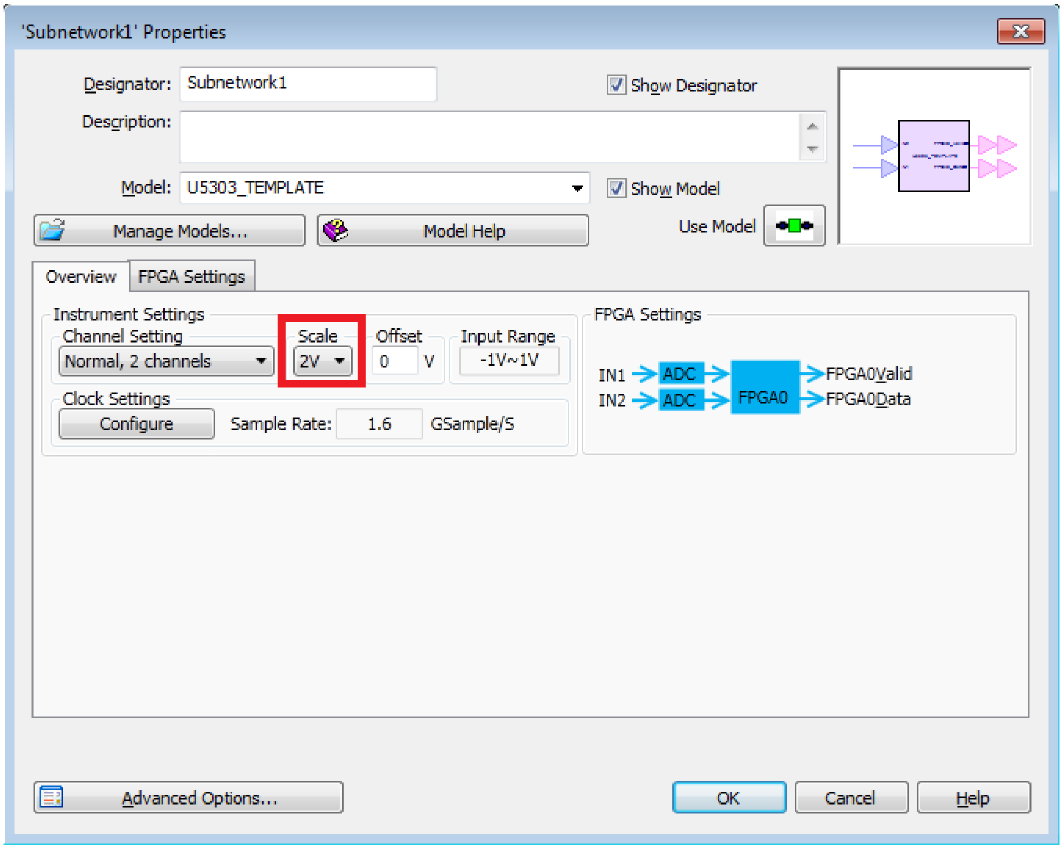
FPGA0 Settings

☑ Register ☑ BlockRegister

Each line of bus ADC(0:31) is an input port of ADC sample and its data type is Fixed-point.

- Its WordLength is 16 bit.
- Its Integer WordLength is 1 bit when Full Scale is 2V (Input Range: -1V~1V); Its Integer WordLength is 0 bit when Full Scale is 1V (Input Range: -0.5V~0.5V)
- It's Signed.

Full Scale can be set in U5303A Design Template model GUI:



'Subnetwork1' Properties

Designator: Subnetwork1 ☑ Show Designator

Description:

Model: U5303_TEMPLATE ☑ Show Model

Overview | FPGA Settings

Instrument Settings

Channel Setting: Normal, 2 channels

Scale: 2V

Offset: 0 V

Input Range: -1V~1V

Clock Settings

Sample Rate: 1.6 GSample/S

FPGA Settings

The data type of U5303A Design Template model's input ports IN1~IN2 is Real. U5303A Design Template will quantize the Real input waveforms to get fixed-point ADC sample value for ADC(0:31) according to U5303A's ADC behavior. It will quantize the Real waveform to a fixed-point number with WordLength 12 and then extend the 12bit fixed-point number to 16bit. Integer WordLength and Signed are always as described above. Please refer to Overview of U5303A High-Speed Digitizer for U5303A's ADC behavior.

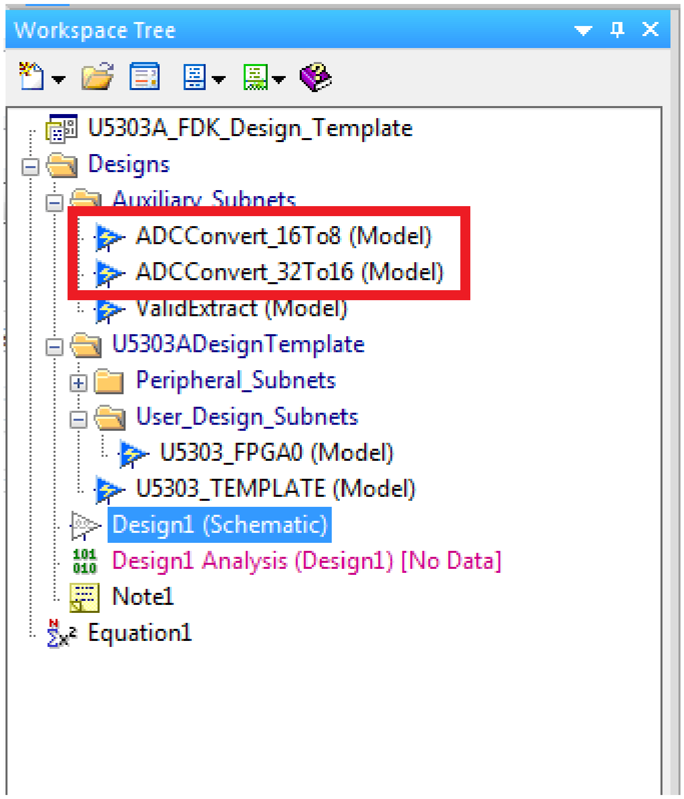
Subnets that can help you remove ADCValid:

ADCValid is Net Label of the non-bus ADC valid input port. Its data type is also fixed-point.

- Its WordLength is 1 bit.
- Its Integer WordLength is 1 bit
- It's UnSigned.

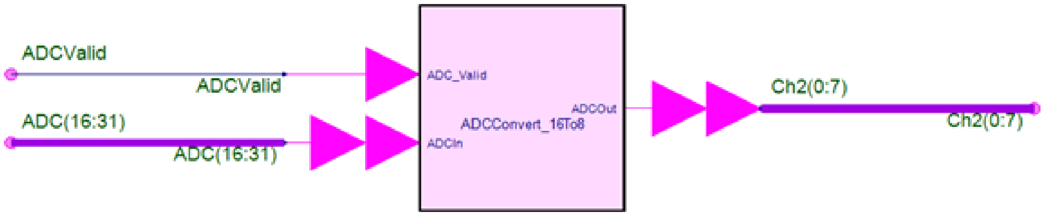
ADCValid alternates between 0 and 1 always. When ADCValid is 1, the corresponding ADC(0:31) are valid ADC samples. When ADCValid is 0, ADC(0:31) will hold their current values. So every two samples, there is a valid ADC(0:31).

If you want to get all valid ADC parallel samples (valid signal is always 1, so all ADC samples are valid. Then you need not ADC valid signal), two subnets ADCConvert_16To8 and ADCConvert_32To16 are provided in example workspace: *<your SystemVue installation dir>*\Examples\Hardware Design\U5303_FDK\U5303_Design_Template\ U5303A_FDK_Design_Template.wsv

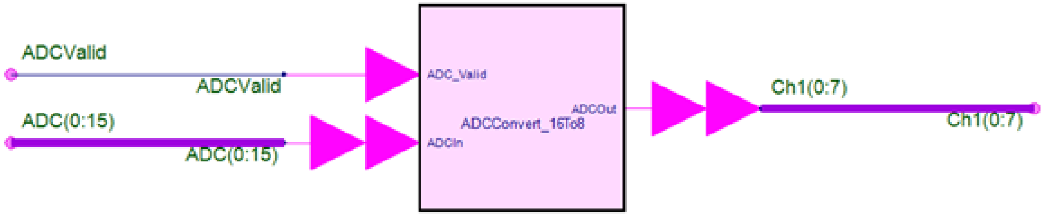


Because ADCValid is 1 every two samples, the width of bus ADC needs go down 2 times when you get all valid parallel streams.

For Normal, 2 channels mode, the width of bus ADC become from 16 to 8, you can use subnet ADCConvert_16To8 as below to get all valid parallel ADC samples for two channels.

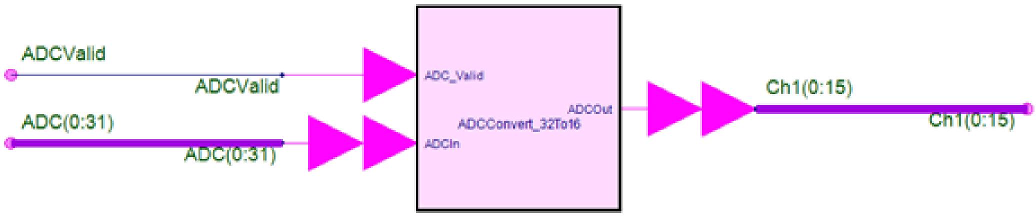


Subnetwork1 {ADCConvert_16To8}



Subnetwork2 {ADCConvert_16To8}

For Interleaved, channel 1 or Interleaved, channel 2 mode, the width of bus ADC become from 32 to 16, you can use subnet ADCConvert_32To16 as below to get all valid parallel ADC samples for one channel.



Subnetwork3 {ADCConvert_32To16}

It is not mandatory to connect all ADC(0:31) lines with user's FPGA design in subnet U5303_FPGA0. For example, if you just want to use IN1 and don't use IN2 for DPU FPGA, then you can only connect ADC(0:15) to your design and don't use ADC(16:31) in subnet U5303_FPGA0.

NOTE

ADC parallel input data: **ADC(0:31)**

ADC Valid: **ADCValid**

Use the net labels to connect your design with ADC input interface in U5303A user design subnet **U5303_FPGA0.**

## OutPort

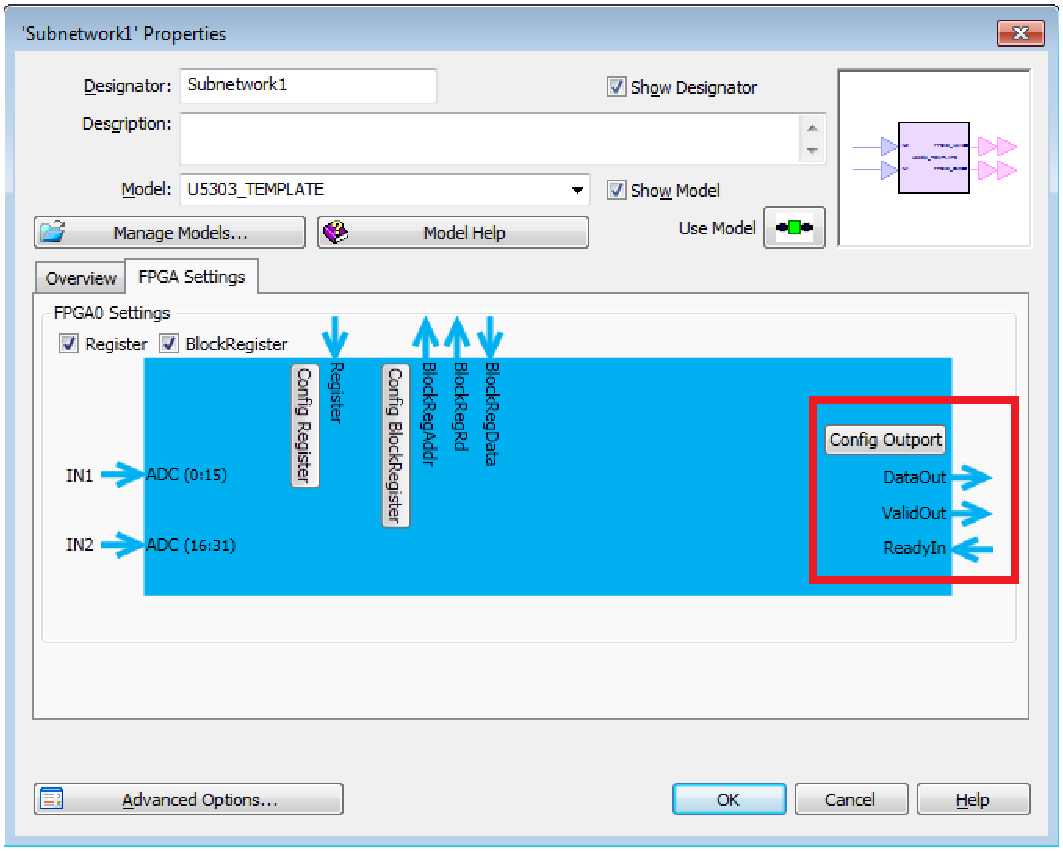
Users can process ADC input samples with their own FPGA design in subnet U5303_FPGA0. Then the processed results can be outputted via OutPort. Users can define multiple OutPort and each their desired output result can be outputted via the corresponding OutPort. Because users' output results from U5303A Design Template user FPGA design subnet (U5303_FPGA0) are Fixed-point data type, the corresponding OutPort can be defined as users desired Fixed-point data format ( WordLength, Integer WordLength and IsSigned).

Because the real-time processed results from U5303A DPU FPGA may be high-speed data stream, PCIe bandwidth may not be enough to fetch all results to PC without data loss. And also, software running on PC is usually too slow to process the real-time processed results from U5303A hardware. So in U5303A FPGA design flow, the real-time processed results are buffered to DDR3 SDRAM on U5303A at first, because the data bandwidth between DPU FPGA and DDR3 SDRAM is much larger than PCIe data bandwidth. User can specify a block of SDRAM for the data buffering. When the block of SDRAM is filled in, SystemVue that runs a U5303A instrument co-simulation on a computer can read back the buffered results via PCIe. Then SystemVue can parse PCIe data format to user defined OutPort Fixed-point data type automatically.

So user just need define their own OutPort Fixed-point data type for software simulation and U5303A instrument co-simulation and don't need care the physical DDR3 and PCIe protocol and data format. SystemVue will automatically generate HDL wrapper to convert between user defined OutPort data format and DDR3 and PCIe data format during generating U5303A DPU FPGA programming file.

Define your own OutPort in GUI of U5303A Design Template:

For each U5303A DPU FPGA, OutPort can be defined in the FPGA Settings tab of U5303A Design Template model GUI:



'Subnetwork1' Properties

Designator: Subnetwork1

☑ Show Designator

Description:

Model: U5303_TEMPLATE

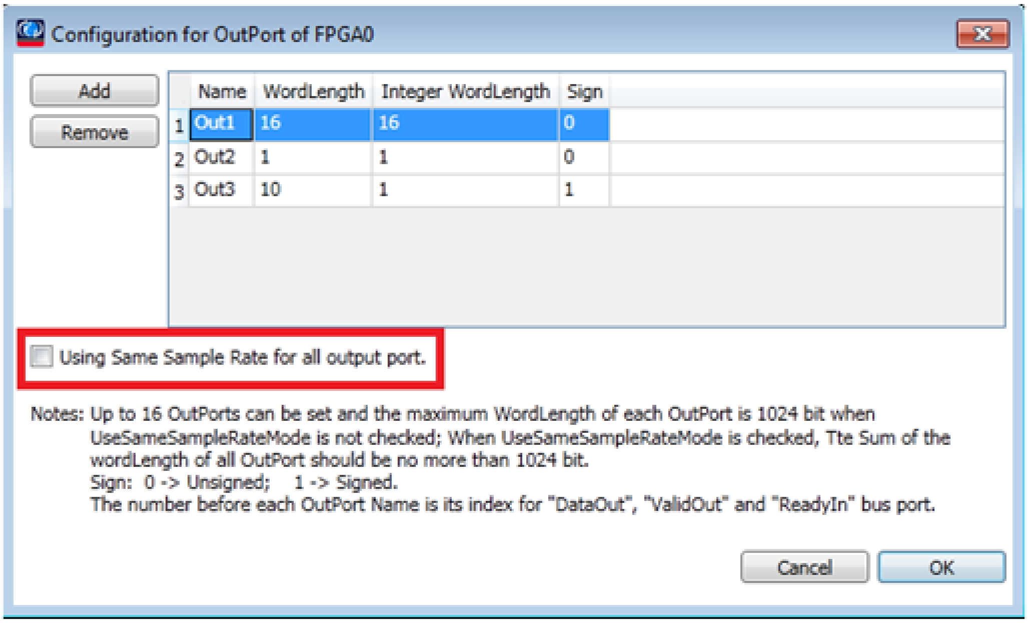
☑ Show Model

FPGA0 Settings

☑ Register ☑ BlockRegister

As shown in the interface diagram, each OutPort is represented as a set of ports: DataOut output port, ValidOut output port and ReadyIn input port. It's a typical data stream style interface. DataOut is the data values output. ValidOut shows the corresponding DataOut is valid when it's 1, otherwise when it's 0, the corresponding DataOut is not valid. ReadyIn shows the following hardware is ready to receive the current output. When both ValidOut and ReadyIn are 1, a data transfer occurs.

Click button Config Outport to open a new GUI to define the OutPort for the DPU FPGA.



|   | Name | WordLength | Integer WordLength | Sign |
|---|---|---|---|---|
| 1 | Out1 | 16 | 16 | 0 |
| 2 | Out2 | 1 | 1 | 0 |
| 3 | Out3 | 10 | 1 | 1 |

□ Using Same Sample Rate for all output port.

Notes: Up to 16 OutPorts can be set and the maximum WordLength of each OutPort is 1024 bit when UseSameSampleRateMode is not checked; When UseSameSampleRateMode is checked, Tte Sum of the wordLength of all OutPort should be no more than 1024 bit.
Sign: 0 -> Unsigned; 1 -> Signed.
The number before each OutPort Name is its index for "DataOut", "ValidOut" and "ReadyIn" bus port.

In the OutPort configuration GUI, user can define multiple OutPort. User needs define the following items for each OutPort:

- Name
- WordLength
- Integer WordLength
- Sign

Name is a meaningful string to help you to identify each OutPort. WordLength, Integer WordLength and Sign specify the fixed-point format for data of the corresponding OutPort. The fixed point format is just used to specify DataOut. Both ValidOut and ReadyIn are 1 bit unsigned integer (its value is 0 or 1).

In each row, a OutPort is defined. There is a number on the left of OutPort's Name. The number is the index of each OutPort. The index is a very useful number for each defined OutPort. In user FPGA design subnet U5303_FPGA0, OutPort are bus IO ports. This index is the bus index for each defined OutPort. The index is one-based.

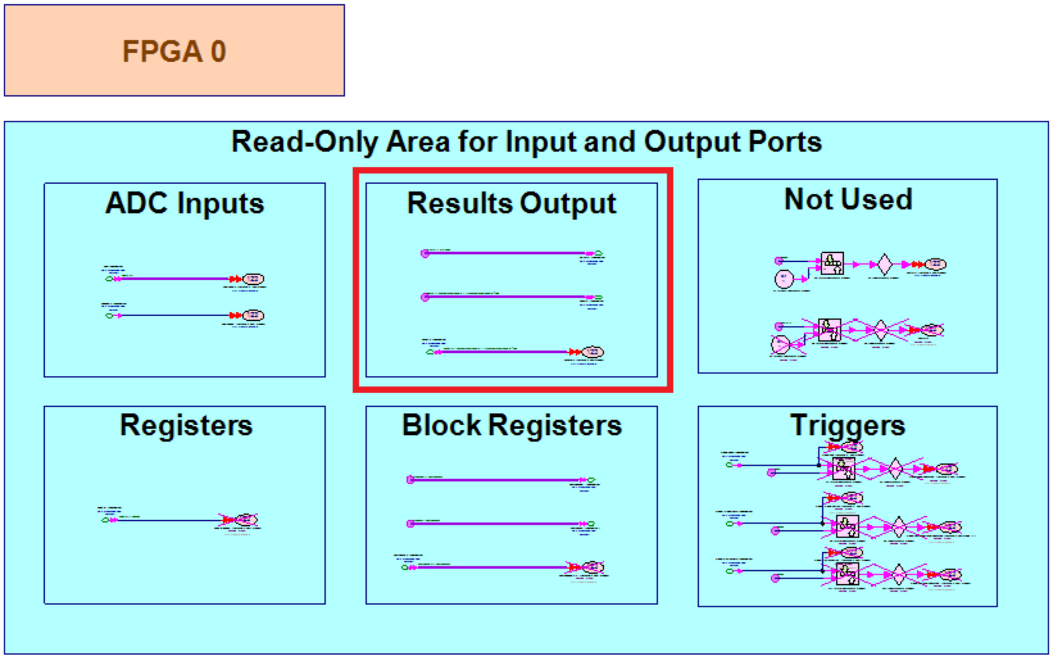
OutPort has two working modes:

- Same Sample Rate Mode: checkbox Using Same Sample Rate for all output port is checked. All defined OutPorts share the same ValidOut signal. So they will generate valid output in the same rate. In this mode, you can define up to 16 OutPorts for each U5303 FPGA, and the maximum bit width of each OutPort is 1024.

- Non-Same Sample Rate Mode: checkbox Using Same Sample Rate for all output port is unchecked. Each defined OutPort has its own ValidOut signal. So they can generate valid output in different rates. In this mode, the sum of all OutPorts' bit width must be less or equal to 1024.

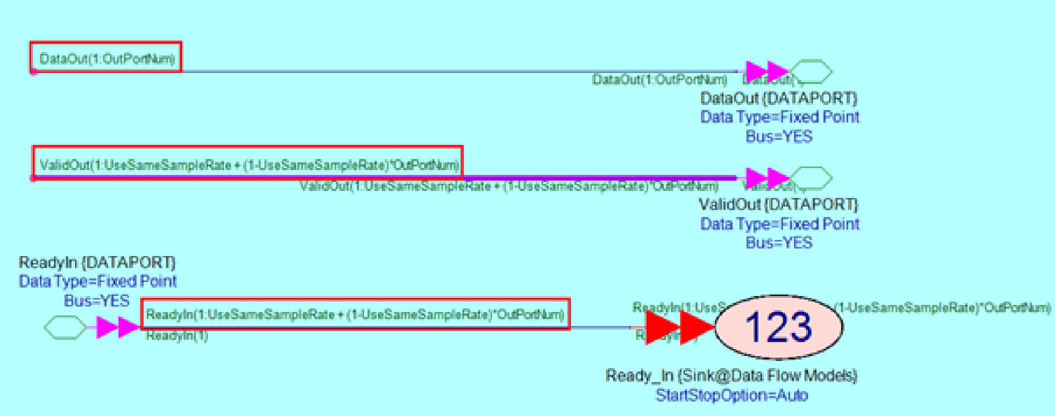
<ins>Connect OutPort with your FPGA design in subnet U5303_FPGA0:</ins>

In user FPGA design subnet U5303_FPGA0, the IO ports of your defined OutPort in GUI are configured by U5303A Design Template model GUI automatically. You don't need edit them and you just need use net labels to connect your model-based design with the pre-configured IO ports.



The OutPort interface is represented by a set of fixed-point bus ports. If you defined N OutPort for DPU FPGA, the OutPort interface will be:

- Data output bus: DataOut(1:N)
- Valid output bus: For Non-Same Sample Rate Mode, ValidOut(1:N); For Same Sample Rate Mode, ValidOut(1). All defined OutPort will share the same ValidOut(1).
- Ready input bus: For Non-Same Sample Rate Mode, ReadyIn(1:N); For Same Sample Rate Mode, ReadyIn(1). All defined OutPort will share the same ReadyIn(1).


DataOut(1:OutPortNum)
DataOut(1:OutPortNum)
DataOut {DATAPORT}
Data Type=Fixed Point
Bus=YES
ValidOut(1:UseSameSampleRate + (1-UseSameSampleRate)*OutPortNum)
ValidOut(1:UseSameSampleRate + (1-UseSameSampleRate)*OutPortNum)
ValidOut {DATAPORT}
Data Type=Fixed Point
Bus=YES
ReadyIn {DATAPORT}
Data Type=Fixed Point
Bus=YES
ReadyIn(1:UseSameSampleRate + (1-UseSameSampleRate)*OutPortNum)
ReadyIn(1)
(1-UseSameSampleRate)*OutPortNum)
123
Ready_In {Sink@Data Flow Models}
StartStopOption=Auto

The number of OutPort and fixed-point data type of each DataOut are defined in GUI of U5303A Design Template. The fixed-point data type of all ValidOut and ReadyIn is one bit logic (unsigned, word length = integer word length = 1).

For each OutPort, it has a set of ports DataOut(*x*) output, ValidOut(*x*) output and ReadyIn(*x*) input. The set of ports follows the handshaking rule of AXI4-stream. So you also need follow the AXI4-stream handshaking rule to deal with the timing of DataOut, ValidOut and ReadyIn ports in your application design logic. You can use net labels DataOut(*x*), ValidOut(*x*) and ReadyIn(*x*) to connect the set of ports with your application design ports.

- In Non-Same Sample Rate Mode, *x* is the bus index between 1:N (N is the number of your defined OutPort).
- In Same Sample Rate Mode, for DataOut(*x*), *x* is the bus index between 1:N ( N is the number of your defined OutPort). For ValidOut(*x*) and ReadyIn(*x*), *x* can only be 1.

For example, if you define 2 OutPorts in GUI of U5303A Design Template, you can:

In Non-Same Sample Rate Mode:

- Connect the data output of the first OutPort to the line with net label DataOut(1);
- Connect the valid output of the first OutPort to the line with net label ValidOut(1);
- Connect the ready input of the first OutPort to the line with net label ReadyIn (1);
- Connect the data output of the second OutPort to the line with net label DataOut(2);
- Connect the valid output of the second OutPort to the line with net label ValidOut(2);
- Connect the ready input of the second OutPort to the line with net label ReadyIn(2);

In Same Sample Rate Mode,

- Connect the data output of the first OutPort to the line with net label DataOut(1);

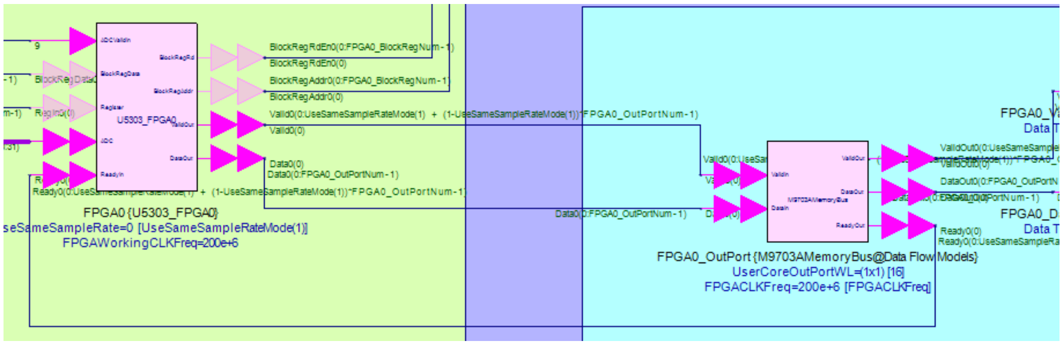
- Connect the data output of the second OutPort to the line with net label DataOut(2);
- Just generate one valid out for all output data in your FPGA design and connect it to the line with net label ValidOut(1);
- Connect the line with net label ReadyIn(1) as input to your FPGA design to control your output

For each OutPort, your fixed-point design needs follow AXI4-stream handshaking rule for the timing of data, valid and ready signals. When both ValidOut and ReadyIn are 1, a data transfer occurs.

NOTE

- If you define **N OutPort** in GUI, don't use net labels **DataOut(*x*)**, **ValidOut(*x*)** and **ReadyIn(*x*)** that ***x*** is beyond the range of **1 to N**. Otherwise, when you run the simulation, SystemVue will post you an error.
- Especially, in **Same Sample Rate** Mode**,** if ***x*** for **ValidOut(*x*)** and **ReadyIn(*x*)** is not **1,** when you run the simulation, SystemVue will post you an error.
- The fixed-point data format of your data output must match with the corresponding **OutPort** definition in GUI of U5303A Design Template. And the fixed-point data format of your **ValidOut(*x*)** must be one-bit logic (word length = integer word length = 1, unsigned). Otherwise, SystemVue will post an error when running the simulation.

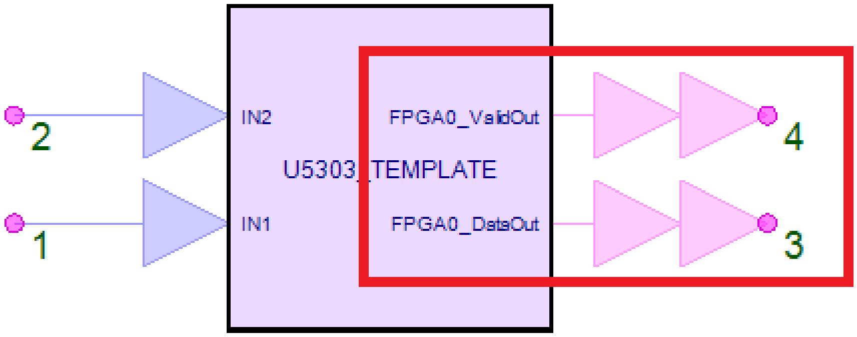
In U5303A Design Template, each selected FPGA has an M9703MemoryBus model (because OutPort of M9703A and U5303A have the same behavior, M9703A and U5303A share the same MemoryBus simulation model) to link its input with DataOut and ValidOut output buses of U5303_FPGA0 subnet and generate output to ReadyIn input bus of U5303_FPGA0 subnet. You can observe it in U5303_TEMPLATE subnet:


ADCValidIn
BlockRegRd
BlockRegData
BlockRegAddr
Register
U5303_FPGA0
ValidOut
ADC
DataOut
ReadyIn
9
BlockRegData0
RegIn0(0)
Ready0(0)
BlockRegRdEn0(0:FPGA0_BlockRegNum-1)
BlockRegRdEn0(0)
BlockRegAddr0(0:FPGA0_BlockRegNum-1)
BlockRegAddr0(0)
Valid0(0:UseSameSampleRateMode(1) + (1-UseSameSampleRateMode(1))*FPGA0_OutPortNum-1)
Valid0(0)
Data0(0)
Data0(0:FPGA0_OutPortNum-1)
Ready0(0:UseSameSampleRateMode(1) + (1-UseSameSampleRateMode(1))*FPGA0_OutPortNum-1)
FPGA0 {U5303_FPGA0}
seSameSampleRate=0 [UseSameSampleRateMode(1)]
FPGAWorkingCLKFreq=200e+6
Data0(0:FPGA0_OutPortNum-1)
Valid0(0:UseSa
ValidIn
DataIn
M9703AMemoryBus
ValidOut
DataOut
ReadyOut
ValidOut0(0:UseSameSample
ValidOut0(0)
DataOut0(0:FPGA0_OutPortN
DataOut0(0)
Ready0(0)
Ready0(0:UseSameSampleRa
FPGA0_V
Data T
FPGA0_D
Data T
FPGA0_OutPort {M9703AMemoryBus@Data Flow Models}
UserCoreOutPortWL=(1x1) [16]
FPGACLKFreq=200e+6 [FPGACLKFreq]

When SystemVue generates FPGA programming file automatically, it inserts an N-to-1 AXI4-stream switcher to connect N defined **OutPorts** to FPGA DDR3 RAM AXI4 interface. It also packages different OutPort data with header indicators. The M9703MemoryBus model will mimic the hardware behavior to generate ReadyIn feedback for user's application design when you do the software simulation.

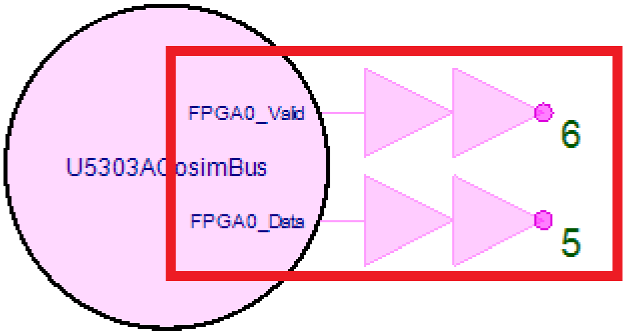
OutPort on top level U5303A Design Template model and U5303ACosimBus model:

As your desired results are outputted via OutPort, the results will be outputted from top level U5303A Design Template model in software simulation, and will be outputted from U5303ACosimBus model in U5303A instrument co-simulation.



Subnetwork1 {U5303_TEMPLATE}

For top level U5303A Design Template model, it has a set of output bus ports for DPU FPGA, FPGA0_DataOut and FPGA0_ValidOut. They have the same function as the output from U5303A user FPGA design subnet U5303_FPGA0. But as it mimic the hardware behavior and when SystemVue reads data from U5303A, actually the data have been buffered in on-board DDR3 SDRAM. So it doesn't need a ReadyIn port anymore.



U1 {U5303ACosimBus@Data Flow Models}
WorkDirectory=
InstrumentAddress=0

For U5303ACosimBus model, its output ports are similar as top level U5303A Design Template model. It has also a set of output bus ports for DPU FPGA, FPGA0_Data and FPGA0_Valid and doesn't need ReadyIn port.

For the top level U5303A Design Template model and U5303ACosimBus model, the DataOut values of OutPort in U5303_FPGA0 subnet can be outputted through FPGA0_DataOut, but the behavior of the FPGA0_ValidOut will be different . Please refer to Software Simulation Behavior Description and U5303 Co-Simulation Model Simulation Behavior Description.

NOTE

If you define **N OutPort**, the net label you can use to connect your FPGA design in **U5303_FPGA0** subnet:

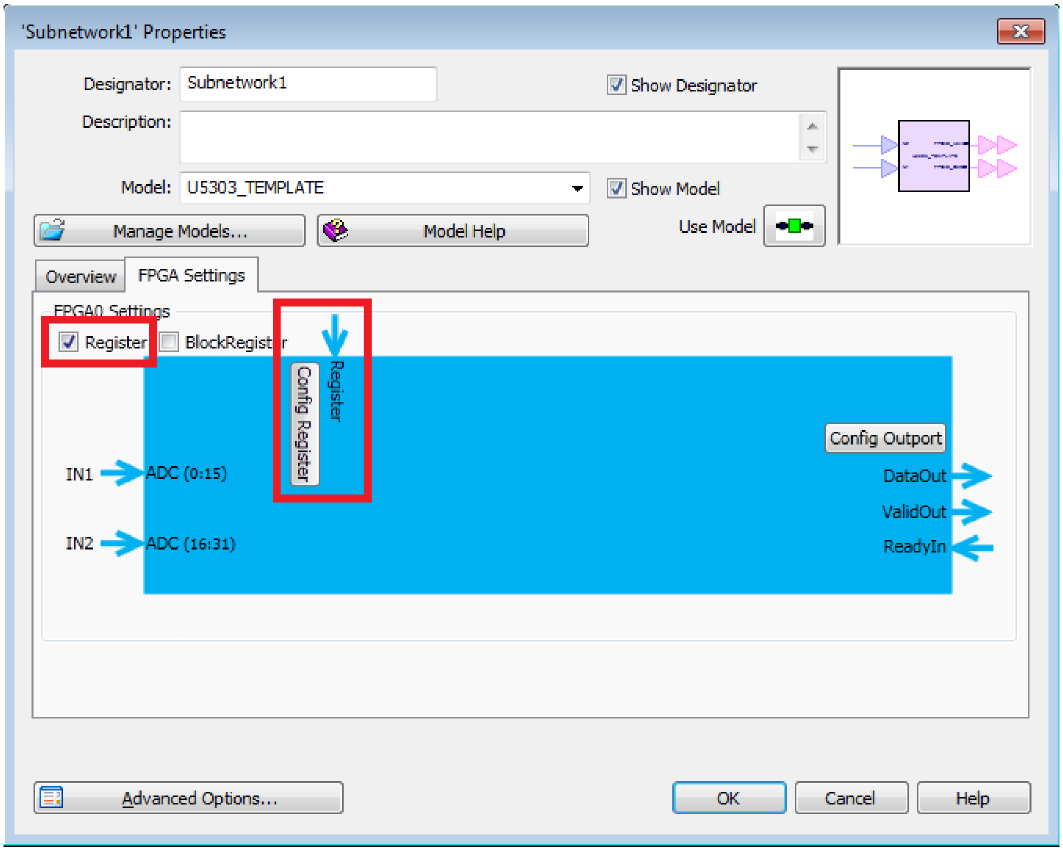
- For **Non-Same Sample Rate** mode: **DataOut(1:N)**, **ValidOut(1:N)** and **ReadyIn(1:N). DataOut(1:N)** and **ValidOut(1:N)** are output ports and **ReadyIn(1:N)** is input port.
- For **Same Sample Rate** mode: **DataOut(1:N)**, **ValidOut(1)** and **ReadyIn(1)**. **DataOut(1:N)** and **ValidOut(1)** are output ports and **ReadyIn(1)** is input port.

## Register

Register is an optional FPGA interface. You can define arbitrary number of Registers as long as it can be implemented on DPU FPGA. Each Register reserves a value that can be configured dynamically after FPGA programming file is generated. It can make your generated FPGA programming file more flexible.

Configure Register in U5303A Design Template GUI:

You can enable or disable Register in U5303A Design Template model GUI by checking or unchecking Register checkbox as shown below.



'Subnetwork1' Properties

Designator: Subnetwork1

☑ Show Designator

Description:

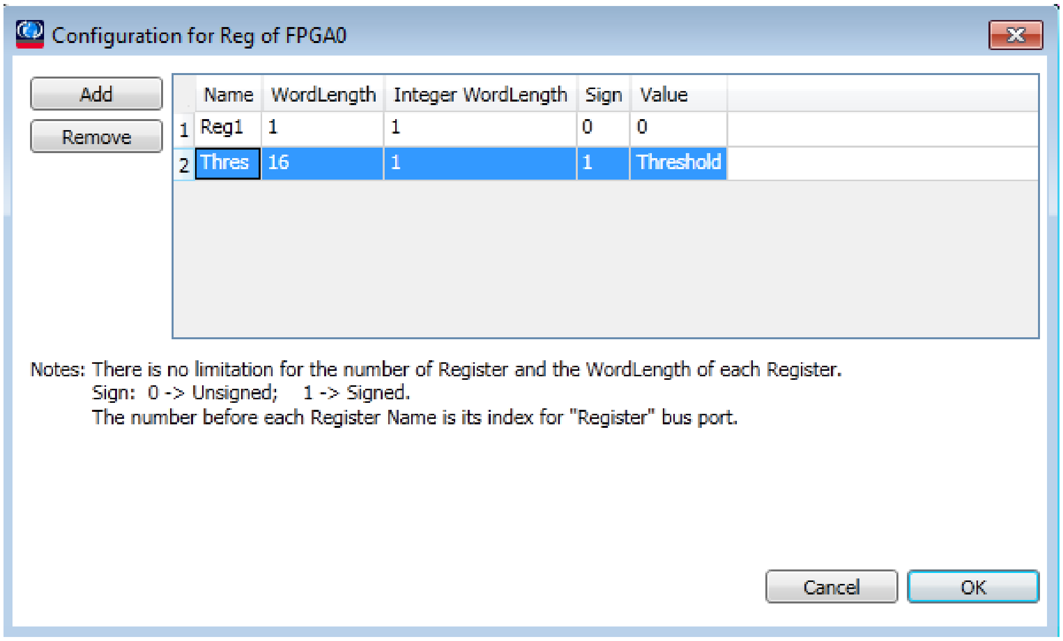
Model: U5303_TEMPLATE

☑ Show Model

FPGA0 Settings

☑ Register ☐ BlockRegist r

Register interface is a set of reserved fixed-point registers whose values can be re-configured at the beginning of a simulation, and then the Register values are kept until the end of simulation. After you enable Register, you can click Config Register button to open a new GUI:



Configuration for Reg of FPGA0

| | Name | WordLength | Integer WordLength | Sign | Value |
|---|---|---|---|---|---|
| 1 | Reg1 | 1 | 1 | 0 | 0 |
| 2 | Thres | 16 | 1 | 1 | Threshold |

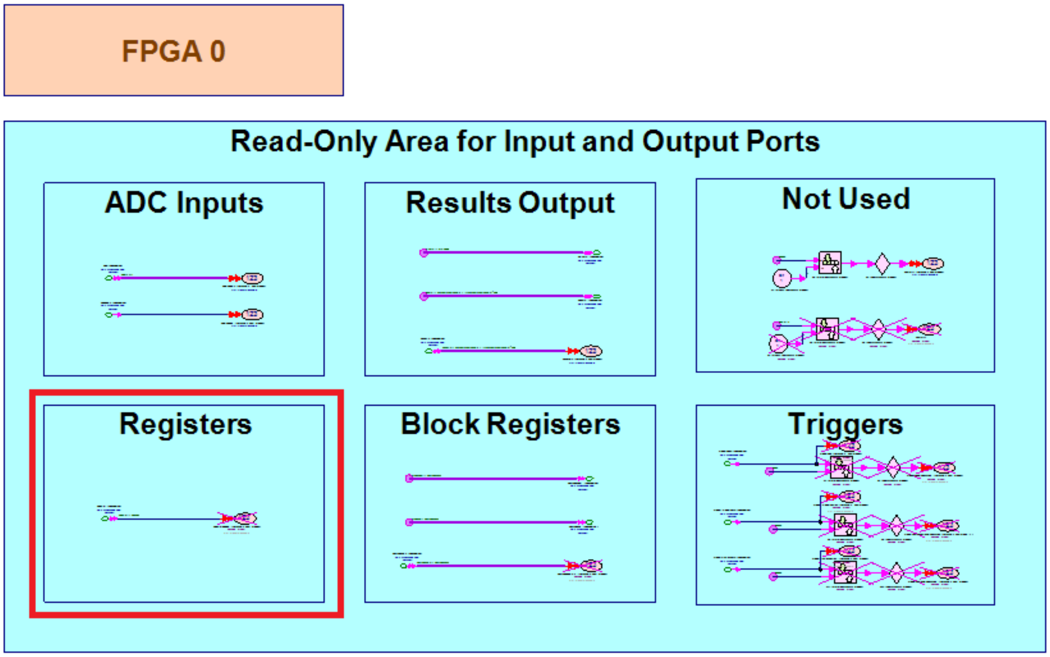
Notes: There is no limitation for the number of Register and the WordLength of each Register.
Sign: 0 -> Unsigned; 1 -> Signed.
The number before each Register Name is its index for "Register" bus port.

You can define the number of Registers and the fixed-point data format of each Register in this GUI. You can also specify Value for each defined Register, then the value of each Register will be set at the beginning of a simulation.

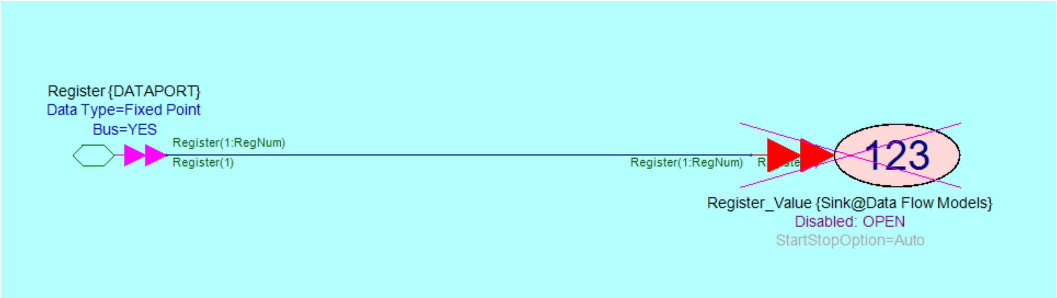
If you generate U5303A FPGA programming file for a U5303A Design Template with defined Registers, the number of Registers and the fixed-point data format of each Register are fixed in FPGA implementation, but you can still re-configure the value of each Register. Then when you run SystemVue and U5303 Co-Simulation, all Register values will be re-configured before ADC data capture and kept until the end of the Co-Simulation.

Connect Register with your FPGA design in subnet U5303_FPGA0:

In subnet U5303_FPGA0 for user's application fixed-point design, Register input bus port has been configured in advance.



If you define Register interface in GUI of U5303A Design Template, you can use the Register input port for your application fixed-point design. Assuming you defined N Registers in GUI, you can use the line with net label Register(x) from Register bus input Register(1:N) to get Register value for your fixed-pointed design.



For example, if you defined two Registers in GUI of U5303A Design Template, the first one with name threshold1 and the second one with name threshold2, assuming that they are the thresholds in your design. You leave the two Registers in order to adjust the thresholds flexibly. Then in U5303_FPGA0 subnet, you can get the values of the defined Registers for your application design in this way:

- Draw a line and double click it to input its net label Register(1). Then this line will be the input of Register threshold1.
- Draw a line and double click it to input its net label Register(2). Then this line will be the input of Register threshold2.

Then you can connect the lines with Register net label to your own fixed-point design.

NOTE If you don't define any **Register** in GUI of U5303A Design Template, you can't use the Register net label. In addition, you can't use the net label **Register(*x*)** that ***x*** is beyond the range you defined.

Besides, your defined Fixed-point data format of each Register must be consistent with your FPGA design, otherwise, an error will be posted when your run SystemVue simulation.

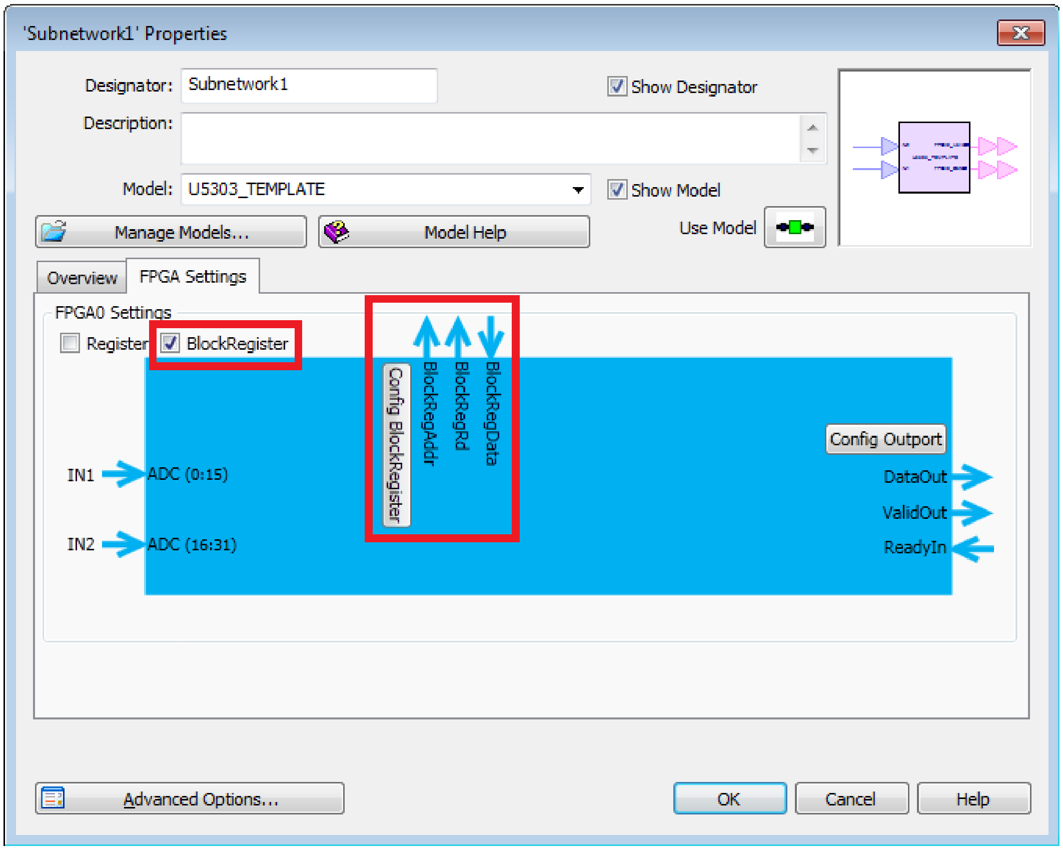
NOTE **Register(*x*)**, where ***x*** is a number between **1 to N** (**N** is the Register number you defined).

### BlockRegister

BlockRegister is an optional FPGA interface. You can define arbitrary number of BlockRegister as long as it can be implemented on DPU FPGA. Each BlockRegister reserves a table of values that can be configured dynamically after FPGA programming file is generated. It can make your generated FPGA programming file more flexible. In your U5303A FPGA design subnet, you can use address to look up the values of the table, just like looking up a RAM.

Configure BlockRegister in U5303A Design Template GUI:

You can enable or disable it in U5303A Design Template GUI by checking or unchecking BlockRegister checkbox as shown below.



'Subnetwork1' Properties

Designator: Subnetwork1 ☑ Show Designator

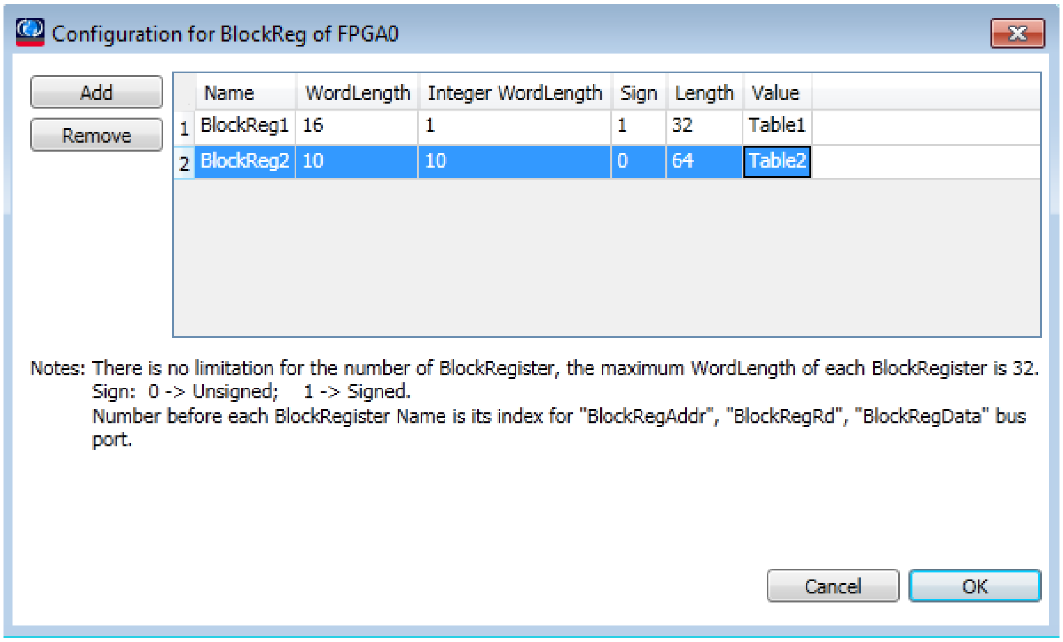
Description:

Model: U5303_TEMPLATE ☑ Show Model

Use Model

BlockRegister is similar as Register interface. It is a table of reserved fixed-point memories (A memory is a block of registers that have the same fixed-point data format. The block of registers can be accessed via address. So a memory is like a RAM.)

After you enable BlockRegister, you can click Config BlockRegister button to open a new GUI:



|   | Name | WordLength | Integer WordLength | Sign | Length | Value |
|---|---|---|---|---|---|---|
| 1 | BlockReg1 | 16 | 1 | 1 | 32 | Table1 |
| 2 | BlockReg2 | 10 | 10 | 0 | 64 | Table2 |

Notes: There is no limitation for the number of BlockRegister, the maximum WordLength of each BlockRegister is 32.
Sign: 0 -> Unsigned; 1 -> Signed.
Number before each BlockRegister Name is its index for "BlockRegAddr", "BlockRegRd", "BlockRegData" bus port.

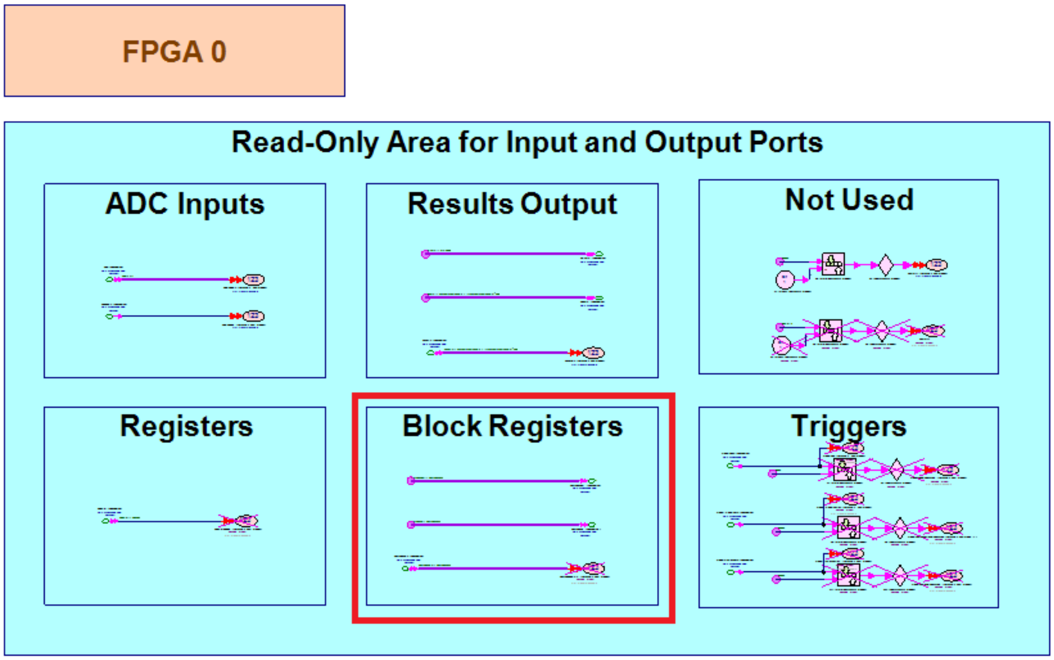
You can define the number of BlockRegister and the fixed-point data format of each BlockRegister in this GUI. You can also specify Length and Value for each defined BlockRegister, Length is the table size and Value is the values of all element of the table. Value must be a row vector whose size is equal to your specified Length value. Then the values of each BlockRegister will be set at the beginning of a simulation.

BlockRegister values can be re-configured before capturing ADC input data, and then the BlockRegister values are kept until the end of simulation. You can define the number of BlockRegisters and the fixed-point data format of each BlockRegister in GUI of U5303A Design Template. You can also specify the values for each defined BlockRegister, then the values of each BlockRegister will be initialized for simulation.

If you generate U5303A FPGA programming file for a U5303A Design Template with defined BlockRegisters, the number of BlockRegisters and the fixed-point data format of each BlockRegister are fixed in FPGA implementation, but you can still re-configure the values of each BlockRegister. Then when you run SystemVue and U5303 Co-Simulation, all BlockRegister values will be re-configured before ADC data capture and kept until the end of the Co-Simulation.

Connect BlockRegister with your FPGA design in subnet U5303_FPGA0:

In subnet U5303_FPGA0 for user's application fixed-point design, BlockRegister ports have been configured in advance. If you define BlockRegister interface in the GUI of U5303A Design Template, you can use the BlockRegister ports for your application fixed-point design.



Because a BlockRegister is like a block of RAM, so it has the memory mapper IO ports. For a BlockRegister, its memory mapped IO ports are:

BlockRegAddr: it is an output bus port of user's application design. Users can use this port to output the address to look up the corresponding value. The fixed-point data format of this port must be:

Word length = ceil( log2(Length of BlockRegister) )
Integer word length = ceil( log2(Length of BlockRegister) )
Unsigned
Length of BlockRegister is defined in the GUI of U5303A Design Template.

BlockRegRd: it is an output bus port of user's application design. Users can use this port to output "Read Enable". When the "Read Enable" is high, the look-up value at the current address will be valid at the next simulation sample.

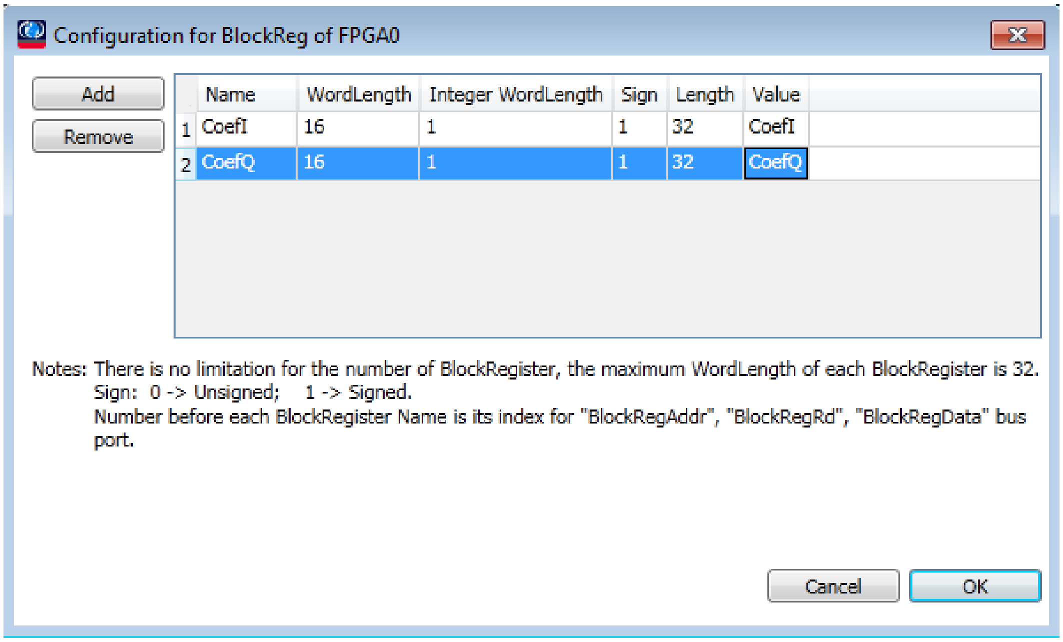
The fixed-point data format of this port must be:
Word length = Integer word length = 1
Unsigned

BlockRegData: it is an input bus port of user's application design. Users can get the look-up value from this port. Note that which input samples of this port are valid depends on output port BlockRegRd.

The fixed-point data format of this port is defined in GUI of U5303A Design Template.

Because you can define multiple BlockRegister in U5303A Design Template GUI, above three ports are all bus ports. Assuming that you defined *N* BlockRegister, the BlockRegister interface in subnet U5303_FPGA0 are three bus ports: BlockRegAddr( *1:N*), BlockRegRd(*1:N*) and BlockRegData(*1:N*). You can use net labels BlockRegAddr(*x*), BlockRegRd(*x*) and BlockRegData(*x*) to access one of the defined BlockRegister.

For example, if you have below BlockRegister definition in the GUI:



|  | Name | WordLength | Integer WordLength | Sign | Length | Value |
|---|---|---|---|---|---|---|
| 1 | CoefI | 16 | 1 | 1 | 32 | CoefI |
| 2 | CoefQ | 16 | 1 | 1 | 32 | CoefQ |

Notes: There is no limitation for the number of BlockRegister, the maximum WordLength of each BlockRegister is 32.
Sign: 0 -> Unsigned; 1 -> Signed.
Number before each BlockRegister Name is its index for "BlockRegAddr", "BlockRegRd", "BlockRegData" bus port.

Assuming that the two BlockRegister are defined for two coefficients re-configurable FIR filter in FPGA0, the length is 32 for both of them.

Then in subnet U5303_FPGA0, you can use the BlockRegister like:

- Connect the "CoefI" address output port of your design to the line with net label BlockRegAddr(1).
- Connect the "CoefI" read enable output port of your design to the line with net label BlockRegRd(1).
- Connect the "CoefI" look-up value input port of your design to the line with net label BlockRegData(1).
- Connect the "CoefQ" address output port of your design to the line with net label BlockRegAddr(2).
- Connect the "CoefQ" read enable output port of your design to the line with net label BlockRegRd(2).
- Connect the "CoefQ" look-up value input port of your design to the line with net label BlockRegData(2).

The fixed point data format of BlockRegAddr(1) and BlockRegAddr(2) must be:

- Word length = Integer word length = 5 (because the length of BlockRegister is defined as 32)
- Unsigned

The fixed point data format of BlockRegRd(1) and BlockRegRd(2) input is:

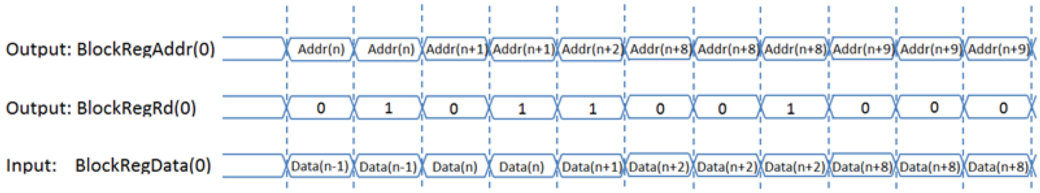
- Word length = 1
- Integer word length = 1
- Unsigned

The fixed point data format of BlockRegData(1) and BlockRegData(2) input is:

- Word length = 16
- Integer word length = 1
- Signed

As defined in GUI.

The timing of BlockRegister interface is shown below:



In users' application design U5303_FPGA0 subnet, you can generate BlockRegAddr and BlockRegRd output and read back BlockRegData. When you output BlockRegRd as 0, BlockRegData will hold the current value at the next clock cycle; While you output BlockRegRd as 1, BlockRegData will be updated to the look-up value of the current address at the next clock cycle.

NOTE

**BlockRegAddr(*x*)**

**BlockRegRd(*x*)**

**BlockRegData(*x*)**

where ***x*** is a number between **1 to N** (**N** is the number of your definedBlockRegister).

### Trigger

In U5303A digitizer, the DPU FPGA has trigger input signals from control FPGA. Users can configure the trigger source, and then once the trigger condition is met, the trigger signals will reach to the DPU FPGA. Users can use the input trigger signals on the DPU FPGA to assist their own real-time signal processing. In SystemVue U5303A FPGA flow, we provide design, simulation, FPGA implementation and digitizer hardware control for trigger.

#### Trigger Interface on DPU FPGA

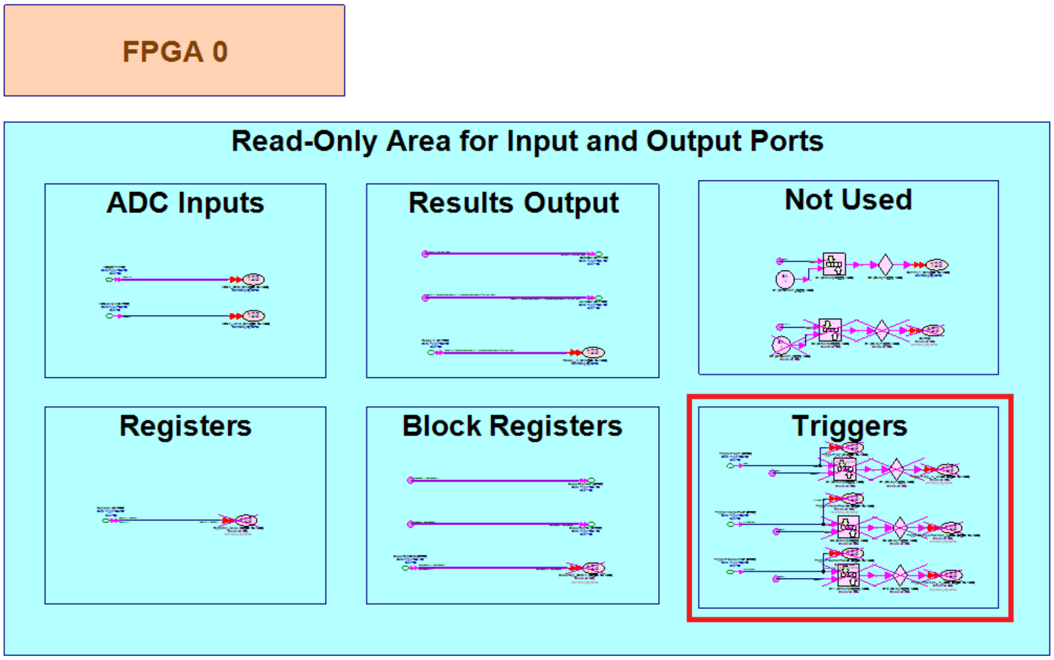
The Trigger interface on each DPU FPGA are 3 input fixed-point signals:

- TriggerFlag: It represents whether it's triggered on the current FPGA system clock cycle. When it's 1, it represents that it's triggered on the current FPGA system clock cycle; When it's 0, it represents that it's not triggered on the current FPGA system clock cycle. Its fixed-point format is <WordLength = 1, IntegerWordLength = 1, Unsigned>.
- TriggerIntegerPos: It represents the ADC sample index before the trigger instant. Because the ADC samples are inputted to DPU FPGA parallel, on each FPGA system clock cycle, there are NumParallel ADC samples are inputted (NumParallel=16 for normal sampling mode and NumParallel=32

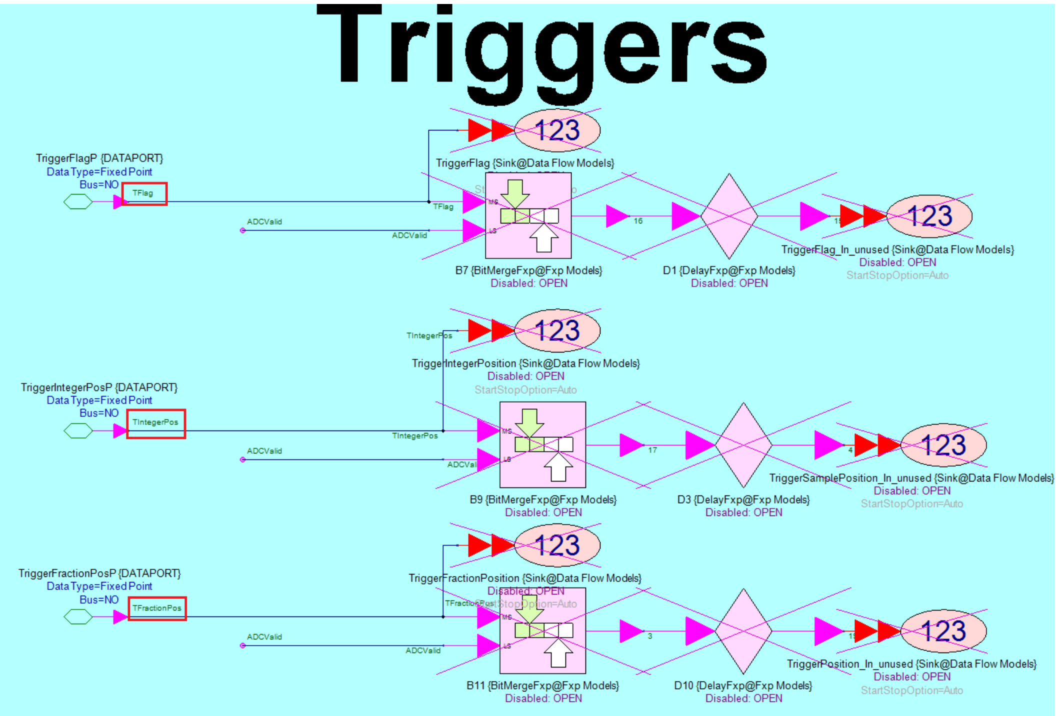
for interleaved sampling model). And also the block of the input ADC samples hold for two FPGA system clock cycles and ADCValid input signal is used to represent whether the block of ADC samples are valid on the current FPGA system clock cycles. So TriggerIntegerPos will be an integer between 0 and NumParallel-1. Its fixed-pointed format is <WordLength=7, IntegerWordLength=7, Unsigned>.

- TriggerFractionPos: It represents the precise trigger position between the trigger instant and the next ADC sample. It's a value in the range [0, 1). Its fixed-pointed format is <WordLength=32, IntegerWordLength=0, Unsigned>.

In the DPU FPGA design subnet U5303_FPGA0, the 3 Trigger input ports are pre-configured.



If you configure to use the trigger interface in your DPU FPGA in U5303A Design Template GUI, you can connect the 3 input ports to your FPGA design using the pre-configured Net Labels: TFlag, TIntegerPos and TFractionPos as shown below.



NOTE

Only when you select to use Trigger interface in U5303A Design Template GUI, the 3 input ports are enabled and you can connect them to your design. (You can also leave them open even you enable the trigger ports.)

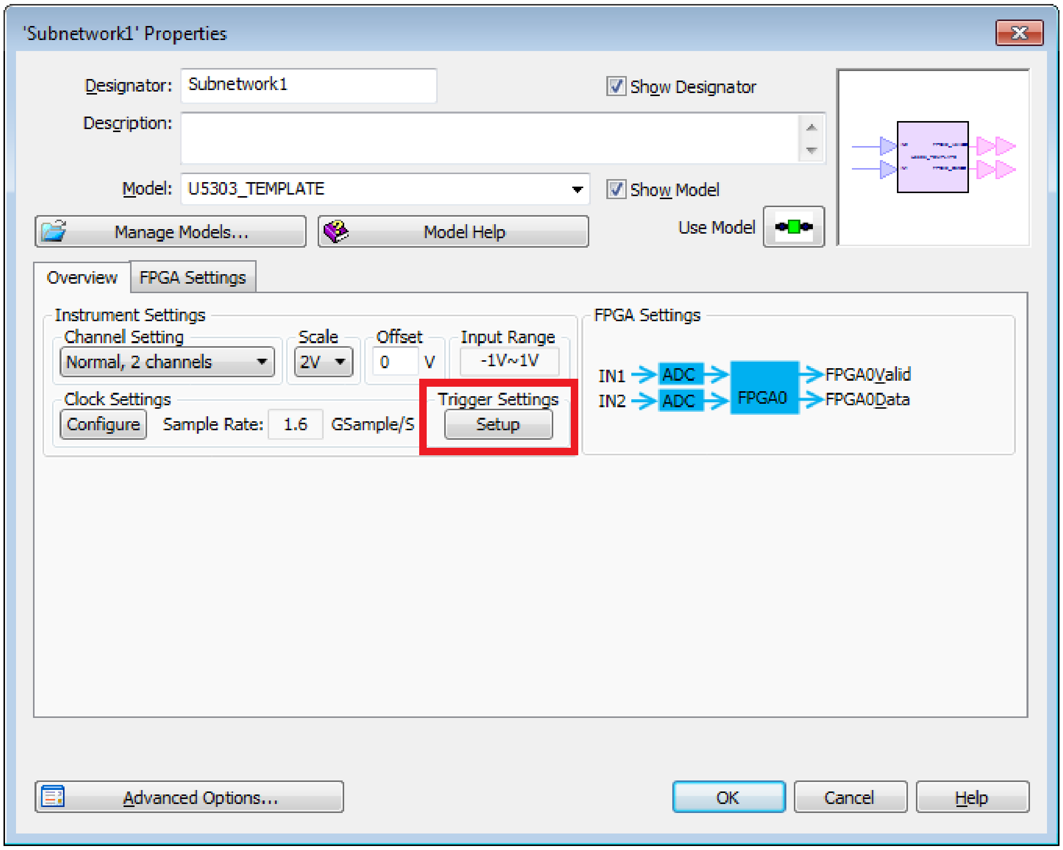
Otherwise, if you configure to not use Trigger interface, the 3 input ports are disabled. You can't connect them to your design.

**Net Label Key Words For Trigger:**

- **TFlag**
- **TIntegerPos**
- **TFractionPos**

## Trigger Simulation in U5303A Design Template

In Overview tab of U5303A Design Template GUI, there is a button for Trigger Settings.



'Subnetwork1' Properties

Designator: Subnetwork1

☑ Show Designator

Description:

Model: U5303_TEMPLATE

☑ Show Model

Use Model

Overview | FPGA Settings

Instrument Settings

Channel Setting: Normal, 2 channels

Scale: 2V

Offset: 0 V

Input Range: -1V~1V

Clock Settings

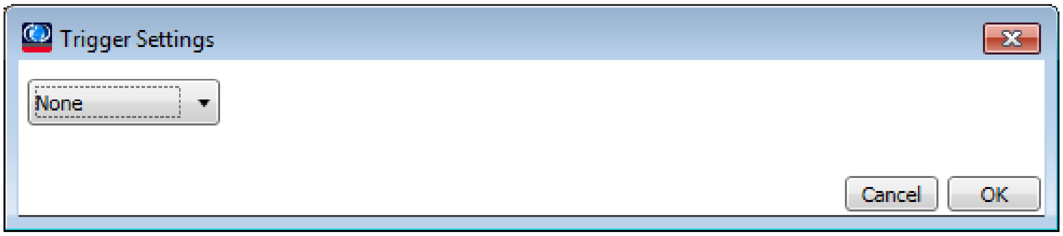
Sample Rate: 1.6 GSample/S

Trigger Settings

You can click it to open a new GUI to configure the trigger mode for U5303A FPGA design and simulation.

You can select to not to use trigger interface or use the trigger interface and specify the triggers time in a time vector.

None

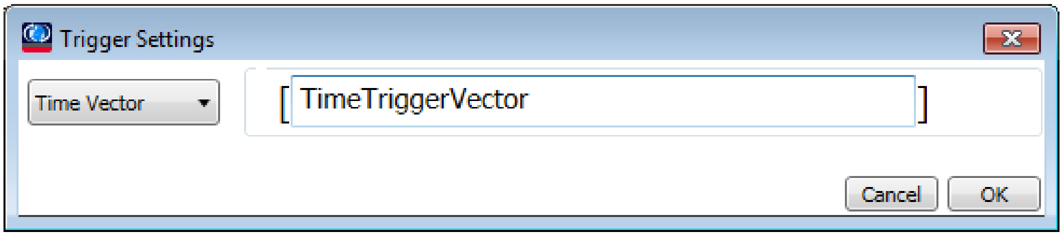


Trigger Settings

None

Trigger interface is not used. The 3 trigger input ports are disabled in U5303_FPGA0 subnet. You can't connect them with your FPGA design.

Time Vector



Trigger Settings

Time Vector

[ TimeTriggerVector ]

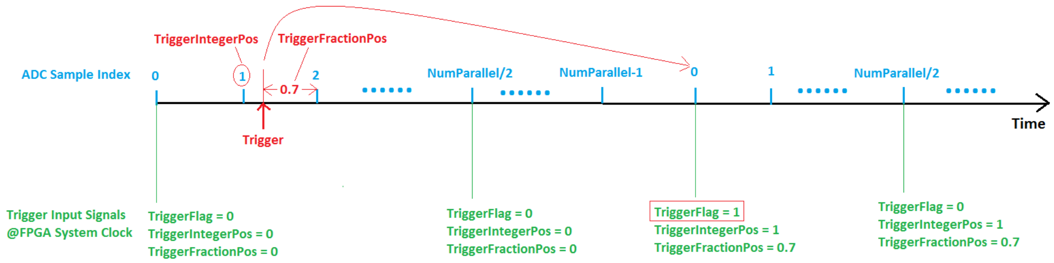
You specify a time vector. On each time point of the time vector, a trigger will be generated. This trigger mode is used to generate triggers at any desired instants. Then you can simulate whether your FPGA design works well with your specified triggers.

A trigger can arrive on any instant, below figure shows the 3 trigger input ports behavior on DPU FPGA when a trigger arrives.

For example, a trigger arrives at 1.3 * (ADC sampling period). The ADC sample index before the trigger instant is 1, so TriggerIntegerPos will be 1 for this trigger. And the ADC sampling period normalized time from trigger instant to the next ADC sample is 0.7, so TriggerFractionPos will be 0.7 for this trigger.

The 3 trigger input ports work at FPGA system clock. And FPGA system clock period is NumParallel/2 times of ADC sampling period. So for each block of NumParallel ADC parallel input samples, there are 2 FPGA system clock cycles. When a trigger arrives, the 3 trigger input signal values will be valid from the next block of NumParallel ADC input samples. And TriggerFlag is 1 for only one FPGA system clock cycles. The values of TriggerIntegerPos and TriggerFractionPos will hold until the next trigger.

When a trigger arrives exactly on an ADC sample position, this ADC sample's index will be TriggerIntegerPos and TriggerFractionPos will be 0. For example, a trigger arrives at 2 * (ADC sampling period). Its TriggerIntegerPos will be 2 and its TriggerFractionPos will be 0.



NOTE

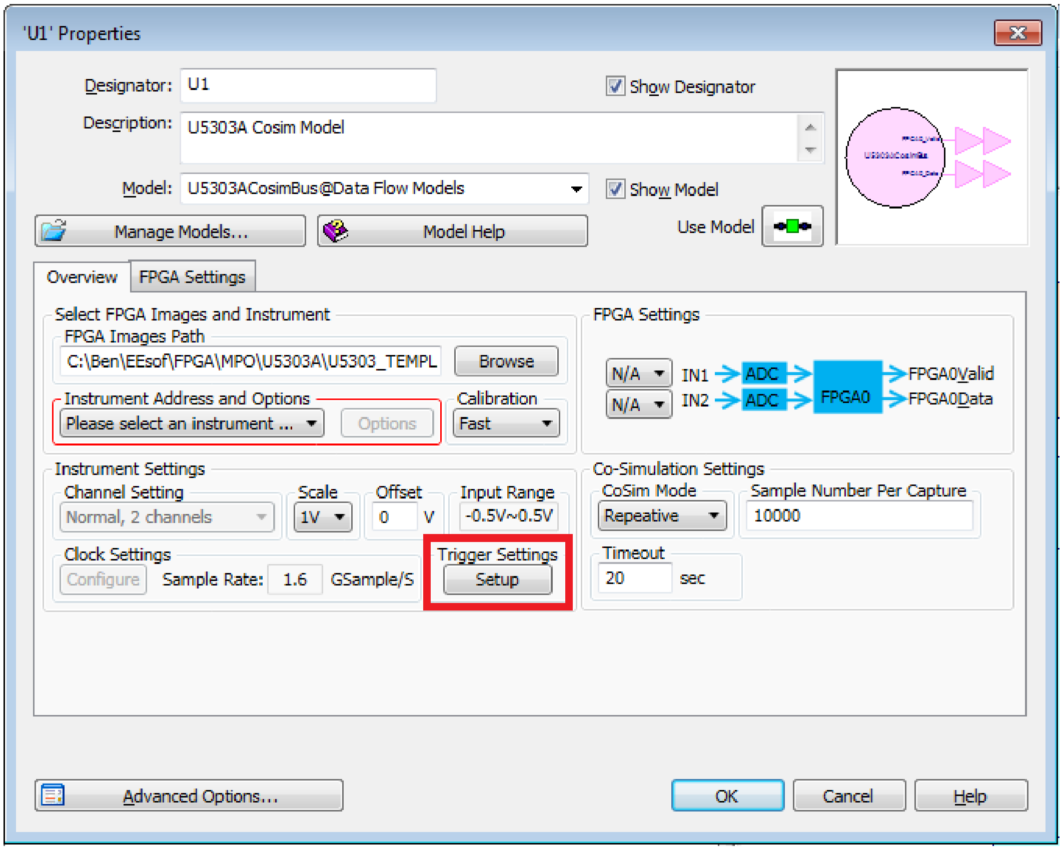
Corresponding to U5303A hardware behavior, if a trigger is too close to its previous trigger, it can't be generated.

So the rule is that when a trigger is generated, the next trigger has to be generated after 40 FPGA system clock cycles. All triggers that are specified within the 40 FPGA system clock cycles will be ignored.

In **U5303_FPGA0** design subnet, the parameter **FPGAWorkingCLKFreq** is the FPGA system clock frequency.

### Trigger Configuration for U5303A Hardware Co-Simulation

In U5303ACosimBus model GUI, there is a Trigger Settings button. You can click it to open a new GUI to configure the actual trigger source for U5303A hardware co-simulation.



'U1' Properties

Designator: U1 ☑ Show Designator

Description: U5303A Cosim Model

Model: U5303ACosimBus@Data Flow Models ☑ Show Model

Manage Models... | Model Help | Use Model

Overview | FPGA Settings

Select FPGA Images and Instrument

FPGA Images Path: C:\Ben\EEsof\FPGA\MPO\U5303A\U5303_TEMPL | Browse

Instrument Address and Options: Please select an instrument ... | Options

Calibration: Fast

FPGA Settings: N/A IN1 ADC; N/A IN2 ADC; FPGA0; FPGA0Valid; FPGA0Data

Instrument Settings

Channel Setting: Normal, 2 channels | Scale: 1V | Offset: 0 V | Input Range: -0.5V~0.5V

Clock Settings: Configure | Sample Rate: 1.6 GSample/S

Trigger Settings: Setup

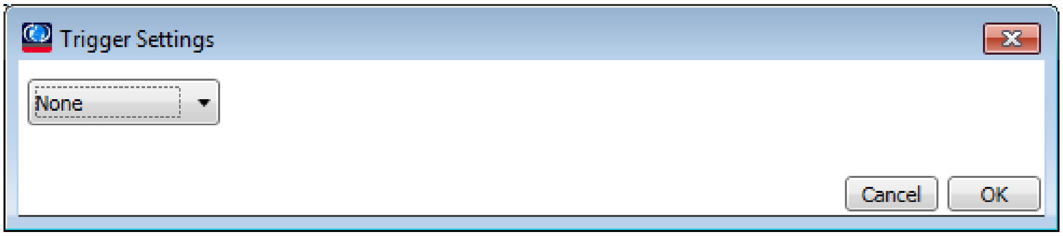
Co-Simulation Settings

CoSim Mode: Repeative | Sample Number Per Capture: 10000

Timeout: 20 sec

Advanced Options... | OK | Cancel | Help

You can select None to disable the actual trigger source. Then TriggerFlag input to DPU FPGA will be 0.

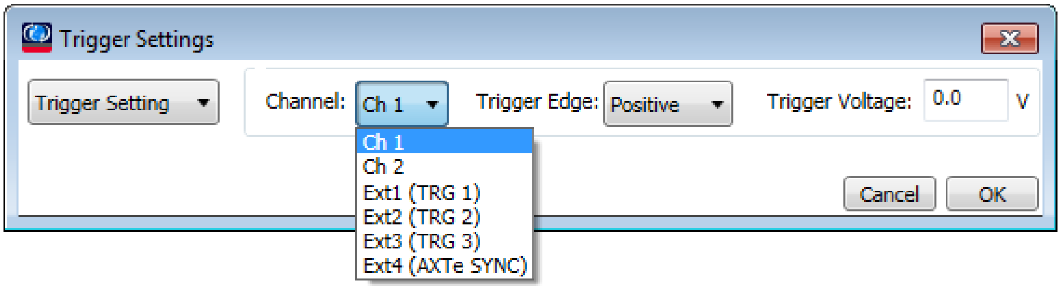


Trigger Settings

None

Cancel | OK

You can select Trigger Setting to configure the actual trigger source.



Trigger Settings

Trigger Setting | Channel: Ch 1 | Trigger Edge: Positive | Trigger Voltage: 0.0 V

Ch 1<br>Ch 2<br>Ext1 (TRG 1)<br>Ext2 (TRG 2)<br>Ext3 (TRG 3)<br>Ext4 (AXTe SYNC)

Cancel | OK

The actual trigger source can be all enabled channels and External Trigger source. External Trigger 1~3 are corresponding to the 3 Trigger Input ports on U5303A digitizer front panel. External Trigger 4 is not used for U5303A.

Trigger Edge:

The Trigger Edge defines which one of the two possible transitions will be used to initiate the trigger when it passes through the specified Trigger Voltage. Positive slope indicates that the signal is transitioning from a lower voltage to a higher voltage. Negative slope indicates the signal is transitioning from a higher voltage to a lower voltage.

Trigger Voltage:

The Trigger Voltage specifies the voltage at which the selected trigger source will produce a valid trigger. All trigger circuits have sensitivity levels that must be exceeded in order for reliable triggering to occur.

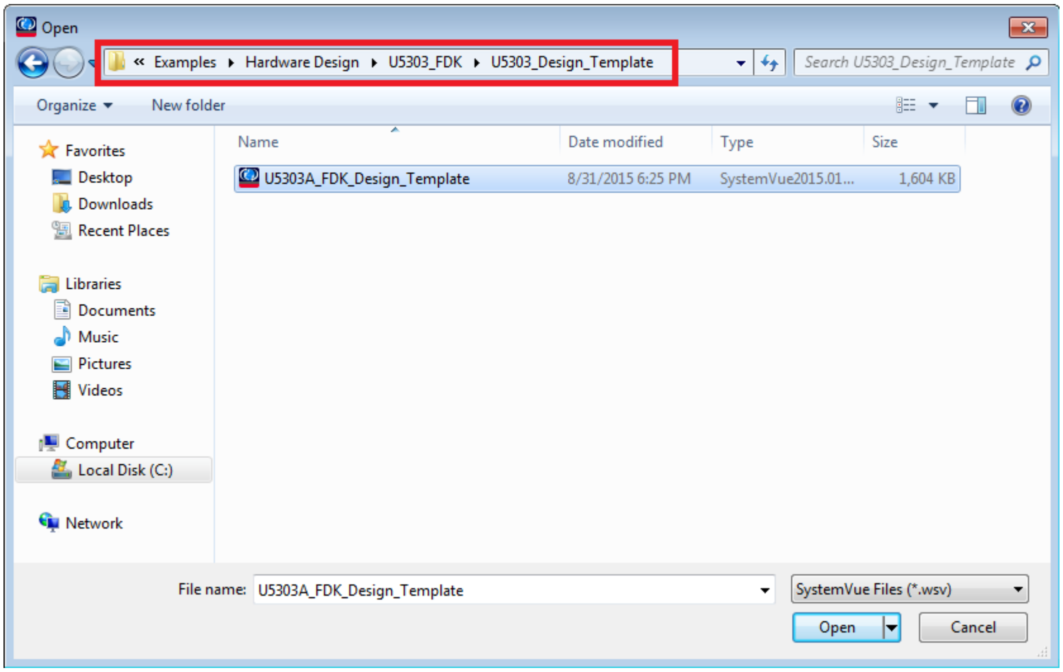
Both the external trigger input and channel triggers have a hysteresis of 5% of Full Scale Range -- The span of the voltage input of the Digitizer (negative to positive) including the configured offset voltage.

On external trigger, the Full Scale Range is ±5 V, therefore the digitizer will trigger on signals with a peak-to-peak amplitude > 0.5 V. The input range of Trigger Voltage is ±5 V for Ext1~Ext4.

When using the channel triggers, the trigger level must be set within Offset ± Scale . For example, when Offset is 0 V and Scale is 2 V, the input range of all enable channels is ±2 V.

## U5303A Design Template GUI

You can find U5303A Design Template from SystemVue Example workspace U5303A_FDK_Design_Template.wsv.



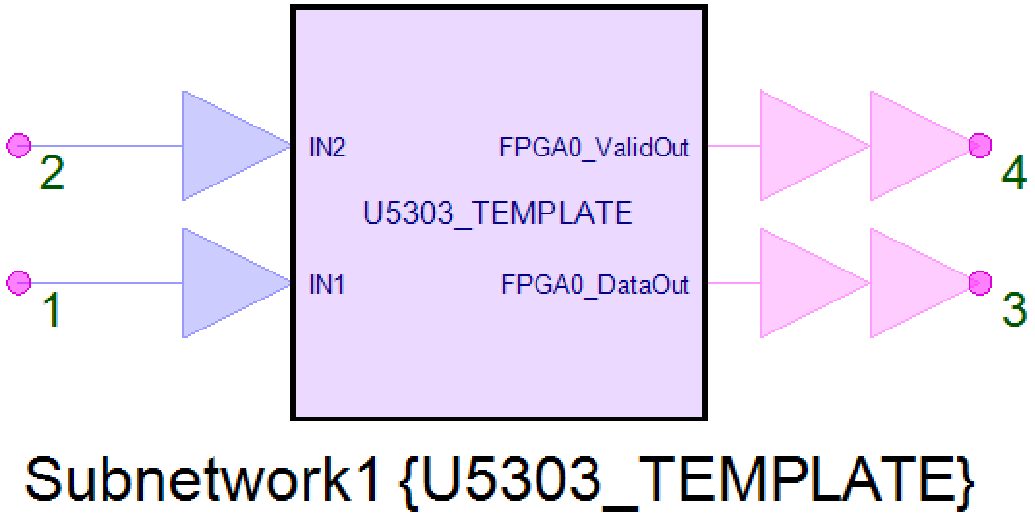
« Examples ▸ Hardware Design ▸ U5303_FDK ▸ U5303_Design_Template

| Name | Date modified | Type | Size |
|---|---|---|---|
| U5303A_FDK_Design_Template | 8/31/2015 6:25 PM | SystemVue2015.01... | 1,604 KB |

File name: U5303A_FDK_Design_Template

SystemVue Files (*.wsv)

You can find subnet U5303_TEMPLATE (model) in SystemVue Workspace Tree of this example and drag-and-drop it to a SystemVue top level schematic. Then you will see the U5303 design template model as show below:



Input Ports:

It has 2 input ports that are all floating point (real) data type input. The input ports IN1 ~ IN2 are corresponding to the input ports on the front panel of U5303A instrument. Because the U5303A design template is used for software simulation, you need create input waveforms for the input ports of U5303A design template in SystemVue. Then you can simulate your U5303A FPGA designs with your input waveforms.

You can configure the clock source and sample mode in the GUI of U5303A Design Template. Then the U5303A's sample rate can be displayed in the GUI. It supports both internal and external clock modes and also interleaved channel mode. When you run simulation, SystemVue will check the sample rates of input waveform and U5303A. If they are not the same, the simulation will post an error.

These input ports are all optional input ports, which means you need not connect all of them. For example, you just want to use DPU FPGA of U5303 in Normal, 2 channels mode, then you can just create input waveforms and connect IN1 and IN2 . If you just want to use DPU FPGA of U5303 in Interleaved, channel 1 mode, then you can just create input waveforms and connect IN1 and leave IN2 input port open. If you just want to use DPU FPGA of U5303 in Interleaved, channel 2 mode, then you can just create input waveforms and connect IN2 and leave IN1 input port open.

NOTE

For all enabled input ports, the SystemVue will check its connectivity when running simulation. For example, if you use DPU FPGA in **Normal, 2 channels** mode, it means you enable **IN1** and **IN2**. Then you have to provide input waveforms for both **IN1** and **IN2**. If you leave one of them open, SystemVue will post an error when you run the simulation.

Output Ports:

It has 2 bus output ports. For DPU FPGA of U5303A, it has a pair of bus output ports, FPGA0_DataOut *and* FPGA0_ValidOut.

The bus widths of FPGA0_DataOut and FPGA0_ValidOut:

The custom real-time processing results are outputted through OutPort in U5303A Design Template subnet U5303_FPGA0. OutPorts are defined in the GUI of U5303A Design Template. You can define OutPorts as two modes: Non-SameSampleRate mode and SameSampleRate mode.

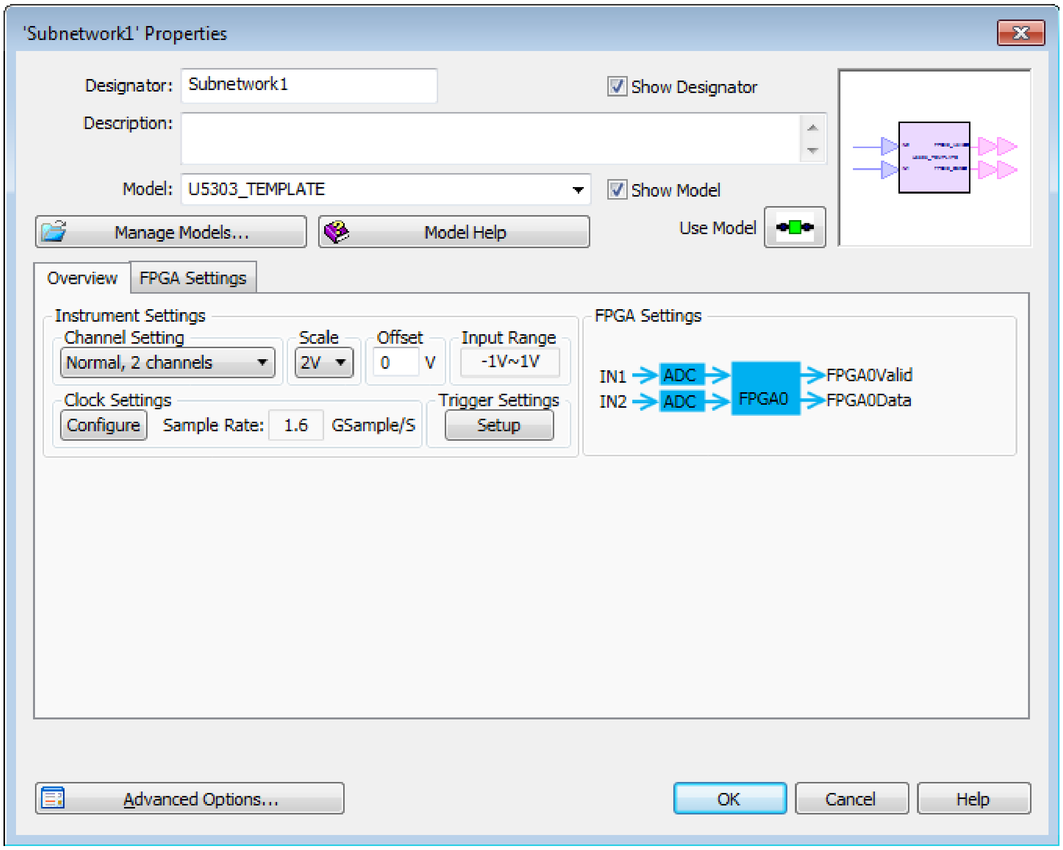
For Non-SameSampleRate mode, the bus widths of FPGA0_DataOut and FPGA0 _ValidOut are the same and are equivalent to the OutPort number that you defined in U5303A Design Template GUI. For example, you define two OutPort in U5303A Design Template GUI for FPGA0. Then FPGA0_DataOut(1) is the output data of your first OutPort of FPGA0 and FPGA0_ValidOut(1) is the corresponding valid indicator of FPGA0_DataOut(1). When FPGA0_ValidOut(1) is 1, it indicates the corresponding data sample of FPGA0_DataOut(1) is valid. Similarly, FPGA0_DataOut (2) and FPGA0_ValidOut(2) are the output pair of the second OutPort of FPGA0.

For SameSampleRate mode, the bus widths of FPGA0 DataOut are equivalent to the OutPort number that you defined in U5303A Design Template GUI. The bus widths of FPGA0_ValidOut is 1. All OutPorts' DataOut share the same ValidOut. For example, you define two OutPorts in U5303A Design Template GUI for FPGA0. Then FPGA0_DataOut(1) is the output data of your first OutPort of FPGA0 and FPGA0_DataOut(2) is the output data of your second OutPort of FPGA0. The output of FPGA0_ValidOut is the valid indicator of both FPGA0_DataOut(1) and FPGA0_DataOut(2). When FPGA0_ValidOut is 1, it indicates the corresponding data sample of FPGA0_DataOut(1) and FPGA0_DataOut(2) are valid.

All output ports are SystemVue fixed-point data type. The fixed-point data format (wordlength, integer wordlength and signed) of FPGA0_DataOut is defined in U5303A Design Template GUI. The FPGA0_ValidOut is one-bit logic output, so its wordlength and integer wordlength are 1 and unsigned.

After drag-and-drop U5303_TEMPLATE subnet from workspace tree to a Schematic, you can double click the U5303A Design Template model in Schematic to open its GUI. You can use this GUI to configure the U5303A DPU FPGAs interfaces.

Overview Tab:



'Subnetwork1' Properties

Designator: Subnetwork1

☑ Show Designator

Description:

Model: U5303_TEMPLATE

☑ Show Model

Channel Setting: Normal, 2 channels

Scale: 2V

Offset: 0 V

Input Range: -1V~1V

Sample Rate: 1.6 GSample/S

The first tab of the GUI is Overview. It is used to configure the top level attributions of U5303A Design Template.

Channel Setting

It has three options:

- Normal, 2 channels: Normal Mode, 2 channels
- Interleaved, channel 1: Interleaved Mode, 1 combined channel, sample rate x2
- Interleaved, channel 2: Interleaved Mode, 1 combined channel, sample rate x2

Scale

It has two options: 1V and 2V. It's the full voltage range for input signal. This parameter in U5303A Design Template is used to mimic the ADC input full scale. There is a corresponding parameter for controlling the real U5303A instrument in U5303ACosimBus model. The input value exceeding this full scale will be limited to the maximum or minimum value of this scale.

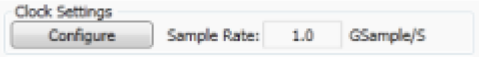
Offset

This parameter is used to mimic the offset adjust of U5303A's input amplifier. There is a corresponding parameter for controlling the real U5303A instrument. The valid range is from -2*Scale to 2*Scale.

Input Range

It is a read-only item to show the valid range of input signal based on specified Scale and Offset parameters.

Clock Settings

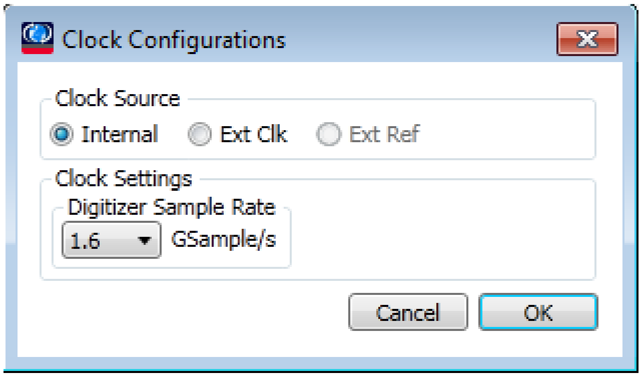

Clock Settings

Sample Rate: 1.0 GSample/S

You can click Configure button to open a new GUI to set the clock and the final sample rate for your current configuration will be shown on the right side.

After clicking Configure button, a new GUI will be opened. You can select to use Internal or External (External Clock Source) clock mode.

When you select Internal clock mode, you have two clock options that can result in two different sample rates. When Channel Setting is set as Normal, 2 channels, you can set the sample rate as 1.0GS/s for SR1 option or 1.6GS/s for SR2 option. While when Channel Setting is set as Interleaved, channel 1 or Interleaved, channel 2, you can set the sample rate as 2.0GS/s for SR1 option or 3.2GS/s for SR2 option.


Clock Configurations

Clock Source

● Internal ○ Ext Clk ○ Ext Ref

Clock Settings

Digitizer Sample Rate

1.6 GSample/s

When you select Ext Clk clock mode, you can set the sample rate in a range. When you use an actual U5303A instrument, you need to provide an external clock input with the corresponding frequency.

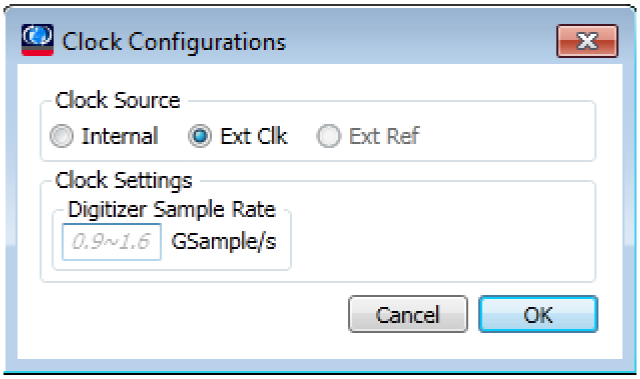
When Channel Setting is set as Normal, 2 channels, the valid sample rate range is 0.9 ~ 1.6 GSample/s. But you need to pay attention to SR1/SR2 option of your U5303A instrument. If your U5303A instrument has SR1 option, the valid sample rate range is only 0.9 ~ 1.0GSample/s, while valid sample rate range is 1.0 ~ 1.6 GSample/s for SR2 option. The U5303A Design Template subnet is only used for FPGA design entry and software simulation, so you can use the full sample rate range, but when you use real U5303A instrument, you need to pay attention to its SR1/SR2 option.

In Normal, 2 channels mode, the real external clock frequency that you need input to U5303A instrument is 2 times of the sample rate.

When Channel Setting is set as Interleaved, channel 1 or Interleaved, channel 2, the valid sample rate range is 1.8 ~ 3.2 GSample/s. But you need to pay attention to SR1/SR2 option of your U5303A instrument. If your U5303A instrument has SR1 option, the valid sample rate range is only 1.8 ~ 2.0GSample/s, while valid sample rate range is 2.0 ~ 3.2GSample/s for SR2 option. The U5303A Design Template subnet is only used for FPGA design entry and software simulation, so you can use the full sample rate range, but when you use real U5303A instrument, you need to pay attention to its SR1/SR2 option.

In Interleaved, channel 1 or Interleaved, channel 2 mode, the real external clock frequency that you need input to U5303A instrument is the same as the sample rate.

The GUI gives you a hint for the valid range.



Clock Configurations

Clock Source

○ Internal ● Ext Clk ○ Ext Ref

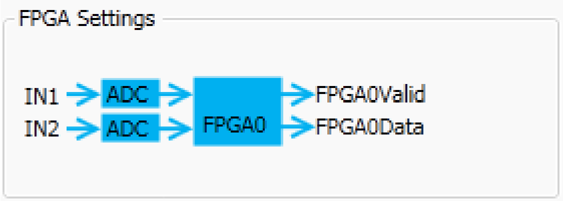
Clock Settings

Digitizer Sample Rate

0.9~1.6 GSample/s

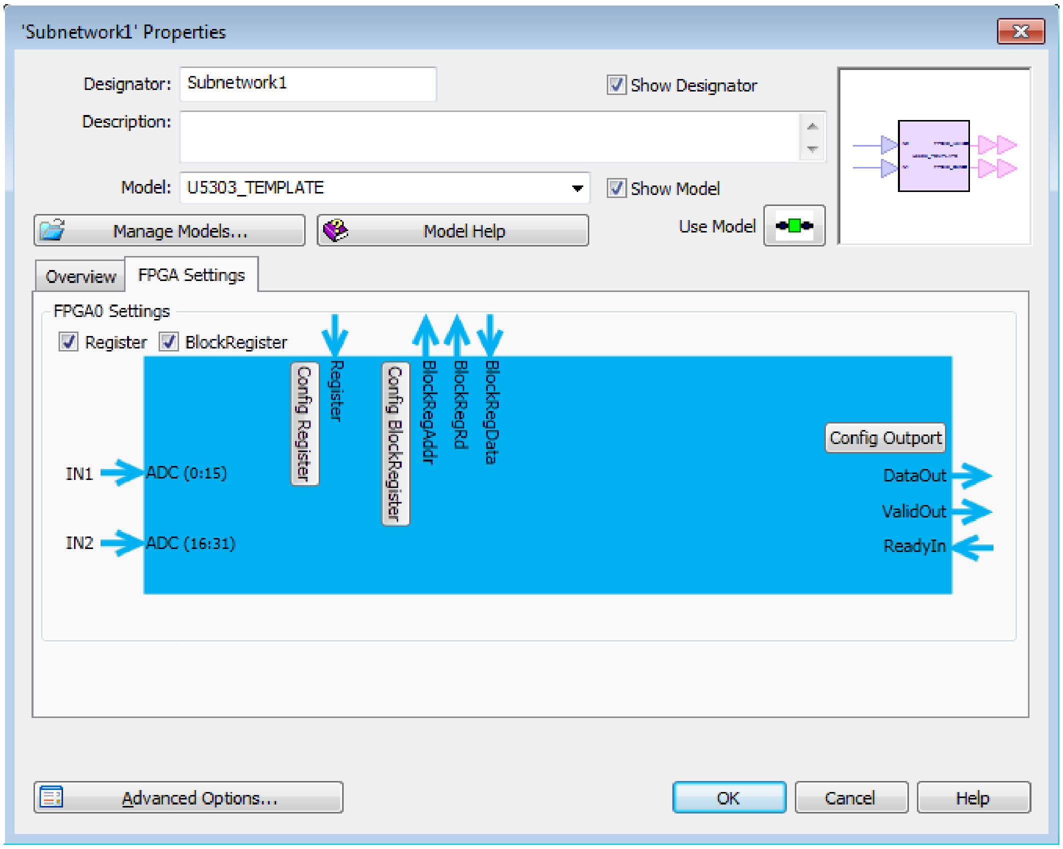
The clock setting configuration doesn't affect the U5303A DPU FPGA programming file generation. So if your U5303A instrument is used with different clock setting that results in different sample rate. Your generated FPGA programming files also works. But a warning will be shown in the GUI of U5303CosimBus model to ask you to notice this difference because it may cause different results in comparison to original software simulation.

FPGA Settings



This figure shows your current DPU FPGA configuration of U5303A. The input ports will be adjusted on this figure according to the specified Channel Setting parameter.

FPGA Settings Tab:



'Subnetwork1' Properties

Designator: Subnetwork1

☑ Show Designator

Description:

Model: U5303_TEMPLATE

☑ Show Model

Use Model

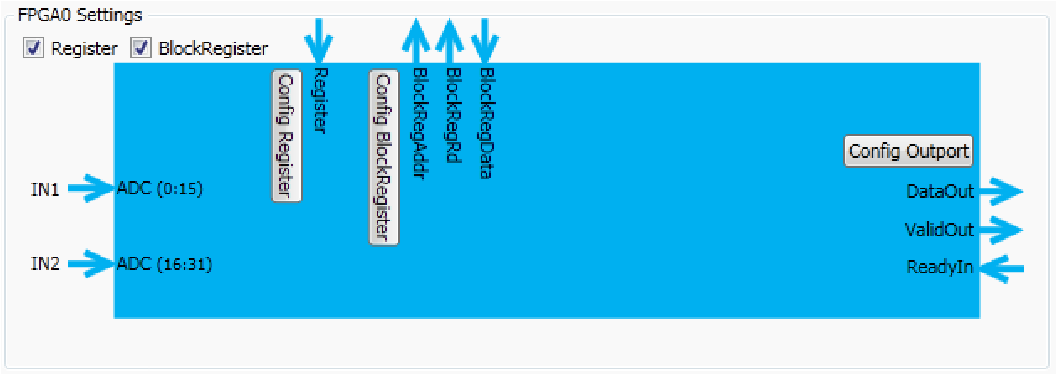
FPGA0 Settings

☑ Register ☑ BlockRegister

In **FPGA Settings** tab, you can define FPGA interface ("Register", "BlockRegister" and "OutPort") for DPU FPGA of U5303A.

FPGA0 Settings

In FPGA0 Settings area, a diagram of DPU FPGA interface is shown.



FPGA0 Settings

☑ Register ☑ BlockRegister

It shows the interface that you configure for DPU FPGA. The U5303_FPGA0 subnet in that you design the custom real-time processing for DPU FPGA will have the corresponding I/O ports according to your configuration in this area. The DPU FPGA has the following interface types:

- ADC Input
- Register

- BlockRegister
- OutPort

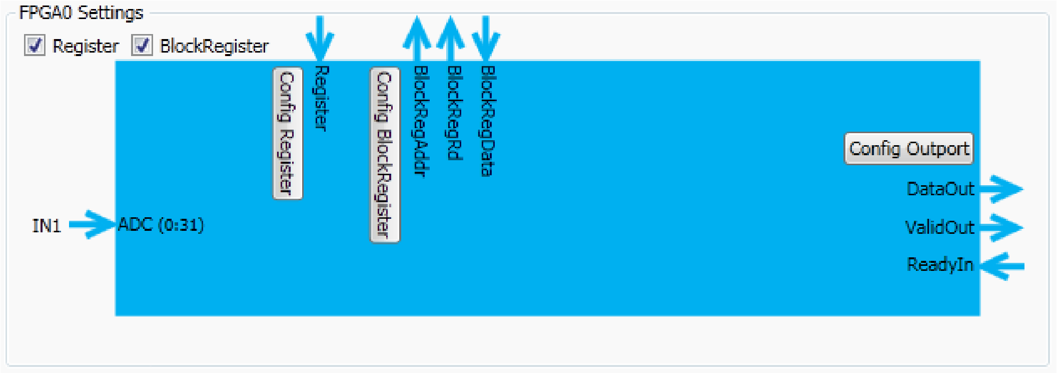
Register and BlockRegister are optional interface types.

ADC Input

The DPU FPGA receives the corresponding ADC input. In Normal, 2 channels mode, the DPU FPGA receives two ADC channels input. In Interleaved, channel 1 or Interleaved, channel 2 mode, the DPU FPGA receives one ADC channel input. In U5303_FPGA0 subnet, there is an input bus ADC(0:31) that you can connect with your real-time processing part to get the parallel ADC input streams. The interface diagram shows how the ADC input parallel streams correspond to ADC(0:31) bus.

For example, if you set Normal, 2 channels mode, IN1 and IN2 are inputted to FPGA0. You can see IN1 is connected to ADC(0:15) and IN2 is connected to ADC (16:31) in the interface diagram. It means that input from IN1 is converted to 16 parallel input streams and you can get these parallel streams from ADC(0:15), where ADC(0) is the oldest sample and ADC(15) is the newest sample. Input from IN2 is converted to 16 parallel input streams and you can get these parallel streams from ADC(16:31), where ADC(16) is the oldest sample and ADC(31) is the newest sample.

If you set Interleaved, channel 1 mode, IN1 is inputted to FPGA0. You can see IN1 is connected to ADC(0:31) in the interface diagram. It means that input from IN1 is converted to 32 parallel input streams and you can get these parallel streams from ADC(0:31), where ADC(0) is the oldest sample and ADC(31) is the newest sample.



FPGA0 Settings

☑ Register ☑ BlockRegister

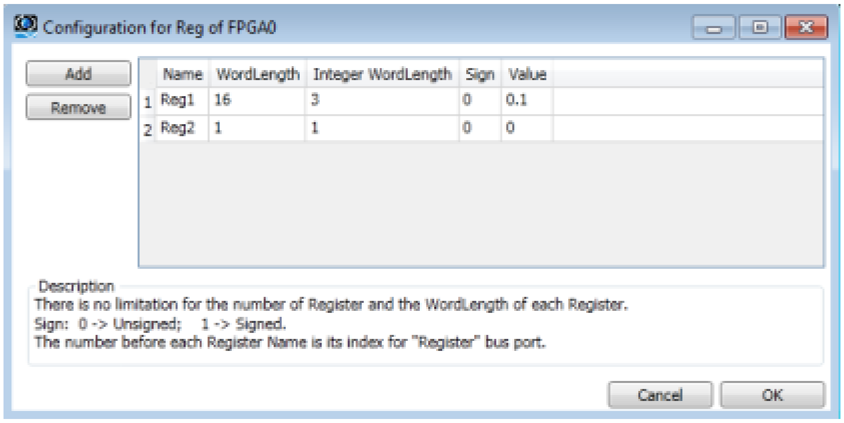
Register

Register Checkbox

Check/uncheck this box to enable/disable the Register interface for the DPU FPGA. When the box is checked, the corresponding Register interface will be shown in the FPGA interface diagram below the checkbox, and a button Config Register is also shown.

Config Register Button

Click this button to open a new GUI to define the Register interface. This button is only shown when Register checkbox is checked.



| | Name | WordLength | Integer WordLength | Sign | Value |
|---|---|---|---|---|---|
| 1 | Reg1 | 16 | 3 | 0 | 0.1 |
| 2 | Reg2 | 1 | 1 | 0 | 0 |

Description
There is no limitation for the number of Register and the WordLength of each Register.
Sign: 0 -> Unsigned; 1 -> Signed.
The number before each Register Name is its index for "Register" bus port.

Click Add button to add a new row to define a new Register.
Click Remove button to remove a selected row (Register).
There is no limitation of the number of Registers, as long as the FPGA has enough area to implement them.

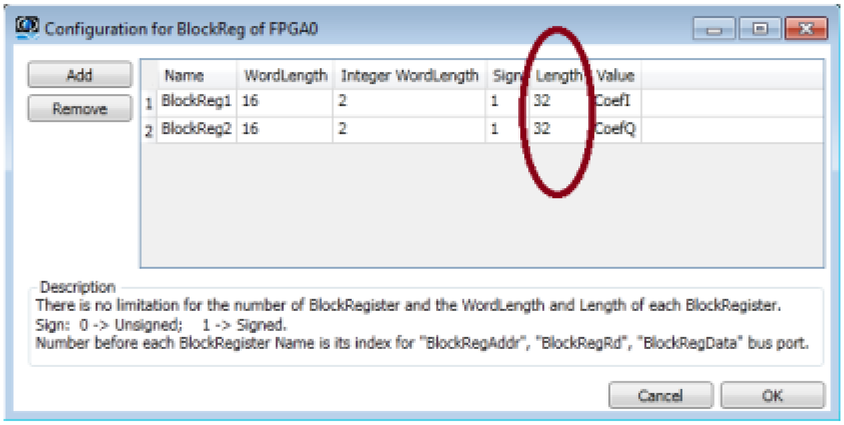
For the definition of a Register, you need input:

- Name: a string to represent the name of the Register.
- WordLength: Word Length (bits number) to represent the Register. Register is a fixed-point data.
- Integer WordLength: Integer Word Length (bits number to represent integer).
- Sign: 0->unsigned; 1->signed.
- Value: The value of current Register. You can input a floating-point value or equation variable name. The floating-point value will be converted to fixed-point type automatically (There may be overflow and quantization for the conversion. The default rule of overflow is Wrap and default rule of quantization is Truncate).

**NOTE** The **Value** of each **Register** must be a **scalar variable** or **a constant**.

BlockRegister

For BlockRegister, it has BlockRegister checkbox and Config BlockRegister button, which is similar as setting of Register. The difference is that when you click Config BlockRegister to open BlockRegister definition GUI, BlockRegister has an additional attribute Length that specified the depth of each BlockRegister. The value of Length must be a scalar variable or a constant integer.



| | Name | WordLength | Integer WordLength | Sign | Length | Value |
|---|---|---|---|---|---|---|
| 1 | BlockReg1 | 16 | 2 | 1 | 32 | CoefI |
| 2 | BlockReg2 | 16 | 2 | 1 | 32 | CoefQ |

Description
There is no limitation for the number of BlockRegister and the WordLength and Length of each BlockRegister.
Sign: 0 -> Unsigned; 1 -> Signed.
Number before each BlockRegister Name is its index for "BlockRegAddr", "BlockRegRd", "BlockRegData" bus port.

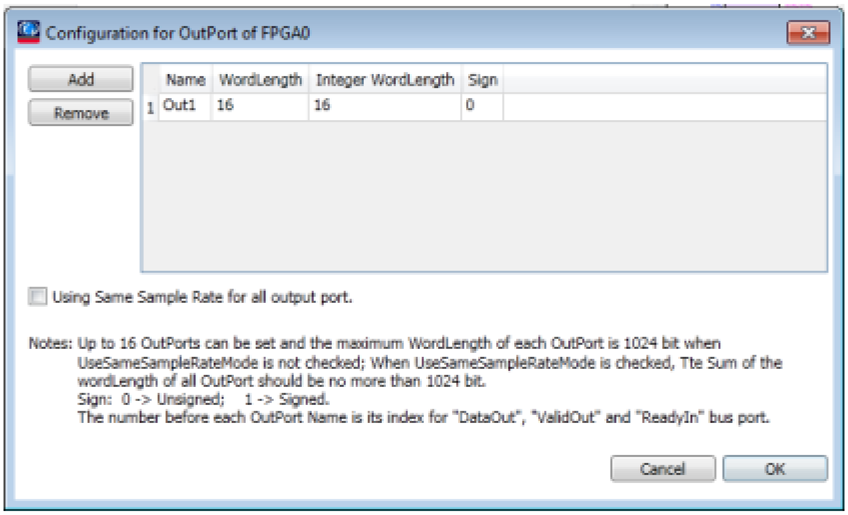
There is no limitation of the number of BlockRegisters and the depth of each BlockRegister, as long as the FPGA has enough area to implement them.

## Note

The Value of each BlockRegister should be a 1xN vector, where N is the value of Length for the corresponding BlockRegister. For example, in above figure, The Value of BlockReg1 CoefI must be a 1x32 vector, as Length of BlockReg1 is 32.

OutPort

OutPort is the mandatory FPGA interface. So it is unlike Register and BlockRegister, you have to define OutPort and there is no checkbox to disable OutPort.
When you click Config Output button, you can open a new GUI to define the OutPort.



| | Name | WordLength | Integer WordLength | Sign |
|---|---|---|---|---|
| 1 | Out1 | 16 | 16 | 0 |

□ Using Same Sample Rate for all output port.

Notes: Up to 16 OutPorts can be set and the maximum WordLength of each OutPort is 1024 bit when UseSameSampleRateMode is not checked; When UseSameSampleRateMode is checked, Tte Sum of the wordLength of all OutPort should be no more than 1024 bit.
Sign: 0 -> Unsigned; 1 -> Signed.
The number before each OutPort Name is its index for "DataOut", "ValidOut" and "ReadyIn" bus port.

You can use Add or Remove button to create a new OutPort or remove a selected OutPort.

When Using Same Sample Rate for all output port is unchecked, each defined OutPort has its own ValidOut signal to indicate whether the current DataOut is valid. So the valid sample rates of OutPorts may be different. In this mode, the maximum number of OutPort is 16 and the maximum WordLength is 1024 for each OutPort.

When Using Same Sample Rate for all output port is checked, all defined OutPorts share the same ValidOut signal to indicate whether the current DataOut is valid. So the valid samples are generated in the same rate for all OutPorts. In this mode, the sum of all OutPorts' WordLength can't be larger than 1024.

The OutPort data type is also SystemVue fixed-point. So you can define the follow attributes of an OutPort:

- Name: a string to represent the name of an OutPort.
- WordLength: Word Length (the bits number) of an OutPort.
- Integer WordLength: Integer Word Length (the bits number to represent integer) of an OutPort.
- Sign: 0->unsigned; 1->signed.

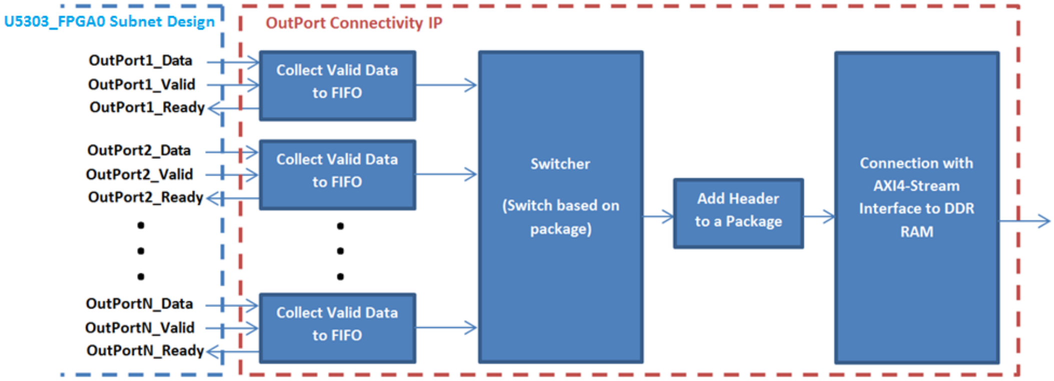
### Software Simulation Behavior Description

After finishing the design entry in U5303A Design Template, users can create test waveforms and input them to the corresponding input ports of U5303A Design Template model. The test waveform can be floating point, so you can use all SystemVue models to create the waveforms. Note that the sample rate of input waveform must be the same as the one that you specified in GUI of U5303A Design Template. Then you can run SystemVue software simulation to debug and verify your U5303A FPGAs design.

The OutPort of U5303A Design Template model have been described in this tutorial . The simulation behavior of output signals will be described in this part.

The FPGA interfaces – ADC input, Register and BlockRegister, are simple and clear. Their behaviors have been described in previous parts. We will emphasis on the simulation behavior of OutPort in this part.

For Non-Same Sample Rate mode, we can define up to 16 OutPorts for DPU FPGA and the maximum word length can be 1024 bits for each OutPort. In the automatic FPGA programming file generation flow, SystemVue will create a HDL wrapper to connect an OutPort Connectivity IP core with the defined OutPorts automatically. The connectivity IP will package and switch the data from all OutPorts to the AXI4-stream interface with data width 256 bits. The AXI4-stream interface provides a data path to send data stream from FPGA to on-board DDR3 SDRAM. The diagram implemented in FPGA programming file is shown below:



The connectivity IP has a FIFO for each OutPort. The FIFO only saves valid data from its corresponding OutPort and generates Ready to its corresponding OutPort. The outputs of all FIFOs are connected to the N-to-1 switcher, where N is the number of defined OutPorts. When the FIFO is full, it outputs Ready as 0 to its corresponding OutPort and isn't written in the new valid data. So only when both Valid and Ready are 1, the OutPort Data can be transferred to DDR3 SDRAM. Current data width of the switch is 128 bit.

In the connectivity IP, the data switching is based on package. It means that every time the switcher arbitrates and selects an OutPort, it will read a package from the selected OutPort FIFO. It will not arbitrate until the current package is transferred completely. The connectivity IP will also add a header at the beginning of each package to indicate which OutPort the current package comes from. The data length of a package is 64 times of data width of DDR3 SDRAM AXI4-stream interface (64*256 bits). The header length of a package is 2 times of data width of DDR3 SDRAM AXI4-stream interface (2*256 bits). So the total length of a package is 66 times of data width of DDR3 SDRAM AXI4-stream interface (66*256 bits). As the word length of an OutPort is an arbitrary number less than 1024, the connectivity IP will combine the valid data into a package and it may add some zero-padding bits at the end of a package.

So the total effective throughput (discarding header) to DDR3 SDRAM is (128 * 64 / 66) bit X FPGA working frequency, where 128 is N-to-1 switch data width.

In U5303A Design Template subnet, M9703AMemoryBus model is used to mimic the behavior of the connectivity IP. OutPorts of U5303_FPGA0 user design subnet are connected to M9703AMemoryBus model, and this model generates Ready input of U5303_FPGA0 subnet.

The connectivity IP is transparent to users and it's generated automatically according to the OutPort definition. The purpose of the connectivity IP description is to help you to understand the following rules for your U5303_FPGA0 subnet design:

1. In real FPGA running, only when both ValidOut and ReadyIn of an OutPort are 1, the corresponding DataOut is transferred to DDR3 SDRAM. Only the data buffered in DDR3 SDRAM can be read back to SystemVue during U5303A instrument co-simulation.

2. In software simulation, DataOut output samples of an OutPort of U5303_FPGA0 subnet are not the same as the corresponding DataOut output samples of U5303A Design Template top level subnet. But if you extract all DataOut output samples transferred to DDR RAM (the corresponding ValidOut and ReadyIn are both 1) of an U5303_FPGA0 subnet's OutPort as the results sequence of this OutPort, and extract all DataOut output samples that their corresponding ValidOut are 1 for an U5303A Design Template subnet' OutPort as this OutPort's results sequence. The two results sequences are the exactly same.
3. In U5303 instrument co-simulation, DataOut output samples of an U5303ACosimBus model's OutPort are not the same as DataOut output samples of the corresponding U5303A Design Template subnet's OutPort in software simulation. But if you extract DataOut output samples that their corresponding ValidOut are 1 as results sequences, the results sequences from U5303A instrument co-simulation and software simulation are the exactly same.
4. As we just care about the valid output samples, above 2 and 3 can guarantee the valid output samples of an OutPort are the same between software simulation and U5303A instrument co-simulation.
5. If the sum of all OutPorts' word lengths is not larger than (128 * 64 / 66) ≈ 124 bits (the effect throughput to DDR3 SDRAM), even ValidOut of all OutPorts are always 1, the AXI4-Stream switch has enough bandwidth to transfer DataOut of all OutPorts to DDR3 SDRAM. So when the sum of all OutPorts' word lengths is not larger than 128 bits, ReadyIn of all OutPorts are always 1. You can use this condition to simply your U5303_FPGA0 subnet design.

When the sum of all OutPorts' word lengths is larger than 124 bits, it may cause the connectivity IP outputs 0 to ReadyIn of an OutPort. Then you need add some logics in U5303_FPGA0 subnet design to buffer valid DataOut samples, if you don't want to break the continuous data stream.

In SystemVue Example workspace U5303A_FDK_Design_Template.wsv, there is a subnet "ValidExtract".You can use this subnet to only extract valid data samples, but it uses Dynamic Data Flow so need notice the limitation of Dynamic Data Flow.

For Same Sample Rate mode, it's actually a special case of Non-Same Sample Rate mode. Because all OutPort has the same ValidOut, SystemVue combines them together in background before switcher and just use a 1-to-1 switcher.

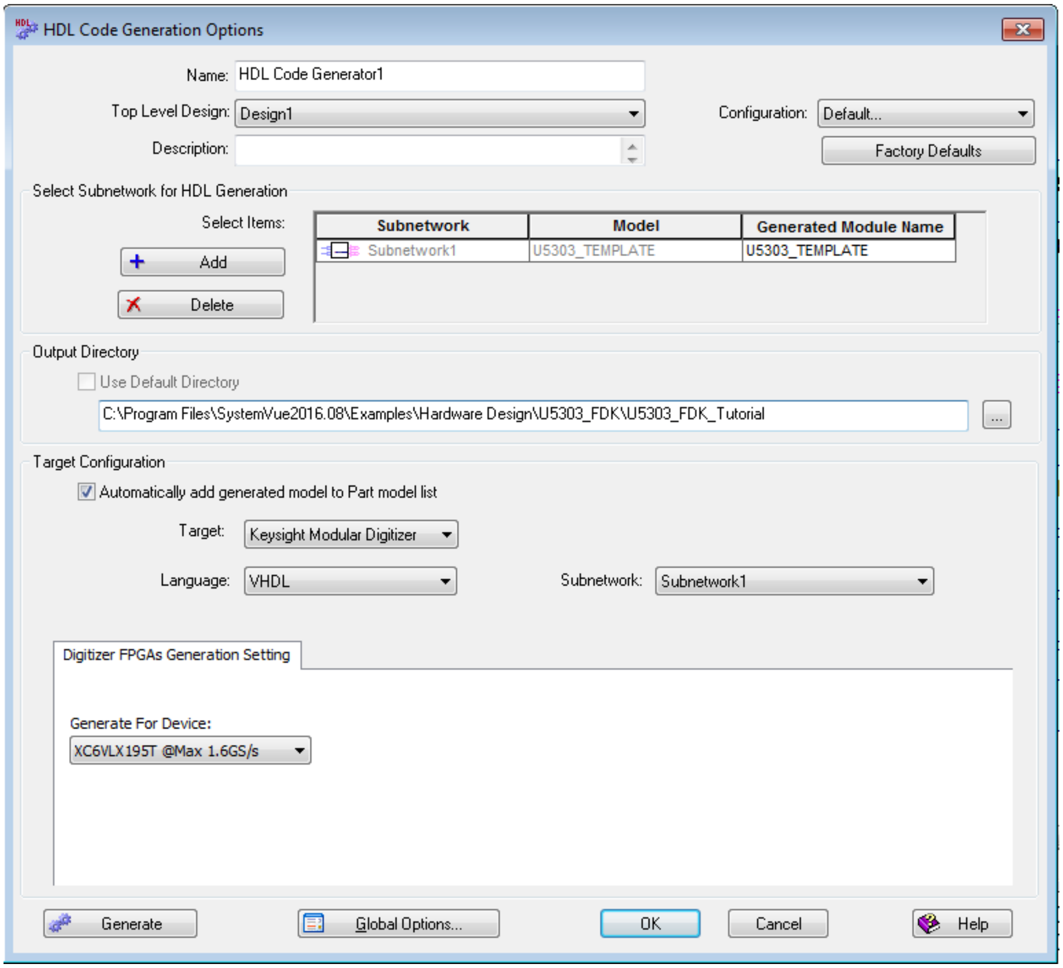
## U5303A FPGA Programming File Generation

After the verification of SystemVue software simulation, you can generate U5303A FPGA programming file with an automatic flow in SystemVue HDL Code Generator. For information on how to add a HDL Code Generator, please refer to HDL Code Generation. In HDL Code Generator, you can select the Target as Keysight Modular Digitizer. Then Click Add to select the U5303A Design Template in your current schematic to generate FPGA programming file.

Please note that only M9703A Design Template (subnet name: M9703_TEMPLATE) and U5303A Design Template (subnet name: U5303_TEMPLATE) subnets are valid subnet for Keysight Modular Digitizer mode, this GUI will prompt error if you select an invalid subnet.

For U5303A, you have 3 options of Generate For Device.

- If your U5303A has -LX2 option, it's 4 DPU FPGA are all Xilinx XC6VLX195T FPGAs. Then if your U5303A has -SR2 option, it's sample rate is 1.6GS/s, then you should select the option XC6VLX195T @Max 1.6GS/s.
- If your U5303A has -LX2 and -SR1 option, its DPU FPGAs are Xilinx XC6VLX195T and its sample rate is 1.0GS/s, then you should select the option XC6VLX195T @1.0GS/s.
- While if your U5303A has -SX3 option, its DPU FPGAs are all Xilinx XC6VSX315T FPGAs. U5303A with -SX3 option can only have -SR2 option (1.6GS/s sample rate). So you should select the option XC6VSX315T @Max 1.6GS/s.


HDL Code Generation Options

Name: HDL Code Generator1

Top Level Design: Design1

Configuration: Default...

Description:

Select Subnetwork for HDL Generation

Select Items:

| Subnetwork | Model | Generated Module Name |
|---|---|---|
| Subnetwork1 | U5303_TEMPLATE | U5303_TEMPLATE |

Output Directory

□ Use Default Directory

C:\Program Files\SystemVue2016.08\Examples\Hardware Design\U5303_FDK\U5303_FDK_Tutorial

Target Configuration

☑ Automatically add generated model to Part model list

Target: Keysight Modular Digitizer

Language: VHDL

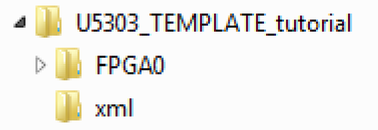
Subnetwork: Subnetwork1

Digitizer FPGAs Generation Setting

Generate For Device: XC6VLX195T @Max 1.6GS/s

Then just click Generate. An automatic flow will be launch until you get the final U5303A FPGA programming files.

You can specify Generated Module Name in HDL Code Generator GUI. Then SystemVue will generate a folder with the name of specified Generated Module Name in the Output Directory. The generated file structure is shown below:



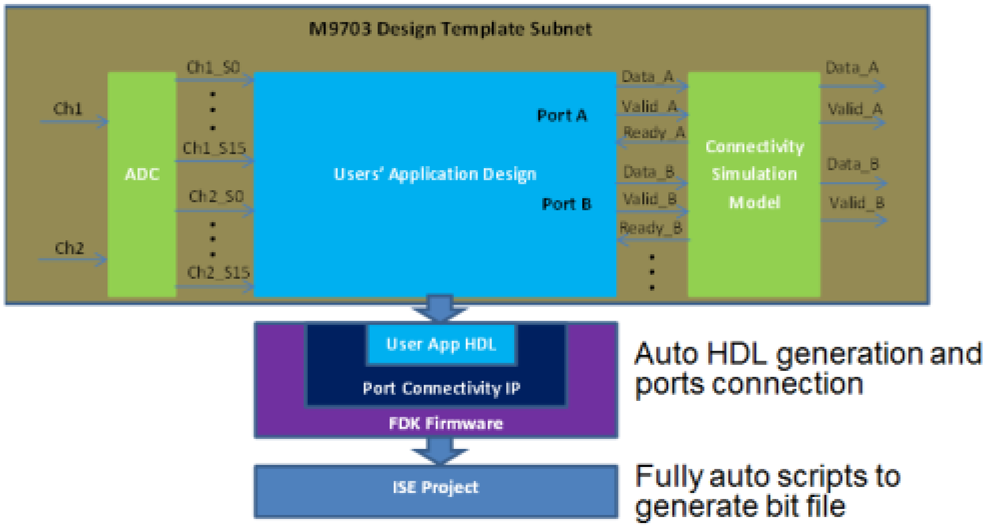
U5303_TEMPLATE_tutorial
FPGA0
xml

For example, if you specify Generated Module Name as U5303_TEMPLATE_tutorial, then a folder U5303_TEMPLATE_tutorial will be generated in the Output Directory. There will be an xml folder and the corresponding FPGA0 folders under the U5303_TEMPLATE_tutorial folder. The generated FPGA programming file and ISE project are under the corresponding FPGA0 folder. Two xml files are generated in xml folder to describe the generated FPGA configuration for the whole U5303A instrument.

In GUI of U5303ACosimBus model, there is a parameter FPGA Images Path. You can select the top level folder, such as U5303_TEMPLATE_tutorial folder, for this parameter. Then the GUI of U5303ACosimBus can parse the folder structure and xml file to understand your generated FPGA configuration.

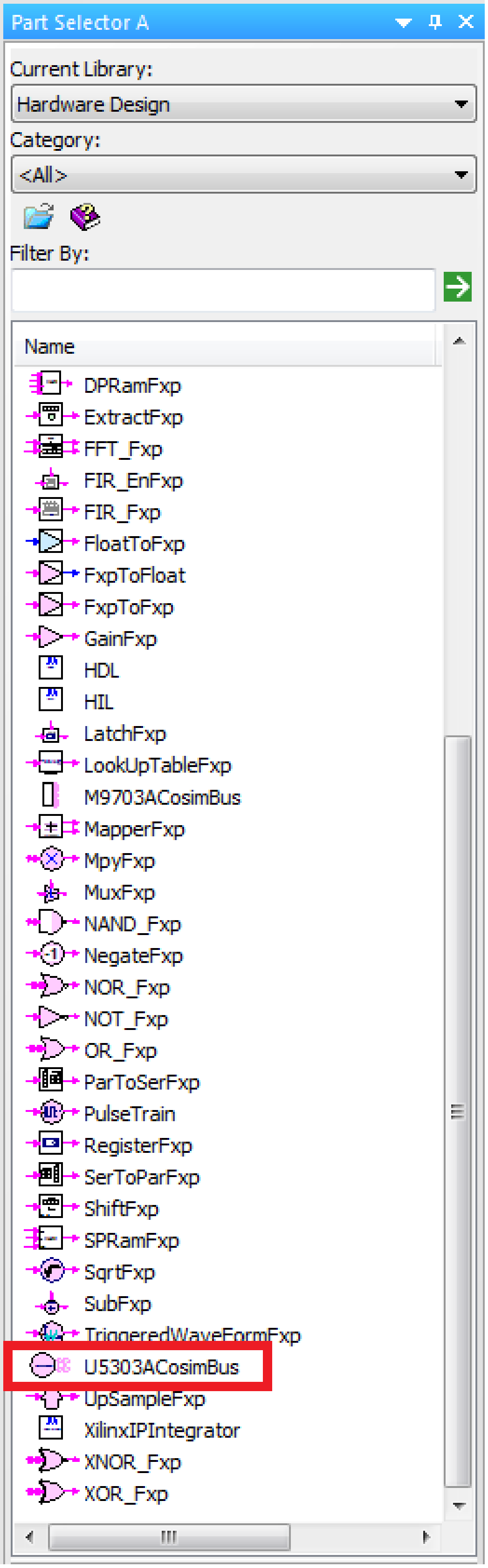
**NOTE** Do not modify the generated FPGA folder manually. It may cause that SystemVue can't pares it correctly and work with U5303A instrument correctly.

After clicking Generate button, SystemVue generates HDL codes for selected subnet application design at first. Then SystemVue generates a HDL wrapper file to connect with the generated application HDL codes automatically. The HDL wrapper file contains some connectivity IP to link your defined FPGA interface with U5303A FPGA infrastructure design that implements some foundational functions, such as link with ADC data stream, DDR3 SDRAM and PCI Express on backplane. Then SystemVue will create an ISE project and run this project to get the final FPGA programming file. The whole flow is automatic. A diagram of the generation flow is shown below.



### U5303A Instrument Co-simulation with SystemVue

After the FPGA programming file is generated, you can use U5303ACosimBus model to download your FPGA programming file to U5303A instrument and capture the output results of FPGA to SystemVue simulation environment. You can find U5303ACosimBus model in Part Selector under Hardware Design.

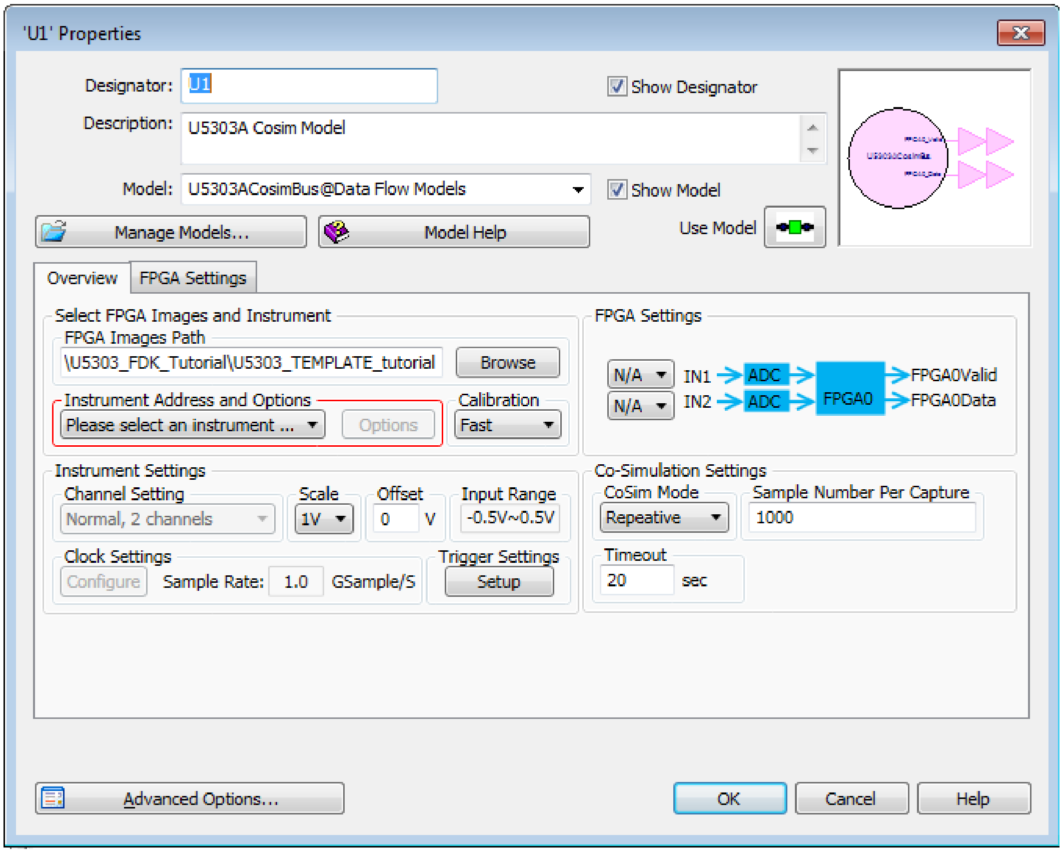


Part Selector A
Current Library:
Hardware Design
Category:
<All>
Filter By:
Name
DPRamFxp
ExtractFxp
FFT_Fxp
FIR_EnFxp
FIR_Fxp
FloatToFxp
FxpToFloat
FxpToFxp
GainFxp
HDL
HIL
LatchFxp
LookUpTableFxp
M9703ACosimBus
MapperFxp
MpyFxp
MuxFxp
NAND_Fxp
NegateFxp
NOR_Fxp
NOT_Fxp
OR_Fxp
ParToSerFxp
PulseTrain
RegisterFxp
SerToParFxp
ShiftFxp
SPRamFxp
SqrtFxp
SubFxp
TriggeredWaveFormFxp
U5303ACosimBus
UpSampleFxp
XilinxIPIntegrator
XNOR_Fxp
XOR_Fxp

## U5303A Co-Simulation Model GUI

Place one U5303ACosimBus model in a schematic to instantiate one U5303A instrument, and you can place multiple U5303ACosimBus models to ask multiple U5303A instruments to work together in one SystemVue simulation.
Double click the U5303ACosimBus model, you can open its GUI. You can use this GUI to configure a U5303A instrument. The GUI of U5303ACosimBus has a similar architecture of the GUI of U5303 Design Template.

Overview Tab:



'U1' Properties

Designator: U1 ☑ Show Designator

Description: U5303A Cosim Model

Model: U5303ACosimBus@Data Flow Models ☑ Show Model

Use Model

Overview | FPGA Settings

Select FPGA Images and Instrument

FPGA Images Path: \U5303_FDK_Tutorial\U5303_TEMPLATE_tutorial

Instrument Address and Options: Please select an instrument ...

Calibration: Fast

FPGA Settings: N/A IN1 → ADC → FPGA0 → FPGA0Valid; N/A IN2 → ADC → FPGA0 → FPGA0Data

Instrument Settings

Channel Setting: Normal, 2 channels | Scale: 1V | Offset: 0 V | Input Range: -0.5V~0.5V

Clock Settings: Sample Rate: 1.0 GSample/S

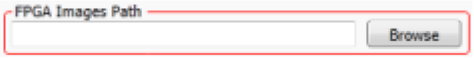
Trigger Settings

Co-Simulation Settings

CoSim Mode: Repeative | Sample Number Per Capture: 1000

Timeout: 20 sec

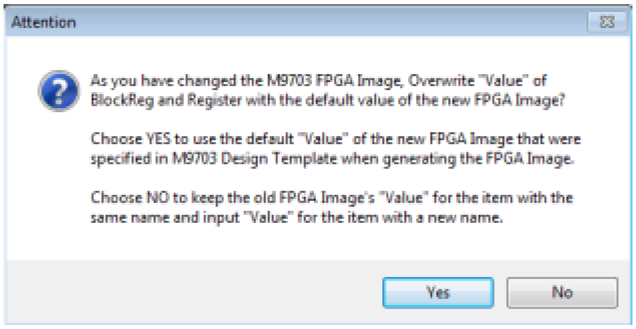
### FPGA Images Path



Specify the generated U5303A FPGA image folder. You should specify the top level folder of a generated U5303A FPGA image.

Then the GUI can parse the generated FPGA image and extract the configuration that was implemented in the FPGA programming files. It can also find the generated FPGA programming files.

When you change FPGA Images Path, the below window will pop-up to ask you how to specify the values for the defined Register and BlockRegister.


Attention

As you have changed the M9703 FPGA Image, Overwrite "Value" of BlockReg and Register with the default value of the new FPGA Image?

Choose YES to use the default "Value" of the new FPGA Image that were specified in M9703 Design Template when generating the FPGA Image.

Choose NO to keep the old FPGA Image's "Value" for the item with the same name and input "Value" for the item with a new name.

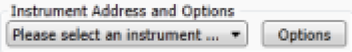
For a generated FPGA image, the configuration of Register and BlockRegister, such as WordLength, Integer WordLength, Sign and Length, are fixed and you can't modify them any more. But you can modify the values for the defined Register and BlockRegister. So if you change FPGA Images Path from an old FPGA image to a new FPGA image, you can specify whether you use the default values of new FPGA image or remain the values you specified for the old FPGA image.

The default values of the new FPGA image are specified in U5303 Design Template custom UI and recorded in the automatic FPGA image generation flow.

If you select to use the values of old FPGA image, all Register and BlockRegister items that have the same names between the new and old FPGA images will remain their values. For every new FPGA image's Register and BlockRegister item that has not the same name in the old FPGA image, the GUI will prompt you to input its value.

For example, if the old FPGA image has Registers with names Reg1 and Reg2 and the new FPGA image has Registers with name Reg2 and Reg3. You have selected the old FPGA image and specified Reg1=1 and Reg2=2 in the GUI of U5303ACosimBus model. Then you change FPGA Images Path to select the new FPGA image. The pop-up window will show. If you select Yes to use the default values of new FPGA image, the values of Reg2 and Reg3 of the new FPGA image will be loaded according to their default values. Otherwise, you select No to remain the values of the old FPGA image, Reg2 will keep its value 2 because there is a Register with the same name in the old FPGA image. And the GUI will prompt you to input a new value for Reg3 because there is no Register with the same name in the old FPGA image.

Instrument Address and Options

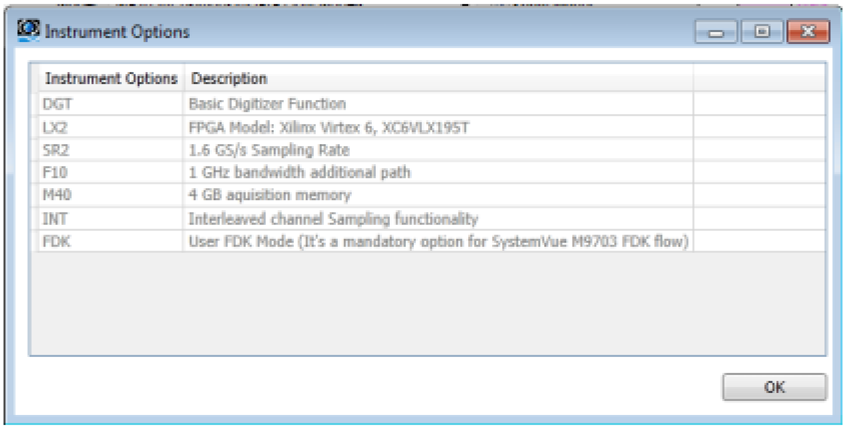

Instrument Address and Options
Please select an instrument ...
Options

Specify an U5303A instrument according to its address. When you open the GUI, it will detect all U5303A instruments connected to this computer and list the detected U5303A instruments addresses. Then you can select an U5303A instrument.

The default value of instrument address is a blank option and shows Please select an instrument ... in GUI. When you select a detected real U5303A instrument address, the GUI will initialize the selected U5303A instrument and extract its hardware options. It usually takes several seconds to initialize a U5303A instrument.

If you have specified a valid U5303A instrument address and click Ok to save the parameters of U5303ACosimBus model, then when you open the GUI next time, the GUI will detect all connected U5303A instruments again and check whether the address that was specified last time is still in current connected instruments list. If so, the GUI will initialize the previously configured U5303A instrument, otherwise, it will switch to the blank option.

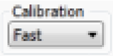
The GUI will extract U5303A instrument hardware options during the instrument initialization. After an U5303A instrument is initialized successfully, you can view all its hardware options by clicking Option button.



| Instrument Options | Description |
|---|---|
| DGT | Basic Digitizer Function |
| LX2 | FPGA Model: Xilinx Virtex 6, XC6VLX195T |
| SR2 | 1.6 GS/s Sampling Rate |
| F10 | 1 GHz bandwidth additional path |
| M40 | 4 GB aquisition memory |
| INT | Interleaved channel Sampling functionality |
| FDK | User FDK Mode (It's a mandatory option for SystemVue M9703 FDK flow) |

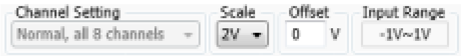
In addition, the GUI will also check whether the selected U5303A instrument has any conflict with the generated FPGA programming files. It will check whether the selected U5303A instrument has FDK option (the U5303A instrument has to have FDK option to use a customized FPGA programming file). If not, the U5303A instrument can't be selected and the U5303A address will be switched to blank option automatically.

Calibration


Calibration: Fast

It has three options: None, Fast and Full.

- None: It will not do U5303A instrument self-calibration at the beginning of simulation.
- Fast: It will do U5303A instrument self-calibration for the current instrument parameters setting at the beginning of simulation. It usually takes several seconds for the fast self-calibration.
- Full: It will do U5303A instrument self-calibration for all instrument parameter setting at the beginning of simulation. It usually takes about one minute for the full self-calibration.



Channel Setting: Normal, all 8 channels

Scale: 2V

Offset: 0 V

Input Range: -1V~1V

### Channel Setting

It has three options: Normal, 2 channels, Interleaved, channel 1 and Interleaved, channel 2.

It's read-only for U5303ACosimBus model. The Channel Setting value is gotten from the specified FPGA image file, so it shows the channel setting that you configured for your own FPGA programming file.

Note: If you configured channel setting as Interleaved mode and generate the FPGA programming file, you have to use an U5303A instrument with INT option in order to use your FPGA programming file. The GUI will check your selected U5303A instrument. If the selected U5303A instrument has no INT option, it will pop-up an error window and ask you to change to another U5303 FPGA image or change to another U5303 instrument.

### Scale

It has two options: 1V and 2V. It's the full voltage range for input signal. This parameter is used to control the ADC input full scale of U5303A instrument.

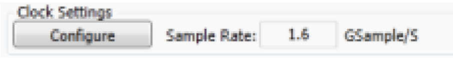
### Offset

This parameter is used to control the offset adjust of U5303A's input amplifier. The valid range is from -2*Scale to 2*Scale. When the input signals ride on a DC value, you can use the offset adjustment to make the input signals be in the center of valid input range of U5303A instrument, which can get the optimal quantization results.

### Input Range

It is a read-only item to show the valid range of input signal based on specified Scale and Offset parameters.

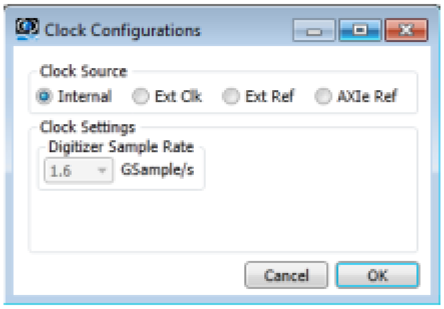
### Clock Settings



Sample Rate: 1.6 GSample/S

In the Clock Settings area, there is a Configure button and a read-only Sample Rate display. You can click Configure button to set clock options. The final sample rate that the U5303A instrument works on is related to U5303A hardware option, channel setting and clock configuration and will be displayed in the GUI.

After clicking Configure button, a new GUI will be opened to configure the clock of U5303A instrument.



Clock Configurations

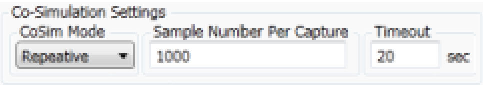
Clock Source

● Internal ○ Ext Clk ○ Ext Ref ○ AXIe Ref

Clock Settings

Digitizer Sample Rate: 1.6 GSample/s

There are three clock options. Refer to Overview of U5303A High-Speed Digitizer for the details of clock options.



Co-Simulation Settings

CoSim Mode: Repeative

Sample Number Per Capture: 1000

Timeout: 20 sec

Cosim Mode

There are two options: Single Pass and Repeative.

Single Pass:

It only asks U5303A digitizer to do data capture once.

When U5303A instrument starts to capture the data, U5303A FPGA will get raw data samples from ADCs and then the raw ADC input samples go through your own FPGA design logics to generate your defined OutPort format output samples. U5303A digitizer will buffer a number of continuous your defined OutPort format output samples in on-board DDR3 RAM without any data loss. The number is specified in next parameter "Sample Number Per Capture".

After the "Sample Number Per Capture" samples of the first data capture are outputted, U5303ACosimBus model will not output valid samples any more, that means that output ports FPGA0_Valid is always 0 after the first data capture.

Repeative:

After the "Sample Number Per Capture" samples of a data capture are outputted, U5303ACosimBus model will do next data capture and output "Sample Number Per Capture" samples for next data capture until the end of simulation.

Sample Number Per Capture

It's the number of valid output samples for each data capture.

If there are multiple OutPorts defined for DPU FPGA in Non-SameSampleRate mode, U5303ACosimBus model will output valid samples for each OutPorts alternatively. The number specified in "Sample Number Per Capture" is the sum of all OutPorts' valid samples for each data capture. You can't specify the valid output samples number for each OutPort, because it depends on the actual valid samples generation rate for each OutPort. For a given buffer size, you can't forecast whether the buffer size is enough to save a specified number of valid samples for an OutPort.

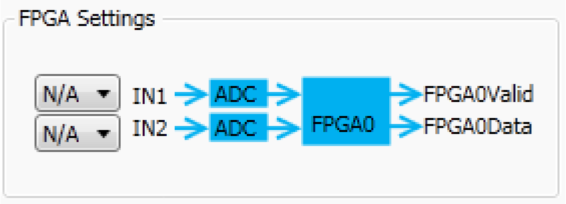
When you use SameSampleRate mode or only define one OutPort, you will only have one ValidOut output, so for each data capture, U5303ACosimBus model will output Sample Number Per Capture valid samples for all defined OutPort.

For example, given the buffer size is fixed and you have two OutPorts, OutA and OutB, and you use Non-SameSampleRate mode. If the actual valid samples generation rate for OutA and OutB is similar, the buffer size is enough to capture NumX valid samples for both OutA and OutB. But if the actual valid samples generation rate of OutA is large and that of OutB is very small, the buffer size may not be enough to capture NumX valid samples for OutB. So you can only specify the sum number of all OutPorts' valid samples, then SystemVue can forecast whether U5303A instrument has enough buffer size for your data capture. If U5303A instrument hasn't enough buffer size for your specified "Sample Number Per Capture", SystemVue will post an error during simulation and tell you the maximum number for "Sample Number PerCapture" according to your current configuration.

TimeOut

It specifies the maximum time limitation for an U5303A instrument to buffer " Sample Number Per Capture" valid output samples to DDR3 SDRAM for each data capture. If U5303A instrument can't save enough valid output samples within this TimeOut time, SystemVue simulation will abort.

It is usually used to avoid SystemVue simulation hang because your FPGA design doesn't generate valid output samples. Sometimes, your FPGA design only generates valid output samples under a specific condition, such as detecting a synchronization signal. If your input analog signals to U5303A instrument don't contain the synchronization signal, your FPGA design will not generate valid output samples and your SystemVue simulation will hang. So you can use this parameter to avoid the simulation hang.



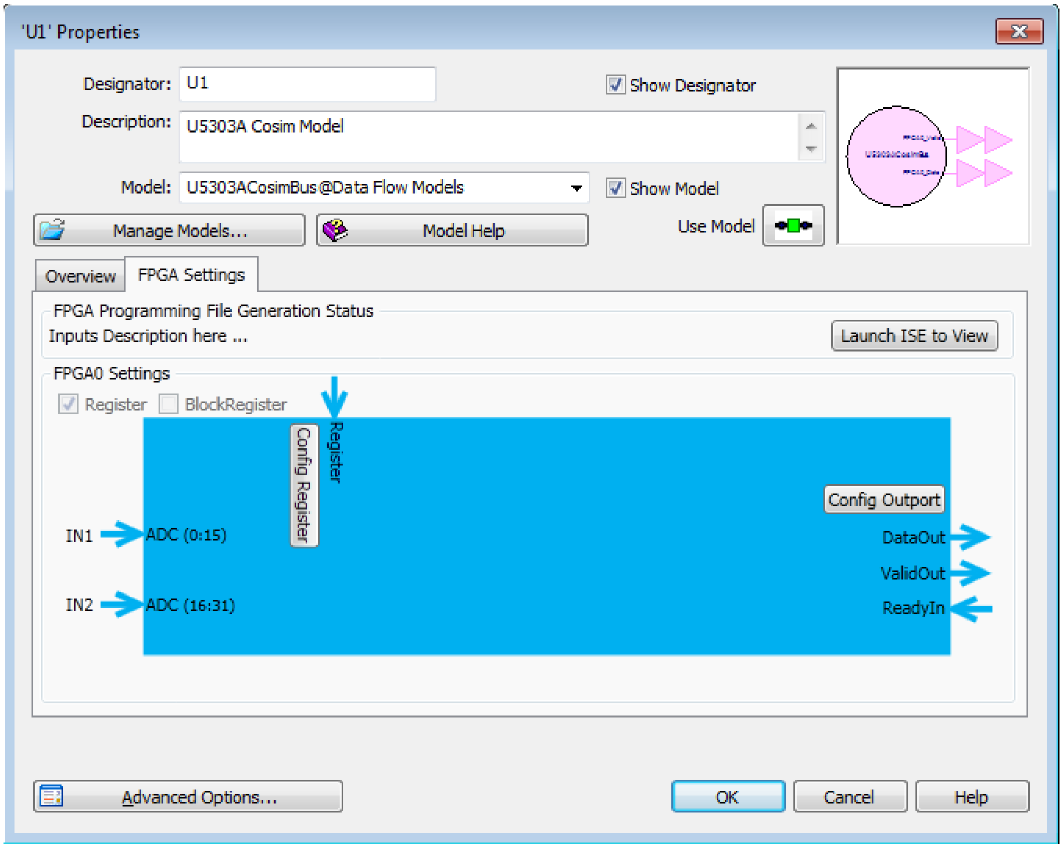
FPGA Settings

FPGA Settings area displays the FPGA architecture according to your current configuration. You can configure the DPU FPGA in FPGA Settings tab.

Bandwidth Limitation

It specifies whether to use or bypass an analog filter before each ADC on U5303A instrument.
If U5303A instrument has F05 option, it's mandatory to use the 600MHz bandwidth analog filter.
If U5303A instrument has F10 option, you can select "N/A" to bypass the analog filter or select "600MHz" to use the filter.

FPGA Settings Tab



'U1' Properties

Designator: U1

☑ Show Designator

Description: U5303A Cosim Model

Model: U5303ACosimBus@Data Flow Models

☑ Show Model

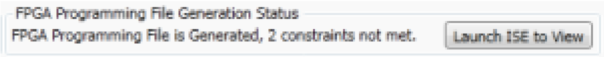
FPGA Programming File Generation Status

Inputs Description here ...

FPGA0 Settings

☑ Register ☐ BlockRegister

FPGA Programming File Generation Status



FPGA Programming File Generation Status

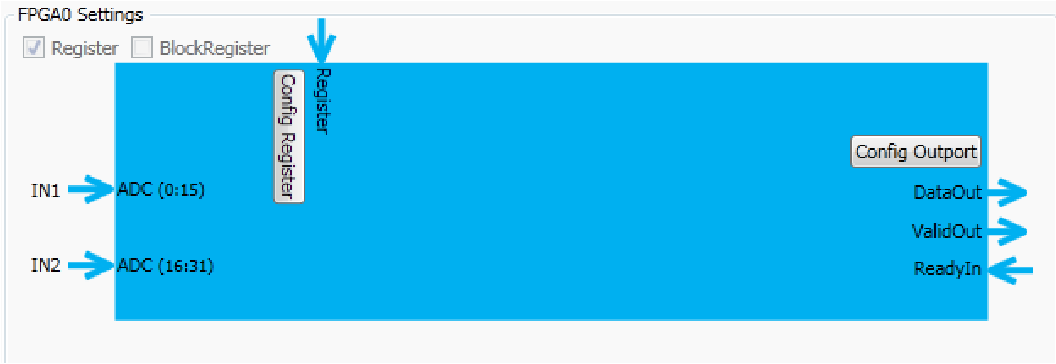
FPGA Programming File is Generated, 2 constraints not met.

This area can help you to know the generation result of FPGA programming file.

The left part of this area will display some read-only information to tell you whether the FPGA programming file is generated successfully. If it's generated, whether there is any timing constraints violation.

The right part of this area is a button Launch ISE to View. You can click this button to open Xilinx ISE GUI and load the ISE project for your specified programming file generation. Then you can view all generation reports in ISE environment. Especially you can view timing report when the read-only information on the left shows some timing constraints violations.

FPGA0 Settings



FPGA0 Settings

☑ Register ☐ BlockRegister

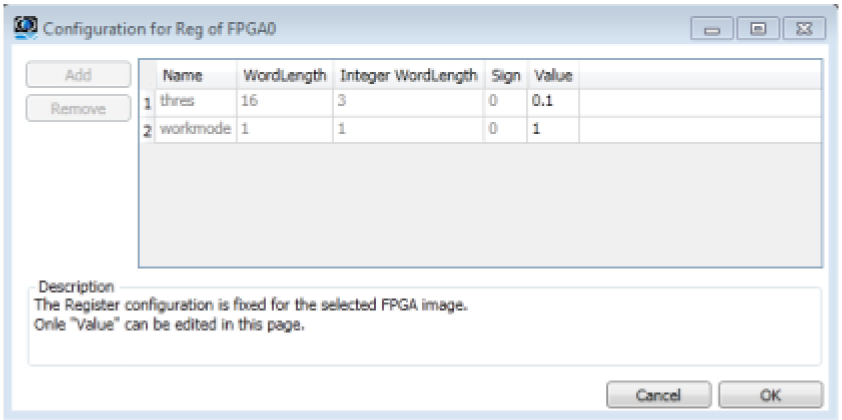
This area displays the DPU FPGA interface. The U5303A Design Template has a similar area in its GUI. You can define the DPU FPGA interface of Register, BlockRegister and OutPort in U5303A Design Template GUI. And in the GUI of U5303ACosimBus model, you can view the generated FPGA interface of Register, BlockRegister and OutPort in the specified FPGA programming file. You can modify the values of Register and BlockRegister in this GUI, but you can't modify the interface format, such as WordLength, because the interface format is fixed for generated FPGA programming file.

Register and BlockRegister checkboxes

The two checkboxes are read-only in this GUI. If Register or BlockRegister interface was defined in your specified FPGA programming file, the corresponding checkbox will be shown checked and the corresponding FPGA interface will be shown in the FPGA interface figure in the GUI.

Config Register Button

If Register interface was defined when generating the specified FPGA programming file, this button will be shown in the FPGA interface figure. After you click this button, a new GUI will be opened.



Configuration for Reg of FPGA0

|   | Name | WordLength | Integer WordLength | Sign | Value |
|---|---|---|---|---|---|
| 1 | thres | 16 | 3 | 0 | 0.1 |
| 2 | workmode | 1 | 1 | 0 | 1 |

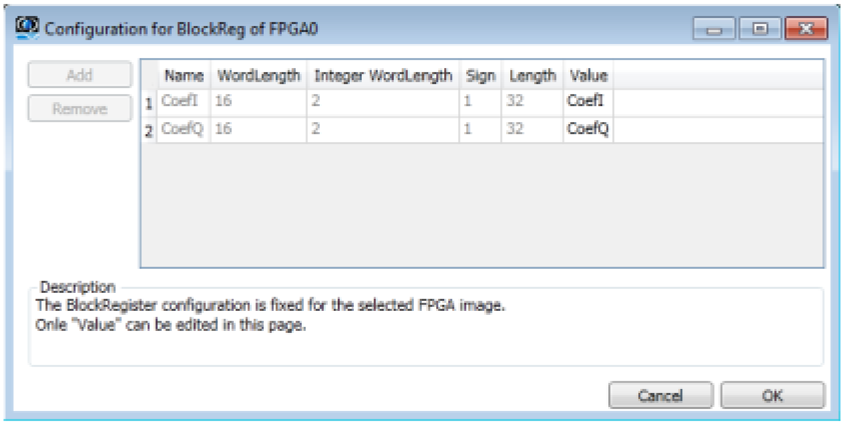
Description
The Register configuration is fixed for the selected FPGA image.
Onle "Value" can be edited in this page.

You can view "Name", "WordLength", "Integer WordLength" and "Sign" of the defined Register in your specified FPGA programming file, but you can't modify them because they are fixed for a generated FPGA programming file. But you can modify "Value" of Register to re-configure the Register values for U5303A instrument co-simulation.

You can use a float-point number as the value of a Register or use a variable that is defined in equation.

Config BlockRegister Button

If BlockRegister interface was defined when generating the specified FPGA programming file, this button will be shown in the FPGA interface figure. After you click this button, a new GUI will be opened.



| | Name | WordLength | Integer WordLength | Sign | Length | Value |
|---|---|---|---|---|---|---|
| 1 | CoefI | 16 | 2 | 1 | 32 | CoefI |
| 2 | CoefQ | 16 | 2 | 1 | 32 | CoefQ |

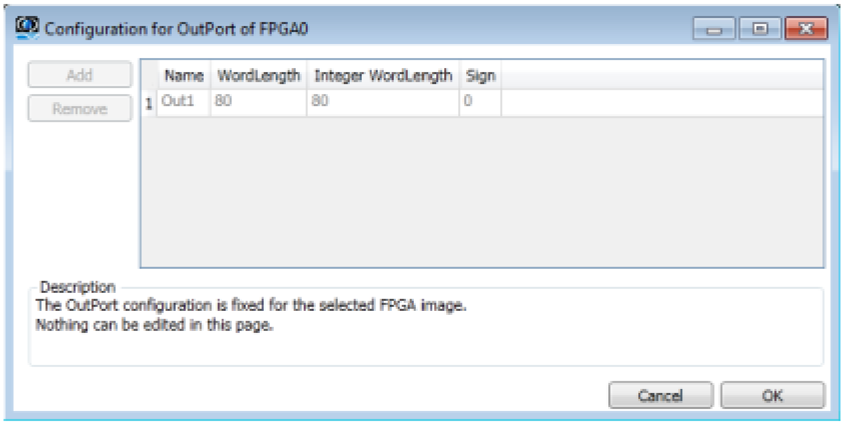
The BlockRegister configuration is fixed for the selected FPGA image.
Onle "Value" can be edited in this page.

You can view "Name", "WordLength", "Integer WordLength", "Sign" and "Length" of the defined BlockRegister in your specified FPGA programming file, but you can't modify them because they are fixed for a generated FPGA programming file. But you can modify "Value" of BlockRegister to re-configure the BlockRegister values for U5303A instrument co-simulation.

You can use a vector with the corresponding "Length" as the value of a BlockRegister or use a variable that is defined in equation.

Config OutPort Button

OutPort is a mandatory FPGA interface, so Config OutPort button and the OutPort interface is always shown in the FPGA interface figure.
After you click this button, a new UI will be opened.



| | Name | WordLength | Integer WordLength | Sign |
|---|---|---|---|---|
| 1 | Out1 | 80 | 80 | 0 |

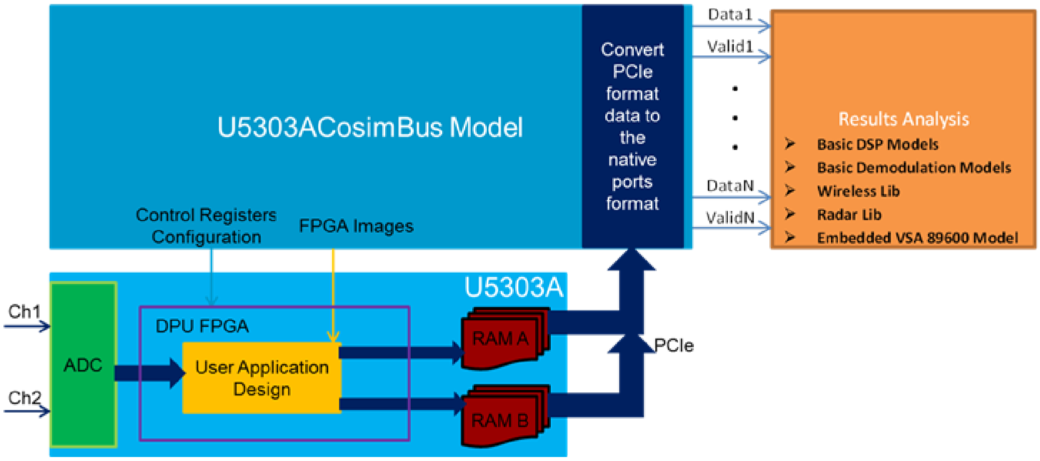
The OutPort configuration is fixed for the selected FPGA image.
Nothing can be edited in this page.

You can view "Name", "WordLength", "Integer WordLength" and "Sign" of the defined OutPort in your specified FPGA programming file, but you can't modify them because they are fixed for a generated FPGA programming file. Nothing can be modified for the defined OutPort interface in a generated FPGA programming file.

### U5303A Co-Simulation Model Simulation Behavior Description

U5303ACosimBus model has the same output ports as U5303A Design Template subnet, but it hasn't any input port. Usually, you can connect the output ports of U5303ACosimBus model to the same bus wires as U5303A Design Template subnet to reuse the data analysis design of software simulation for U5303A instrument co-simulation.

When you have connected U5303ACosimBus model well in a schematic, you can run SystemVue simulation. Then SystemVue will do the following steps to link with real U5303A instrument and capture the results data back SystemVue.



SystemVue On PC

1. SystemVue leverage MD2 driver to download specified FPGA programming files to DPU FPGA, configure the DPU FPGA according to your settings in the GUI of U5303ACosimBus and do self-calibration for the DPU FPGA.
2. If Register or BlockRegister is defined in the FPGA programming file, SystemVue writes initial values for the defined Register or BlockRegister on DPU FPGA.
3. After writing the initial values of Register and BlockRegister for DPU FPGA, DPU FPGA is ready to capture data. Then SystemVue asks U5303A to send a "Start" signal to DPU FPGA. Then DPU FPGA can capture ADC input data.
4. The DPU FPGA process the real-time ADC input data stream as designed in user application design and generate OutPort data streams.
5. The OutPort data streams are packaged and switched and then buffered into U5303A on-board DDR RAM.

6. In the GUI of U5303ACosimBus model, there is a parameter "Samples Number Per Capture". When SystemVue detects that DDR RAM has buffered enough OutPort data for specified "Samples Number Per Capture", SystemVue will stop ADC data capture and read back the buffered OutPorts data via PCI Express connection between U5303A instrument and PC.
7. SystemVue will automatically convert PCI Express data format to user defined OutPort data format and output the OutPort data via the output ports of U5303ACosimBus model. As the OutPort data streams are buffered in DDR RAM based on packages, they will also be outputted based on packages from the output ports of U5303ACosimBus model.
8. After DPU FPGA output "Sample Number Per Capture" samples, if you're using "Single Pass" co-sim mode, SystemVue will not ask U5303A instrument to capture data and ValidOut of DPU FPGA will be 0 until the end of simulation. If you're using "Repeative" co-sim mode, SystemVue will go to step (3) and ask U5303A instrument to capture data again.

For Non-Same Sample Rate mode, the data buffered in DDR3 RAM is based on the OutPort package. One package just includes valid output data from one OutPort, and the package header will have the information of OutPort index. So when SystemVue read data from DDR3 RAM, it is also OutPort package based. The number of samples in one package is not fixed, and it's variable as the word length of OutPort.

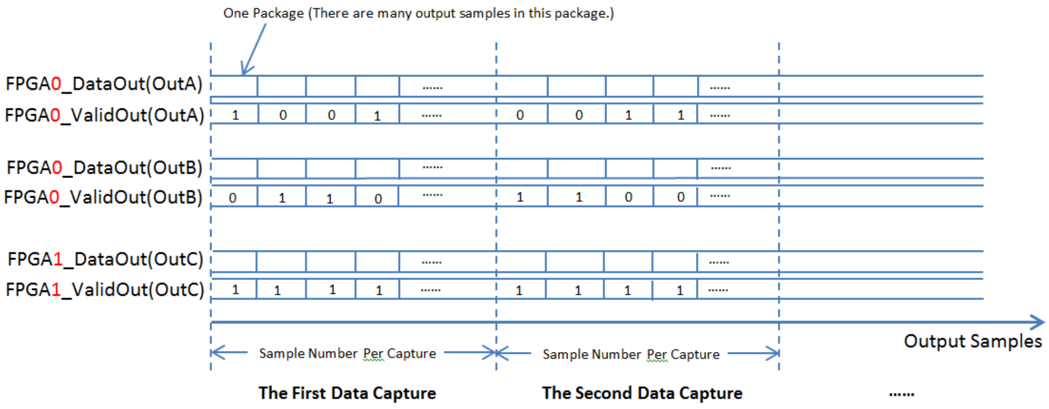
When you just define one OutPort, all packages are for this OutPort and all physical bandwidth between FPGA and DDR3 RAM is used for this OutPort data. So the valid output of the OutPort will be always 1.

When you define multiple OutPorts, U5303ACosimBus model will output valid samples for each OutPorts alternatively. It means that ValidOut of OutPorts will be 1 alternatively. The number of "Sample Number Per Capture" is the sum of all OutPorts' valid samples for each data capture. You can't specify the valid output samples number for each OutPort, because it depends on the actual valid samples generation rate for each OutPort. For a given buffer size, you can't forecast whether the buffer size is enough to save a specified number of valid samples for an OutPort
.

For example, we defined two OutPorts, OutA and OutB for FPGA0 and define one OutPort OutC for FPGA1. When SystemVue read back OutA package, DataOut of OutA will output valid data, and ValidOut of OutA is 1; while ValidOut of OutB is 0 during the time. When SystemVue read back OutB package, DataOut of OutB will output valid data, and ValidOut of OutB is 1; while ValidOut of OutA is 0 during the time. OutA and OutB will output valid data alternatively, and for each output sample, only one OutPort data is valid.
While for FPGA1, ValidOut of OutC will always be 1 and DataOut of OutC will always output valid data.

For both FPGA0 and FPGA1, every data capture generates "Sample Number Per Capture" output samples. The OutPort output samples within one data capture are gotten by processing a span of continuous ADC data stream. While when a new data capture starts, the data stream is not continuous with the previous data capture.

It is shown in below figure:



For Same Sample Rate mode, it's a special case of Non-Same Sample Rate mode, it actually just have one combined output branch. So in this mode, FPGA0_ValidOut is always 1.

## Tutorial of SystemVue U5303A FPGA Design Flow

We will use a simple FIR example to go through the whole U5303A FPGA design flow. You can find the tutorial example from "Help > Open Examples > Hardware Design > U5303_FDK > U5303_FDK_Tutorial".

In Normal, all 8 channels mode, two ADC input channels are connected to a FPGA. In this FIR example, we want to connect IQ signal of a baseband signal to the two channels and ask the IQ signals to go through the FIR filters with the same coefficients. We implemented two set of FIR coefficients, one is for Low Pass Filter and the other is for High Pass Filter. We defined a Register to select which set of coefficients are used.

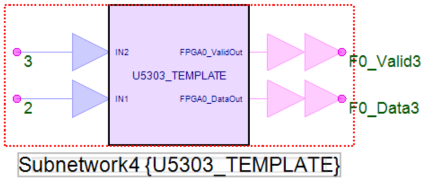
We can use U5303A Design Template subnet to do software simulation. Then we can run the automatic flow to generate FPGA programming file and use the generated FPGA programming file for U5303A instrument co-simulation.

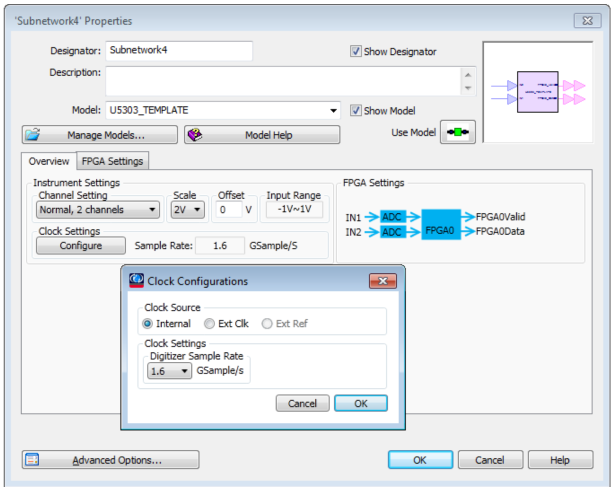
### Design the FIR Example for U5303A FPGA

We need start from a blank U5303A Design Template. So you can open "Help > Open Examples > Hardware Design > U5303_FDK > U5303_Design_Template > U5303_Design_Template.wsv" and save it to a writable directory.

Then you can drag-and-drop a U5303_TEMPLATE subnet to a schematic and double click it to open its GUI.

In U5303A Design Template GUI, you need click "Configure" button in Clock Settings area to open the Clock Configuration GUI and select to use "Internal" clock mode at 1.6 GSample/S sample rate.





'Subnetwork4' Properties

Designator: Subnetwork4 ☑ Show Designator

Description:

Model: U5303_TEMPLATE ☑ Show Model

Overview | FPGA Settings

Instrument Settings

Channel Setting: Normal, 2 channels; Scale: 2V; Offset: 0 V; Input Range: -1V~1V

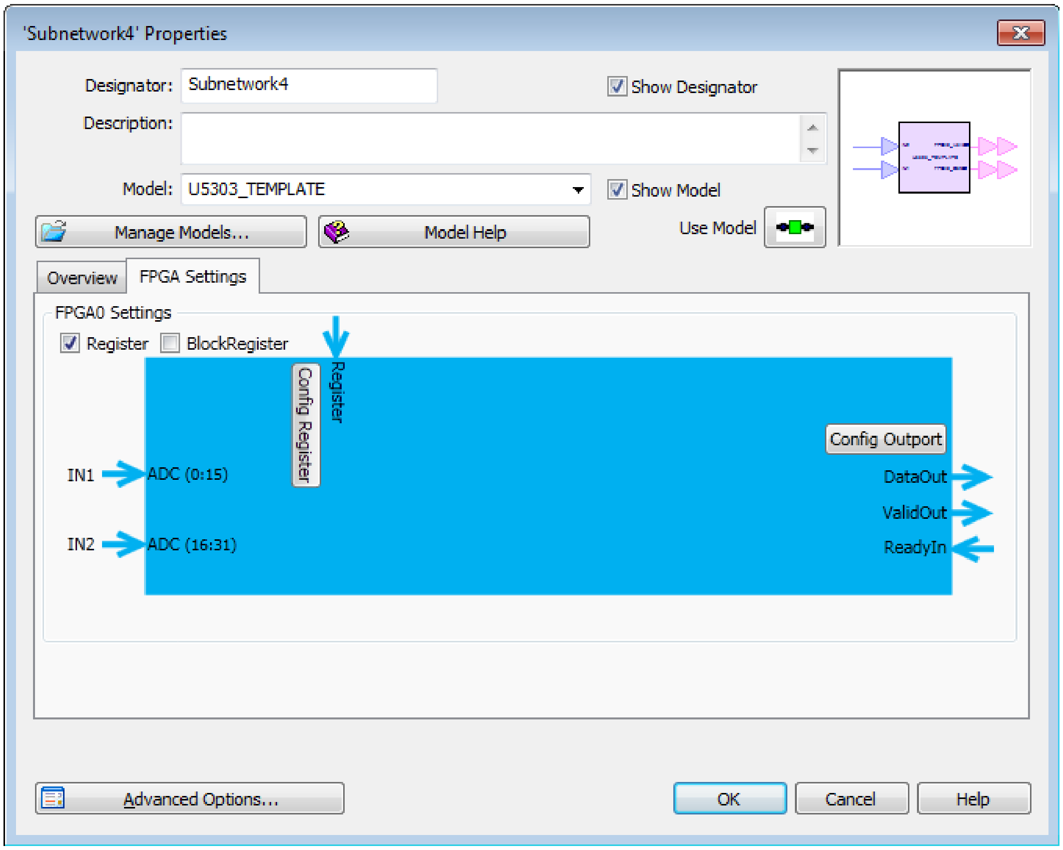
Clock Settings: Sample Rate: 1.6 GSample/S

Clock Configurations

Clock Source: ● Internal ○ Ext Clk ○ Ext Ref

Clock Settings: Digitizer Sample Rate: 1.6 GSample/s

Then you can click FPGA Settings to configure the DPU FPGA interface. As we just use Register in this example, you need uncheck BlockRegister checkbox in this GUI. Then you can click "Config Register" and "Config OutPort" buttons to configure Register and OutPort individually.



'Subnetwork4' Properties

Designator: Subnetwork4 ☑ Show Designator

Description:

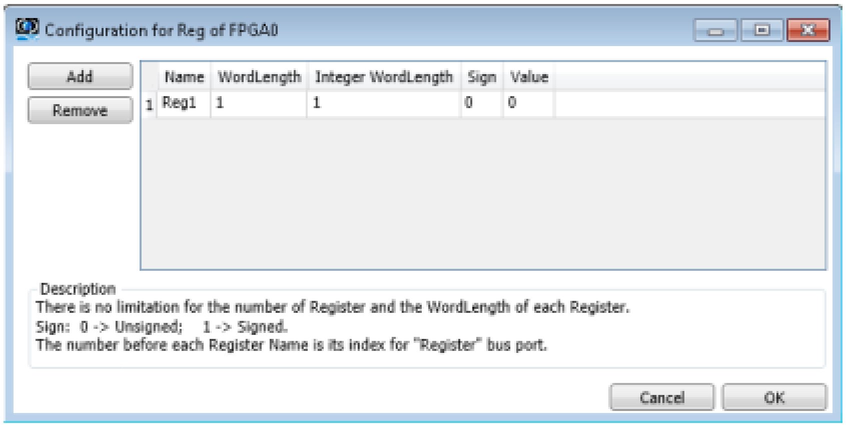
Model: U5303_TEMPLATE ☑ Show Model

Overview | FPGA Settings

FPGA0 Settings

☑ Register □ BlockRegister

For Register configuration, we just need 1 bit to switch two FIR coefficients, so we can configure Register as below. We can give Value as 0 to switch to LPF FIR filter or give Value as 1 to switch to HPF FIR filter for DPU FPGA.

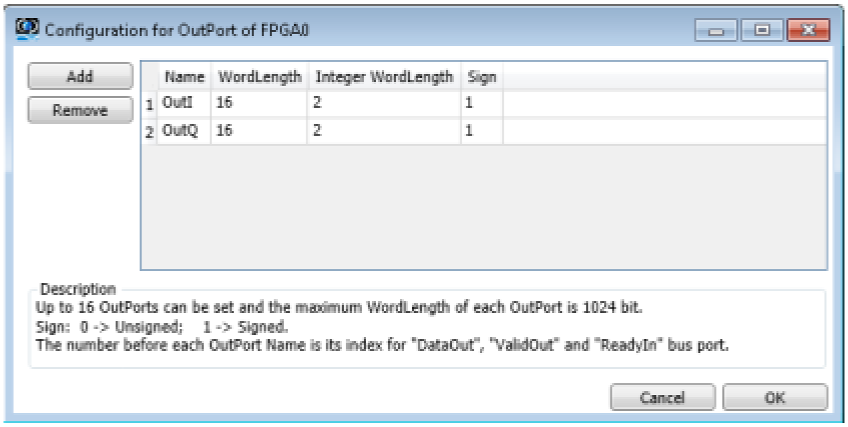


Configuration for Reg of FPGA0

| | Name | WordLength | Integer WordLength | Sign | Value |
|---|---|---|---|---|---|
| 1 | Reg1 | 1 | 1 | 0 | 0 |

Description
There is no limitation for the number of Register and the WordLength of each Register.
Sign: 0 -> Unsigned; 1 -> Signed.
The number before each Register Name is its index for "Register" bus port.

For OutPort configuration, we can define OutI and OutQ for IQ data output from the FIR filters. And the fixed-point data format is defined as Signed, wordlength = 16, integer wordlength = 2.



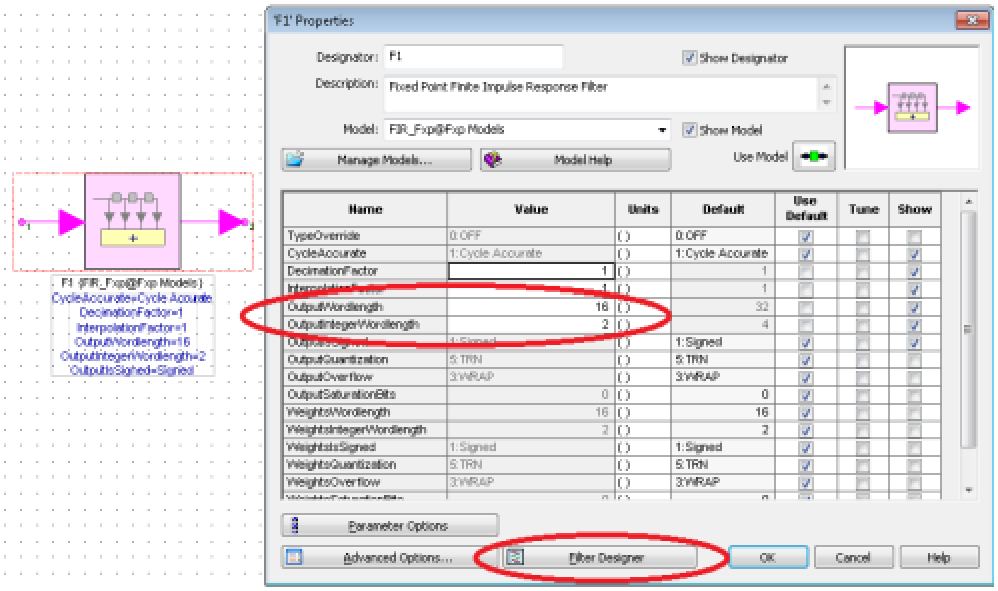
Configuration for OutPort of FPGA0

| | Name | WordLength | Integer WordLength | Sign |
|---|---|---|---|---|
| 1 | OutI | 16 | 2 | 1 |
| 2 | OutQ | 16 | 2 | 1 |

Description
Up to 16 OutPorts can be set and the maximum WordLength of each OutPort is 1024 bit.
Sign: 0 -> Unsigned; 1 -> Signed.
The number before each OutPort Name is its index for "DataOut", "ValidOut" and "ReadyIn" bus port.

You have used the GUI to configure the FPGA interfaces for U5303A DPU FPGA. Then you need create FPGA design for U5303A FPGA. Open U5303_FPGA0 subnet schematic, all interface ports are pre-configured, you need create FIR design in this schematic and connect the FIR design with the FPGA interfaces.

Go to "Hardware Design Library" in Part Selector and drag a FIR_Fxp model to U5303_FPGA0. Double click this model to open its parameters UI. You need modify its OutputWordLength, OutputIntegerWordLength and OutputIsSigned as <16, 2, Signed> to match your OutPort data format definition. Then click "Filter Designer" to open FIR coefficients design UI.



'F1' Properties

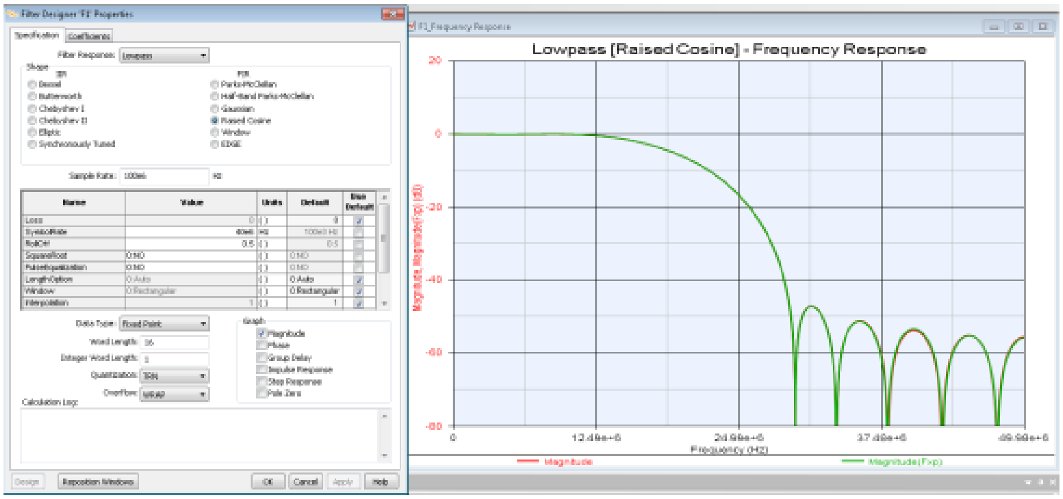
Designator: F1 ☑ Show Designator

Description: Fixed Point Finite Impulse Response Filter

Model: FIR_Fxp@Fxp Models ☑ Show Model

| Name | Value | Units | Default | Use Default | Tune | Show |
|---|---|---|---|---|---|---|
| TypeOverride | 0:OFF | ( ) | 0:OFF | ☑ | □ | □ |
| CycleAccurate | 1:Cycle Accurate | ( ) | 1:Cycle Accurate | ☑ | □ | ☑ |
| DecimationFactor | 1 | ( ) | 1 | □ | □ | ☑ |
| InterpolationFactor | 1 | ( ) | 1 | □ | □ | ☑ |
| OutputWordlength | 16 | ( ) | 32 | □ | □ | ☑ |
| OutputIntegerWordlength | 2 | ( ) | 4 | □ | □ | ☑ |
| OutputIsSigned | 1:Signed | ( ) | 1:Signed | ☑ | □ | ☑ |
| OutputQuantization | 5:TRN | ( ) | 5:TRN | ☑ | □ | □ |
| OutputOverflow | 3:WRAP | ( ) | 3:WRAP | ☑ | □ | □ |
| OutputSaturationBits | 0 | ( ) | 0 | ☑ | □ | □ |
| WeightsWordlength | 16 | ( ) | 16 | ☑ | □ | □ |
| WeightsIntegerWordlength | 2 | ( ) | 2 | ☑ | □ | □ |
| WeightsIsSigned | 1:Signed | ( ) | 1:Signed | ☑ | □ | □ |
| WeightsQuantization | 5:TRN | ( ) | 5:TRN | ☑ | □ | □ |
| WeightsOverflow | 3:WRAP | ( ) | 3:WRAP | ☑ | □ | □ |
| WeightsSaturationBits | 0 | ( ) | 0 | ☑ | □ | □ |

In filer coefficients design UI, configure the parameters as shown below for low pass filter:



Filter Designer 'F1' Properties

Filter Response: Lowpass

Shape

IIR: ○ Bessel ○ Butterworth ○ Chebyshev I ○ Chebyshev II ○ Elliptic ○ Synchronously Tuned

FIR: ○ Parks-McClellan ○ Half-Band Parks-McClellan ○ Gaussian ● Raised Cosine ○ Window ○ EDGE

Sample Rate: 100e6 Hz

| Name | Value | Units | Default | Use Default |
|---|---|---|---|---|
| Loss | 0 | ( ) | 0 | ☑ |
| SymbolRate | 40e6 | Hz | 100e3 Hz | □ |
| RollOff | 0.5 | ( ) | 0.5 | □ |
| SquareRoot | 0:NO | ( ) | 0:NO | □ |
| PulseEqualization | 0:NO | ( ) | 0:NO | □ |
| LengthOption | 0:Auto | ( ) | 0:Auto | ☑ |
| Window | 0:Rectangular | ( ) | 0:Rectangular | ☑ |
| Interpolation | 1 | ( ) | 1 | ☑ |

Data Type: Fixed Point

Word Length: 16

Integer Word Length: 1

Quantization: TRN

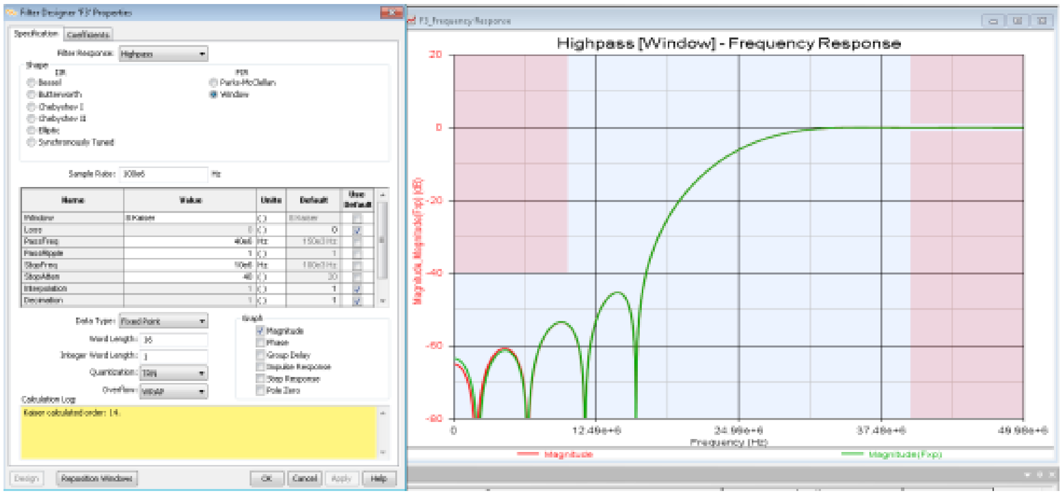
Overflow: WRAP

Graph: ☑ Magnitude □ Phase □ Group Delay □ Impulse Response □ Step Response □ Pole Zero

Calculation Log:

After clicking OK, you can save the coefficients for FIR_Fxp model. You can copy the FIR_Fxp model and paste it to **U5303_FPGA0**, then you have the FIR filters with the same coefficients for I and Q data paths.

In the same way, you can design coefficients for high pass filter as below parameter configuration:



Filter Designer 'F3' Properties

Specification | Coefficients

Filter Response: Highpass

Shape

IIR
○ Bessel
○ Butterworth
○ Chebyshev I
○ Chebyshev II
○ Elliptic
○ Synchronously Tuned

FIR
○ Parks-McClellan
● Window

Sample Rate: 100e6 Hz

| Name | Value | Units | Default | Use Default |
|---|---|---|---|---|
| Window | 8 Kaiser | ( ) | 8 Kaiser | □ |
| Loss | 0 | ( ) | 0 | ☑ |
| PassFreq | 40e6 | Hz | 150e3 Hz | □ |
| PassRipple | 1 | ( ) | 1 | □ |
| StopFreq | 10e6 | Hz | 180e3 Hz | □ |
| StopAtten | 40 | ( ) | 30 | □ |
| Interpolation | 1 | ( ) | 1 | ☑ |
| Decimation | 1 | ( ) | 1 | ☑ |

Data Type: Fixed Point

Word Length: 16

Integer Word Length: 1

Quantization: TRN

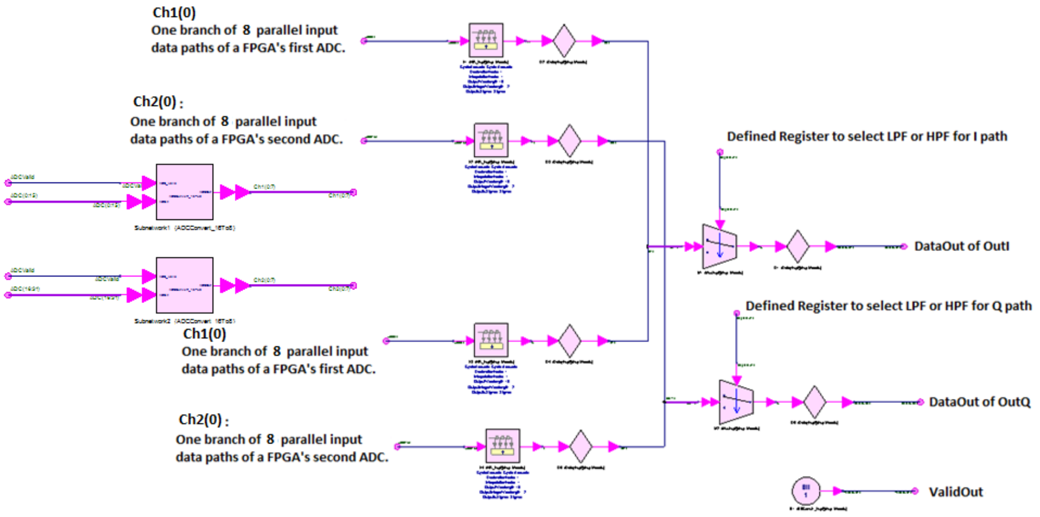
Overflow: WRAP

Graph
☑ Magnitude
□ Phase
□ Group Delay
□ Impulse Response
□ Step Response
□ Pole Zero

Calculation Log:

```
Kaiser calculated order: 14.
```

You can refer to **U5303_FPGA0** in "Help > Open Examples > Hardware Design > U5303_FDK > U5303_FDK_Tutorial > tutorial.wsv" for the whole FPGA design.



Ch1(0)
One branch of 8 parallel input data paths of a FPGA's first ADC.
Ch2(0) :
One branch of 8 parallel input data paths of a FPGA's second ADC.
Defined Register to select LPF or HPF for I path
DataOut of OutI
Defined Register to select LPF or HPF for Q path
Ch1(0)
One branch of 8 parallel input data paths of a FPGA's first ADC.
Ch2(0) :
One branch of 8 parallel input data paths of a FPGA's second ADC.
DataOut of OutQ
ValidOut

Please note:

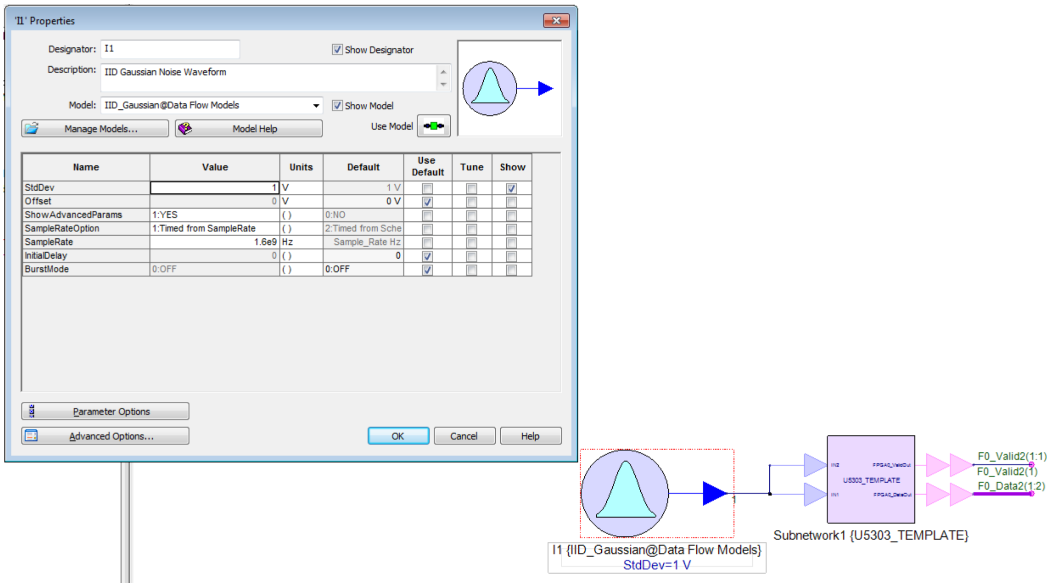
1. Subnet ADCConvert_16To8 is used to convert 16 parallel input streams to 8 parallel streams and make all 8 parallel data valid at every clock cycle. This design just process one of 8 parallel data, it's a 8 times decimation.
2. Because the wordlength of both OutA and OutB are 16 bits, the total bit width of all OutPorts is 32 bits, which is less than 124 bits (the effect throughput to DDR3 SDRAM). So ReadyIn of OutA and OutB will always be 1 and we can simply the design without connecting ReadyIn input ports.

Then you have finished the FPGA design in U5303A Design Template.

## Software Simulation

After FPGA design entry in U5303A Design Entry, we can create a testbench in SystemVue top level schematic to test the FIR design by software simulation.

We need find IID_Gaussian model in "Algorithm Design" library in part selector and drag it to top level schematic. Then we can set its parameters and connect its output to IN1 and IN2 of U5303A Design Template subnet to get a Gaussian white noise test signal for our FPGA design. Please note that you have to set the SampleRate of the IID_Gaussian model as 1.6GHz to match your configuration in the GUI of U5303A Design Template.



'I1' Properties

Designator: I1 ☑ Show Designator

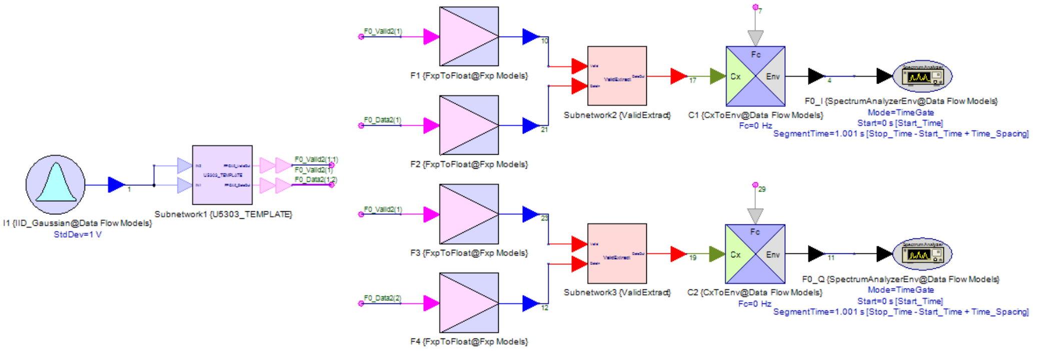
Description: IID Gaussian Noise Waveform

Model: IID_Gaussian@Data Flow Models ☑ Show Model

Use Model

| Name | Value | Units | Default | Use Default | Tune | Show |
|---|---|---|---|---|---|---|
| StdDev | 1 | V | 1 V | ☐ | ☐ | ☑ |
| Offset | 0 | V | 0 V | ☑ | ☐ | ☐ |
| ShowAdvancedParams | 1:YES | ( ) | 0:NO | ☐ | ☐ | ☐ |
| SampleRateOption | 1:Timed from SampleRate | ( ) | 2:Timed from Sche | ☐ | ☐ | ☐ |
| SampleRate | 1.6e9 | Hz | Sample_Rate Hz | ☐ | ☐ | ☐ |
| InitialDelay | 0 | ( ) | 0 | ☑ | ☐ | ☐ |
| BurstMode | 0:OFF | ( ) | 0:OFF | ☑ | ☐ | ☐ |

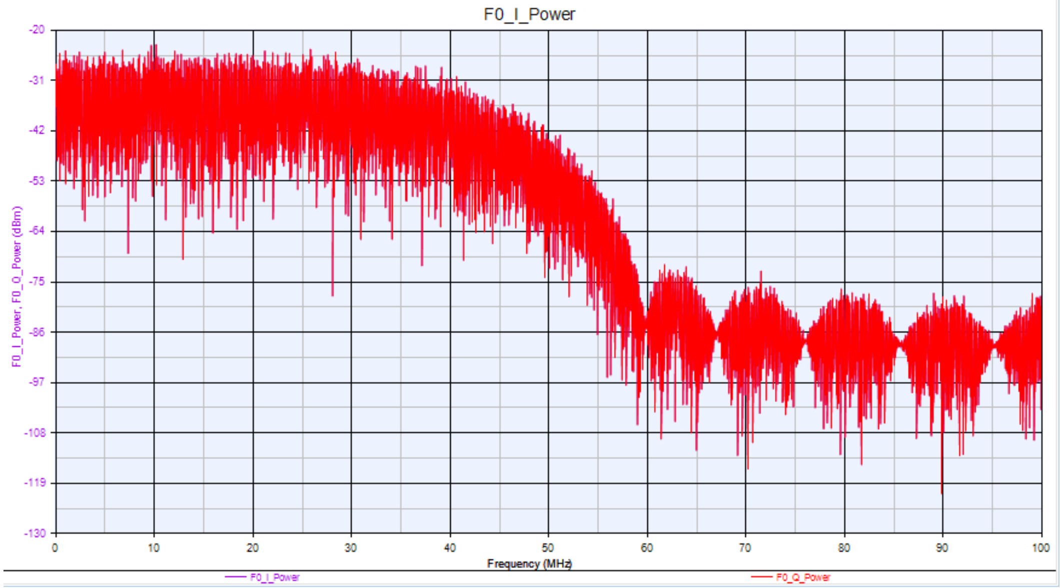
Then you need refer to the top level schematic of "Help > Open Examples > Hardware Design > U5303_FDK > U5303_FDK_Tutorial > tutorial.wsv" to connect models like below to get the spectrum of IQ signals from U5303A DPU FPGA.



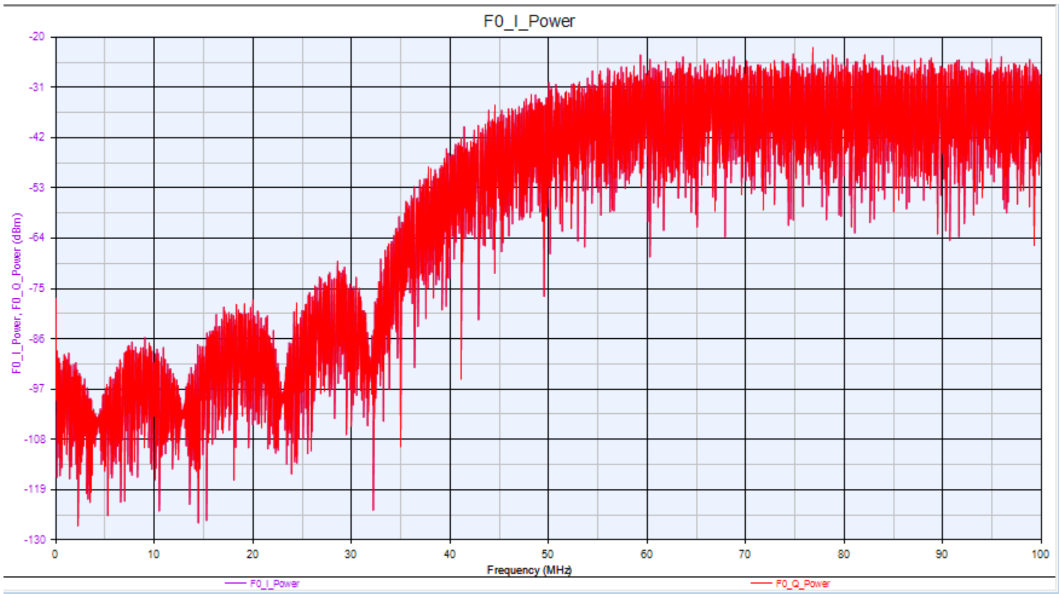
I1 {IID_Gaussian@Data Flow Models}
StdDev=1 V
Subnetwork1 {U5303_TEMPLATE}
F0_Valid2(1:1)
F0_Valid2(1)
F0_Data2(1:2)
F0_Valid2(1)
F1 {FxpToFloat@Fxp Models}
F0_Data2(1)
F2 {FxpToFloat@Fxp Models}
F0_Valid2(1)
F3 {FxpToFloat@Fxp Models}
F0_Data2(2)
F4 {FxpToFloat@Fxp Models}
10
21
23
12
Subnetwork2 {ValidExtract}
Subnetwork3 {ValidExtract}
17
19
7
29
Fc
Cx
Env
C1 {CxToEnv@Data Flow Models}
Fc=0 Hz
C2 {CxToEnv@Data Flow Models}
Fc=0 Hz
4
11
F0_I {SpectrumAnalyzerEnv@Data Flow Models}
Mode=TimeGate
Start=0 s [Start_Time]
SegmentTime=1.001 s [Stop_Time - Start_Time + Time_Spacing]
F0_Q {SpectrumAnalyzerEnv@Data Flow Models}
Mode=TimeGate
Start=0 s [Start_Time]
SegmentTime=1.001 s [Stop_Time - Start_Time + Time_Spacing]

When you connect the top level testbench well, you can run SystemVue simulation. Then you can draw the spectrums of I and Q filtered signals from DPU FPGA in figure

"Design1 Analysis_Design1_F0_I_Power". If you set **Register Value** as 0 to select Low Pass Filter, you can see the results as below figure.

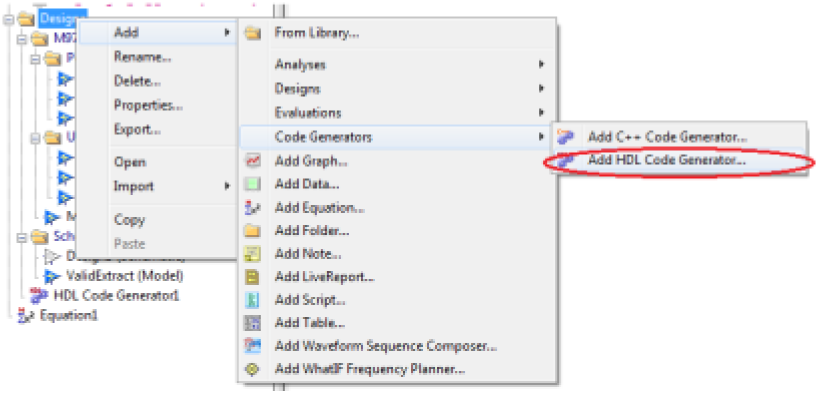


Or if you set Register Value as 1 to select High Pass Filter, you can see the results as below figure.

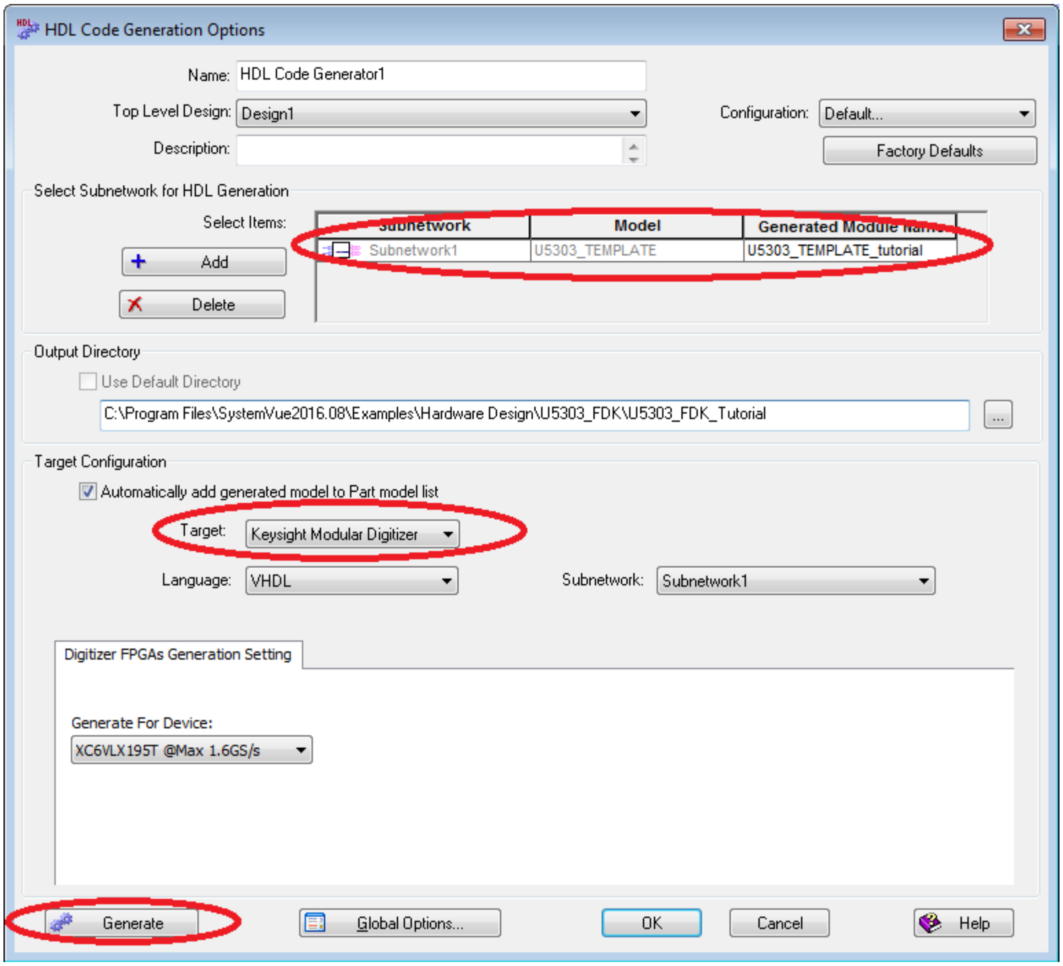


## Generate FPGA Programming File

After you finish software simulation to verify the function of your FPGA design, you can add a "HDL Code Generator" to launch the automatic FPGA programming file generation flow.
You can right click a folder in workspace tree and select to Add HDL Code Generator.



In added HDL Code Generator, you can click Add button to select your **U5303_TEMPLATE** subnet for FPGA image generation. Then select to use "**Keysight Modular Digitizer**" mode and click "**Generate**" button. SystemVue will launch an automatic flow to generate the final FPGA programming file.


HDL Code Generation Options

Name: HDL Code Generator1

Top Level Design: Design1

Configuration: Default...

Description:

Select Subnetwork for HDL Generation

Select Items:

| Subnetwork | Model | Generated Module Name |
|---|---|---|
| Subnetwork1 | U5303_TEMPLATE | U5303_TEMPLATE_tutorial |

Output Directory

☐ Use Default Directory

C:\Program Files\SystemVue2016.08\Examples\Hardware Design\U5303_FDK\U5303_FDK_Tutorial

Target Configuration

☑ Automatically add generated model to Part model list

Target: Keysight Modular Digitizer

Language: VHDL

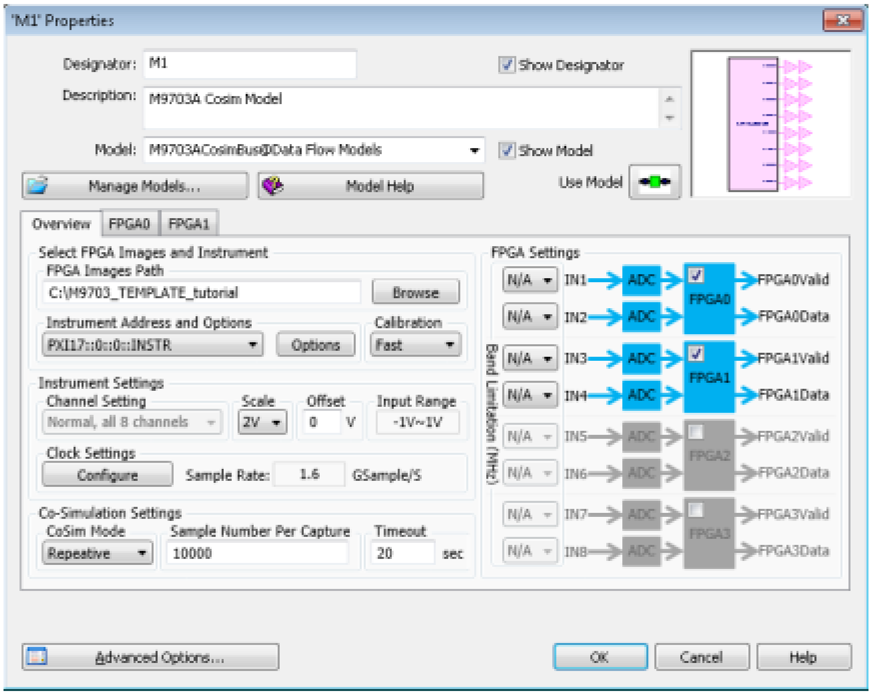
Subnetwork: Subnetwork1

Digitizer FPGAs Generation Setting

Generate For Device:

XC6VLX195T @Max 1.6GS/s

After your FPGA programming file is generated, you can find U5303ACosimBus model in "Hardware Design" library and drag it to the top level testbench schematic. Double click the model to open its GUI and configure as below:


'M1' Properties

Designator: M1 ☑ Show Designator

Description: M9703A Cosim Model

Model: M9703ACosimBus@Data Flow Models ☑ Show Model

Use Model

Overview | FPGA0 | FPGA1

Select FPGA Images and Instrument

FPGA Images Path: C:\M9703_TEMPLATE_tutorial

Instrument Address and Options: PXI17::0::0::INSTR

Calibration: Fast

Instrument Settings

Channel Setting: Normal, all 8 channels

Scale: 2V

Offset: 0 V

Input Range: -1V~1V

Clock Settings

Sample Rate: 1.6 GSample/S

Co-Simulation Settings

CoSim Mode: Repeative

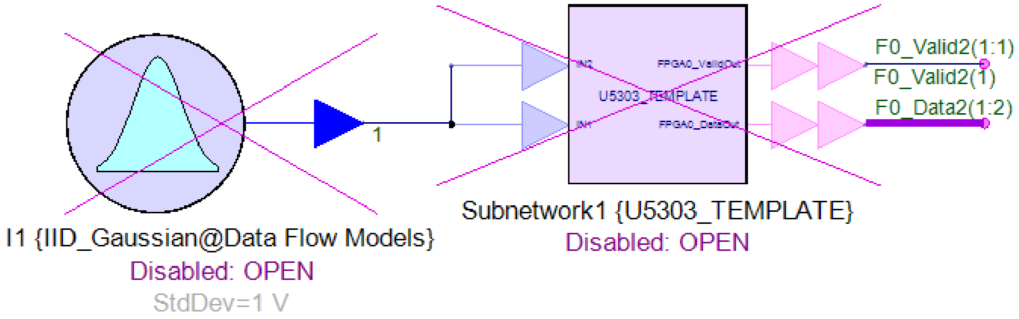
Sample Number Per Capture: 10000

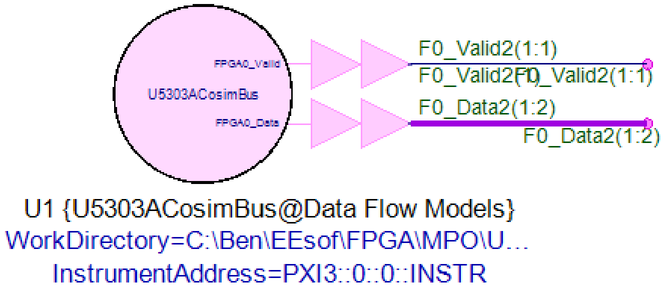
Timeout: 20 sec

FPGA Settings

Select the path of your generated FPGA image in parameter "FPGA Image Path".

You have to install Agilent IO Library and MD2 driver and have U5303A instrument that powers on and connects with your computer. Otherwise, you can't see any instrument listed in this GUI for selection.
Then in the top level testbench, you can connect the U5303ACosimBus model with the same wires as U5303A Design Template and disable Gaussian noise model and U5303A Design Template subnet, as shown below:

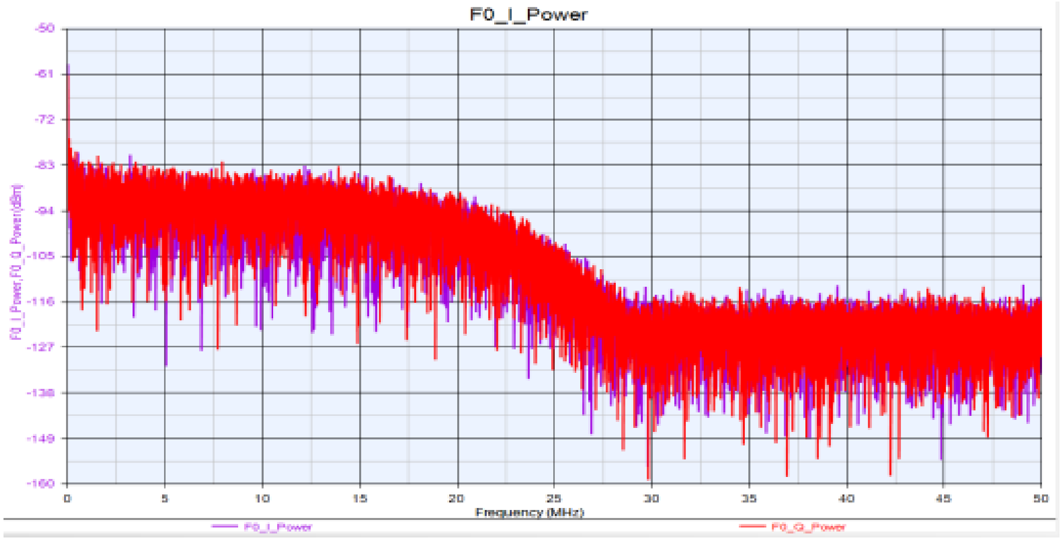



U5303ACosimBus
FPGA0_Valid
FPGA0_Data
F0_Valid2(1:1)
F0_Valid2(1)
F0_Valid2(1:1)
F0_Data2(1:2)
F0_Data2(1:2)
U1 {U5303ACosimBus@Data Flow Models}
WorkDirectory=C:\Ben\EEsof\FPGA\MPO\U…
InstrumentAddress=PXI3::0::0::INSTR

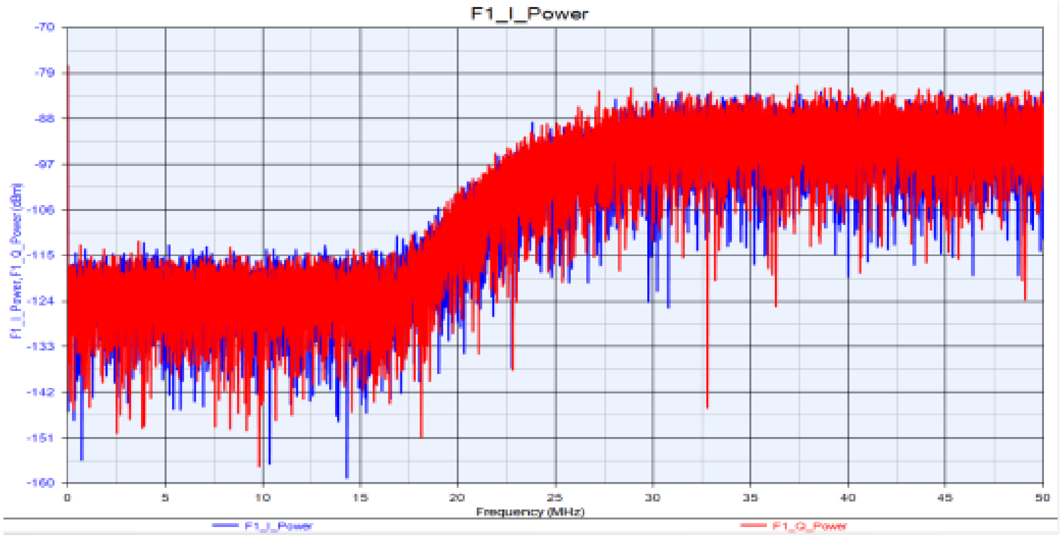
When you connect the **U5303ACosimBus** model well, you can run the SystemVue simulation. **U5303ACosimBus** model will find the actual U5303A instrument and do the co-simulation. You can leave IN1 and IN2 of U5303A instrument open, then it will get the filtering results for the floor noise.

When the co-simulation is finished, you can observe the spectrum in the same figures.

Setting Register Value as 0:



Setting Register Value as 1:



# Algorithm Design

## Algorithm Design

### Contents

## Getting Started with Data Flow

A workspace file in SystemVue is the basic database used to store anything related to a user design, data, graphs, analysis, equations, and so on. A Data Flow simulation in SystemVue requires at least two basic components in a workspace.

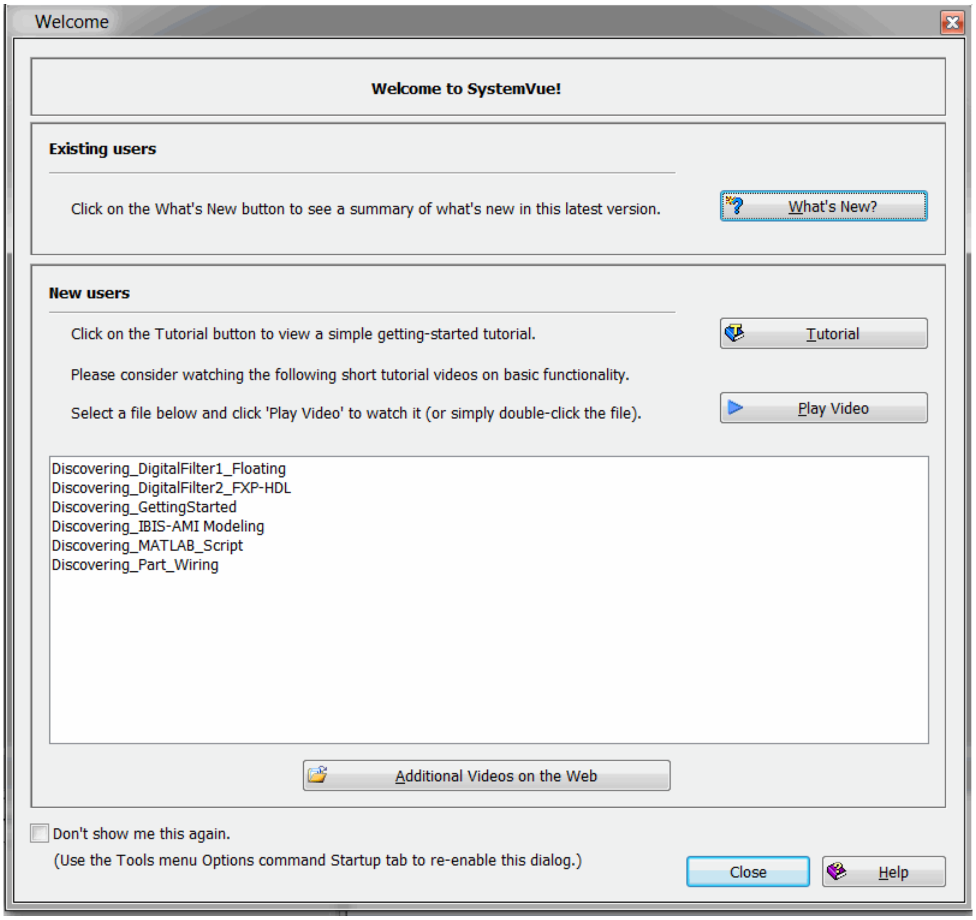
1. A Design: This is used to define how the data flow parts connect together to form a complete System. The Schematic is the graphical view of the design. You can also view the Design as a list of parts (the PartList tab).
2. A Data Flow Analysis: This is the simulation controller that determines sample rate and start time. For more details about data flow technology, see Introduction to Data Flow Simulation.

In this tutorial, you will create a simple design using a sine generator and a sink, run the simulation, and view the output in a graph.

To build a simple simulation, let us start with the Blank workspace containing a Blank Design and a Data Flow Analysis.

### Phase 1- Start SystemVue with a Blank Template

- Start SystemVue. If you encounter any problem in starting SystemVue, then see Installation documentation to make sure the SystemVue has been installed properly.
- If you see a welcome dialog (as shown below), you can also see the Tutorial videos. For now, click on the Close button to proceed with this tutorial.


Welcome

**Welcome to SystemVue!**

**Existing users**

Click on the What's New button to see a summary of what's new in this latest version.

**New users**

Click on the Tutorial button to view a simple getting-started tutorial.

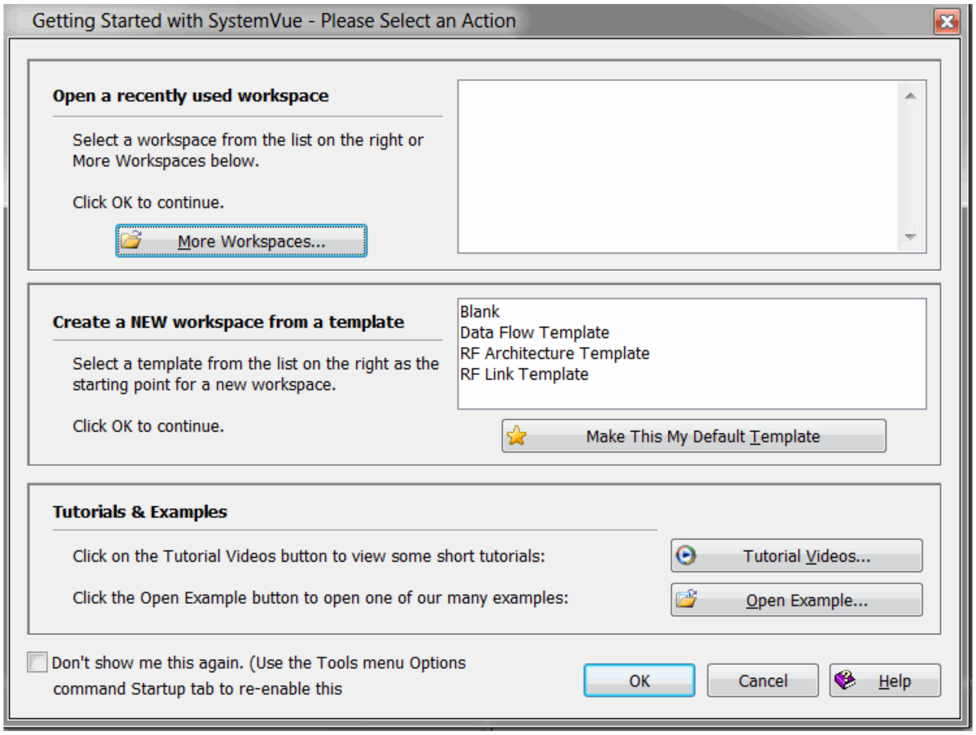
Please consider watching the following short tutorial videos on basic functionality.

Select a file below and click 'Play Video' to watch it (or simply double-click the file).

Discovering_DigitalFilter1_Floating
Discovering_DigitalFilter2_FXP-HDL
Discovering_GettingStarted
Discovering_IBIS-AMI Modeling
Discovering_MATLAB_Script
Discovering_Part_Wiring

□ Don't show me this again.
(Use the Tools menu Options command Startup tab to re-enable this dialog.)

- A Getting Started with SystemVue dialog appears. Select a Blank template in this dialog box (as shown below) and click OK. Optionally, you could open other templates, watch tutorial videos, open examples shipped with SystemVue, or open a recently used workspace using this dialog box. If you do not see this dialog, then you can enable it in Startup of Tools > Options and then select Display the Start Page.


Getting Started with SystemVue - Please Select an Action
More Workspaces...
Make This My Default Template
Tutorial Videos...
Open Example...
OK
Cancel
Help

**Open a recently used workspace**

Select a workspace from the list on the right or More Workspaces below.

Click OK to continue.

**Create a NEW workspace from a template**

Select a template from the list on the right as the starting point for a new workspace.

Click OK to continue.

Blank
Data Flow Template
RF Architecture Template
RF Link Template

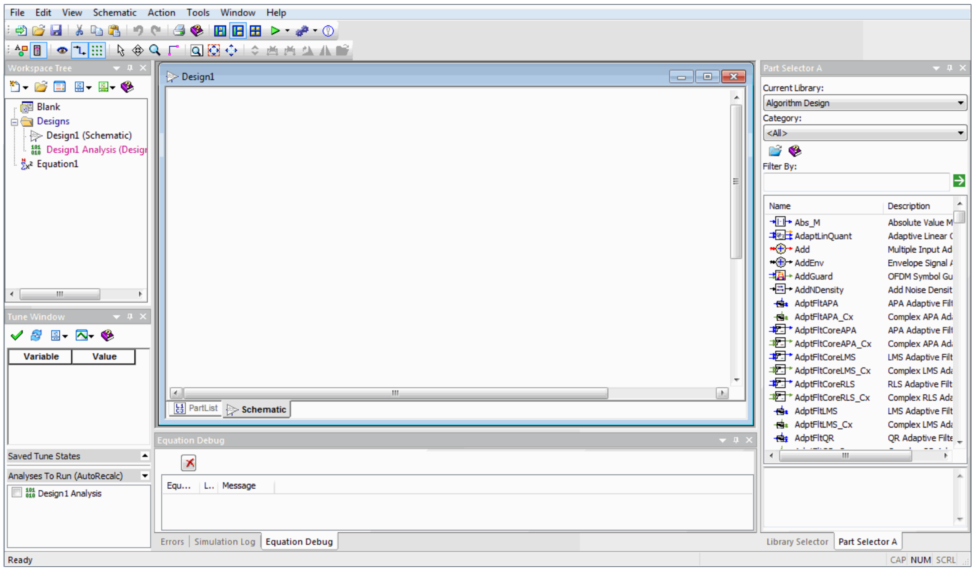
**Tutorials & Examples**

Click on the Tutorial Videos button to view some short tutorials:

Click the Open Example button to open one of our many examples:

□ Don't show me this again. (Use the Tools menu Options command Startup tab to re-enable this

- The SystemVue opens the Blank template (as shown below). The Blank template includes a schematic Design with name Design1 (Schematic), a Data Flow Analysis with name Design1 Analysis, and an Equation with name Equation1. Although we will not be using Equations in this tutorial, it is useful to know that Equations are a powerful tool that enables post-processing of data, control over inputs to simulations, and definition of user-defined custom models. For more details on how to use equations, see Equations documentation.



Current Library: Algorithm Design

Category: <All>

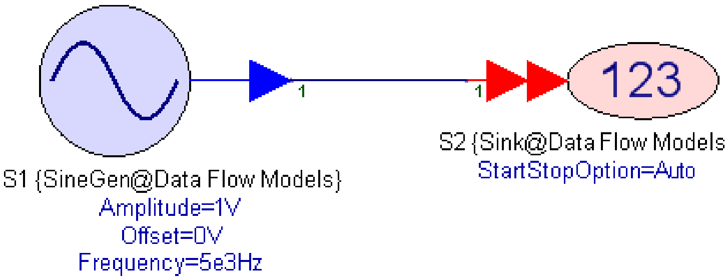
Filter By:

| Name | Description |
|---|---|
| Abs_M | Absolute Value M |
| AdaptLinQuant | Adaptive Linear ( |
| Add | Multiple Input Ad |
| AddEnv | Envelope Signal A |
| AddGuard | OFDM Symbol Gu |
| AddNDensity | Add Noise Densit |
| AdptFltAPA | APA Adaptive Filt |
| AdptFltAPA_Cx | Complex APA Ada |
| AdptFltCoreAPA | APA Adaptive Filt |
| AdptFltCoreAPA_Cx | Complex APA Ada |
| AdptFltCoreLMS | LMS Adaptive Filt |
| AdptFltCoreLMS_Cx | Complex LMS Ada |
| AdptFltCoreRLS | RLS Adaptive Filt |
| AdptFltCoreRLS_Cx | Complex RLS Ada |
| AdptFltLMS | LMS Adaptive Filt |
| AdptFltLMS_Cx | Complex LMS Ada |
| AdptFltQR | QR Adaptive Filte |

## Phase 2- Create the System Design

- To add a Sine Generator to the design, follow these steps:

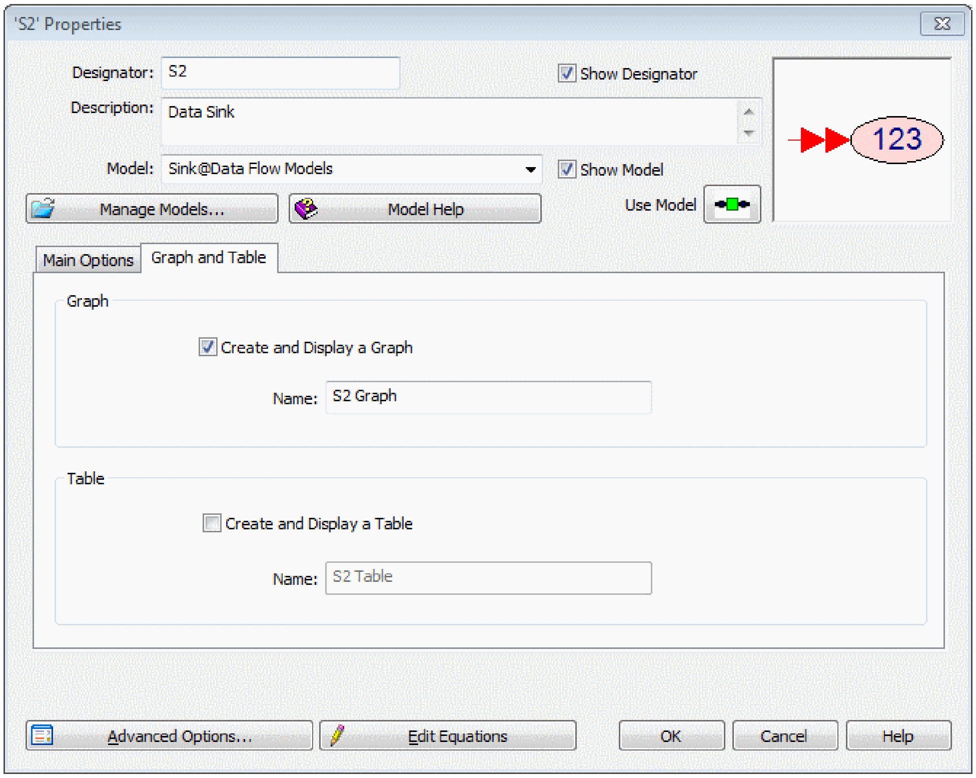
    - Click inside the schematic. The schematic window appears and the part selector may display (depending on its last state).
    - If part selector is not displayed (usually, it is docked on the right of the screen), click the Show Part Selector button in the the Schematic Toolbar.
    - Under *Current Library:*, switch to the Algorithm Design library. You may have many libraries available.
    - There are a lot of parts in the Algorithm Design catalog, so type sine into the *Filter By:* field and press Enter or click the green arrow.
    - Now, click the SineGen part, and then click anywhere in the schematic to place the Sine Generator.
- To add a Data Sink to the schematic, follow these steps.
    - Change the sine to sink in the Filter By: field and press Enter.
    - Click the Sink part in the selector, and then click in the schematic to place it. If you click directly on the SineGen output pin, the two parts connect.
    - If you did not connect them automatically, connect the Generator to the Sink. See Connecting Parts.
        - Connect using a line (connector).
            - Mouse over the SineGen output pin. The cursor changes to a line-connector cursor.
            - Click and drag the line to the Sink pin.
            - Release the mouse to connect the two parts.
        - Or, connect by dragging the sink. The connecting node turns green and stick to the SineGen pin as you drag the part.



NOTE

Note that you do not need to use the **Part Selector** for the most frequently used parts. There are keyboard shortcuts to place those parts quickly. For instance, to place the **SineGen** part, you could just press **Shift-S** and then click on the schematic; or to place the Sink, you could press **S** and click on the schematic. For more information, see Appendix A Keystroke Commands.

- To plot the results in a Graph automatically after simulation, double click the Sink to open its properties. Select the Graph and Table tab (as shown below) and check Create and Display a Graph; this plots the data collected by sink on a graph using the name defined on this tab (in the figure below, it is S2 Graph).



'S2' Properties

Designator: S2 ☑ Show Designator

Description: Data Sink

Model: Sink@Data Flow Models ☑ Show Model

Graph and Table

Graph

☑ Create and Display a Graph

Name: S2 Graph

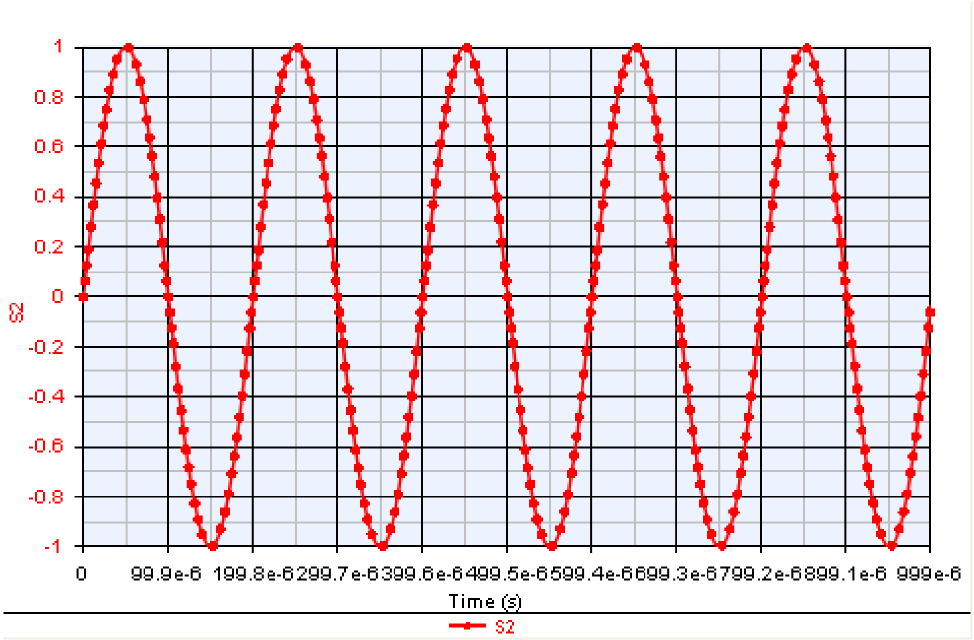
Table

□ Create and Display a Table

Name: S2 Table

## Phase 3- Run the Simulation

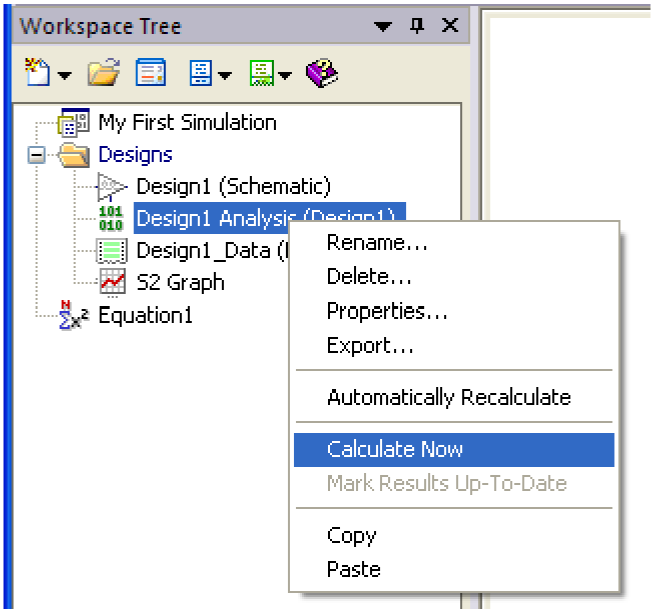
- To run a simulation, click on the Run Analysis button ▷ in the the Schematic Toolbar. This runs the simulation, stores the data collected by the Sink in a Dataset named Design1 Analysis_Design1_Data, and creates a graph named S2 Graph as set in the Sink properties. For further details about dataset, see Datasets documentation. After running the simulation, the S2 Graph is displayed automatically. If you close the window or it gets covered up, you can double click on S2 Graph in the workspace tree to open it again, just like any other workspace tree item.



- Save the workspace by using File -> Save or File -> Save As... and name it My First Simulation.

- Other than using the Run Analysis button ▶ to run a simulation, you can also use one of the following methods to run the simulation:

  - Right-click the analysis and select Calculate Now from the menu.



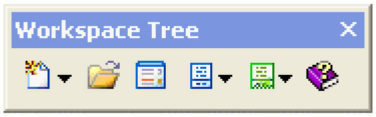
  - Press F5 (this only updates out of date items).

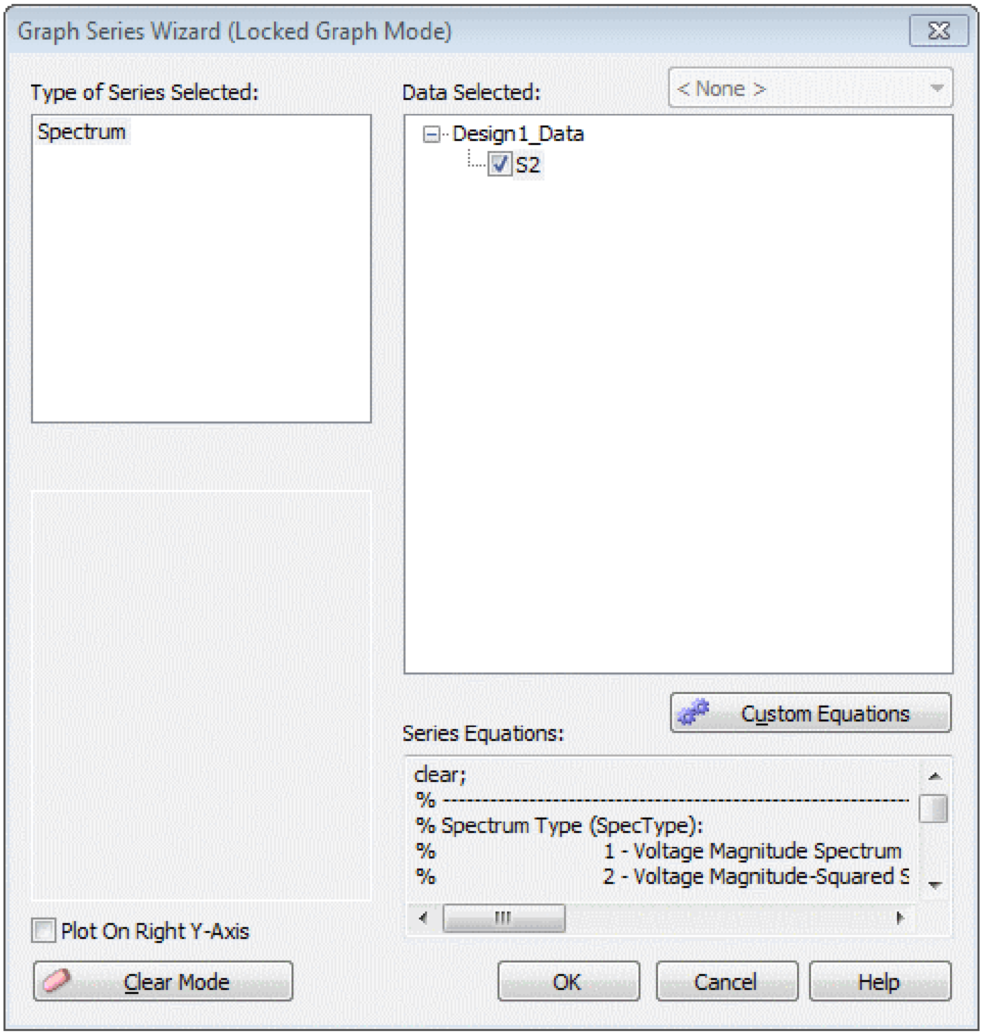
  - Double-click the analysis (open to change it) and when done, click the Calculate Now button in the Analysis dialog.

## Phase 4- Creating Additional Graphs

Once the simulator runs using the settings from the Analysis, it creates a dataset. This is a "bunch" of data variables aggregated into a single container. All of the data variables from the simulation are stored here. You can create Tables and Graphs using this data, post-process it, and compare data from multiple datasets /runs.

To create a graph, follow these steps.

1. Click the New Item button ( ) on the Workspace Tree toolbar ( ).
2. Select Add Graph..., and the Graph Series Wizard window appears.
3. Select the series plot type. For instance, select Spectrum to see the spectrum of your signal.
4. Select the variable that you want plotted (S2 in this example). Some plot types require more than one variable.
5. Click the OK button and the Graph Properties window appears.
6. If desired, change the graph Name, and add a title to the Graph Heading.
7. Click OK.



Type of Series Selected: Spectrum

Data Selected: < None >

Design1_Data
☑ S2

Series Equations:

```
clear;
% ---------------------------------------------------------------
% Spectrum Type (SpecType):
%                    1 - Voltage Magnitude Spectrum
%                    2 - Voltage Magnitude-Squared S
```

□ Plot On Right Y-Axis

For more details about datasets, see Datasets. To learn about creating tables from data in dataset, see Creating Tables.

# Working with MATLAB Script

## Working with MATLAB Script

MATLAB Script is an interpreted programming language that allows you to easily develop algorithms. It is MATLAB compatible but does not support all the programming features of MATLAB. Also, it does not provide all the functions that MATLAB provides. For more details on MATLAB Script, see Using MATLAB Script.

This section covers a series of tutorials that demonstrate how to use MATLAB Script to create simulation models and post process simulation data. It also describes techniques that improve simulation performance of MATLAB Script code.

MATLAB Script also enables you to use your full version of MATLAB for access functionality that is in your toolboxes. You can use your MATLAB installation from a workspace equation, a design equation, a MATLAB Script model or the command prompt by switching to “Use MATLAB” in the right click menu.

### Using MATLAB Script to Create Simulation Models

For all the tutorial examples mentioned here, use the workspace <SystemVue Installation Directory>\Examples\Tutorials\Algorithm_Design\MATLAB_Script\MATLAB_Script_M wsv.

1. Simple Unirate Model
2. Model with State
3. Multirate Model
4. Multirate Model with Bus IO
5. Model with Array IO

### Using MATLAB Script to Post Process Simulation Data

For all the tutorial examples mentioned here, use the workspace <SystemVue Installation Directory>\Examples\Tutorials\Algorithm_Design\MATLAB_Script\PostProcessingSir wsv.

Post processing of simulation data can be done in Equations pages on the workspace tree. In order to access dataset variables from an Equations page, the using function must be called.

1. Time Domain Power Measurements
2. Histogram
3. Spectrum Averaging

### Converting MATLAB Simulation Script to SystemVue Model-based Design

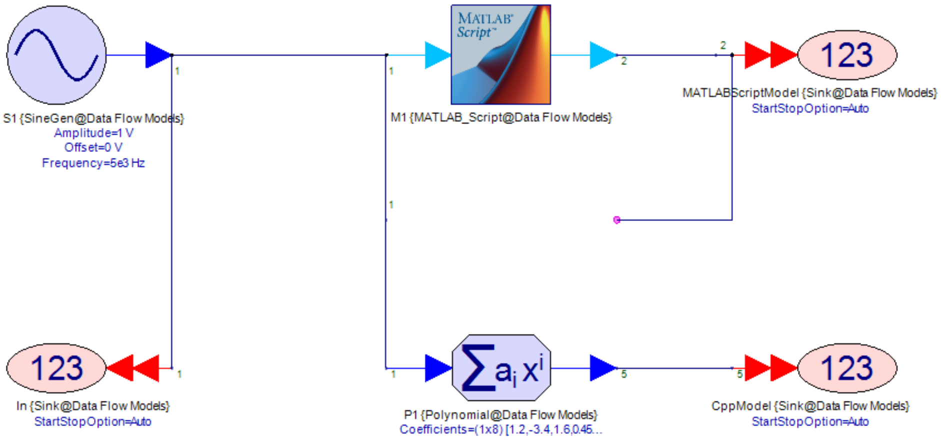
The following tutorial uses a wireless communication simulation script as an example and guides you how to convert a typical MATLAB simulation script for wireless communication into a model-based design in SystemVue. The example MATLAB script, UFMC_OFDM___TransceiverChain.m, was created and made publicly available by Alcatel-Lucent. The MATLAB script file is located in <SystemVue Installation Directory>\Examples\Tutorials\Algorithm_Design\MATLAB_Script\UFMC_OFDM___T m and the corresponding converted workspace is in <SystemVue Installation Directory>\Examples\Tutorials\Algorithm_Design\MATLAB_Script\Converting_UFMC wsv.

1. Converting UFMC Simulation Script

## Simple Unirate Model

In this tutorial example, you will create a simple unirate (reads one input sample and writes one output sample at a time) MATLAB Script model that processes the input signal by applying a polynomial expression. There is already an existing SystemVue model that does this (see Polynomial); but assuming such model does not exist, you could create one.

1. Open the workspace *MATLAB_Script_Modeling.wsv* under <SystemVue Installation Directory>\Examples\Tutorials\Algorithm_Design\MATLAB_Script.
2. Open *Design1* in the folder *1. Simple Unirate Model*.
3. Place down the MATLAB Script part by picking it from the Part Selector or using the shortcut key M and a left mouse click on the schematic.
4. Connect the MATLAB Script part between the SineGen source and the *MATLABScriptModel* Sink.



S1 {SineGen@Data Flow Models}
Amplitude=1 V
Offset=0 V
Frequency=5e3 Hz

M1 {MATLAB_Script@Data Flow Models}

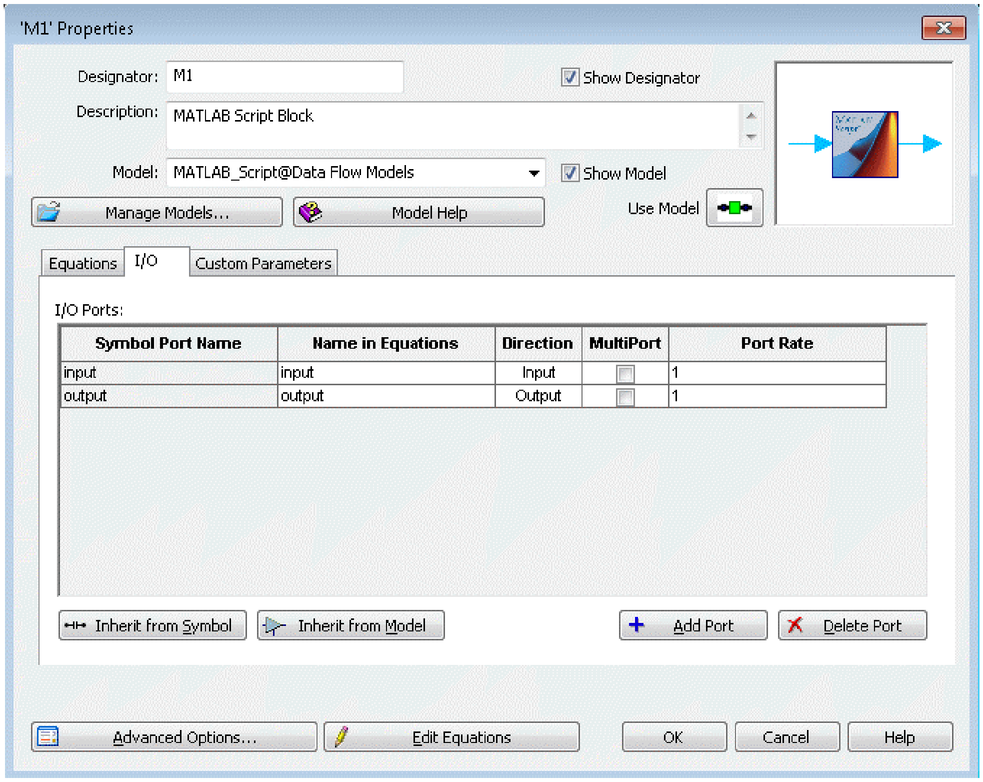
MATLABScriptModel {Sink@Data Flow Models}
StartStopOption=Auto

In {Sink@Data Flow Models}
StartStopOption=Auto

P1 {Polynomial@Data Flow Models}
Coefficients=(1x8) [1.2,-3.4,1.6,0.45...

CppModel {Sink@Data Flow Models}
StartStopOption=Auto

5. By default, the MATLAB Script part has one input and one output, and they are both unirate. Double click on the MATLAB Script part and go to the *I/O* tab to confirm this.



'M1' Properties

Designator: M1 ☑ Show Designator

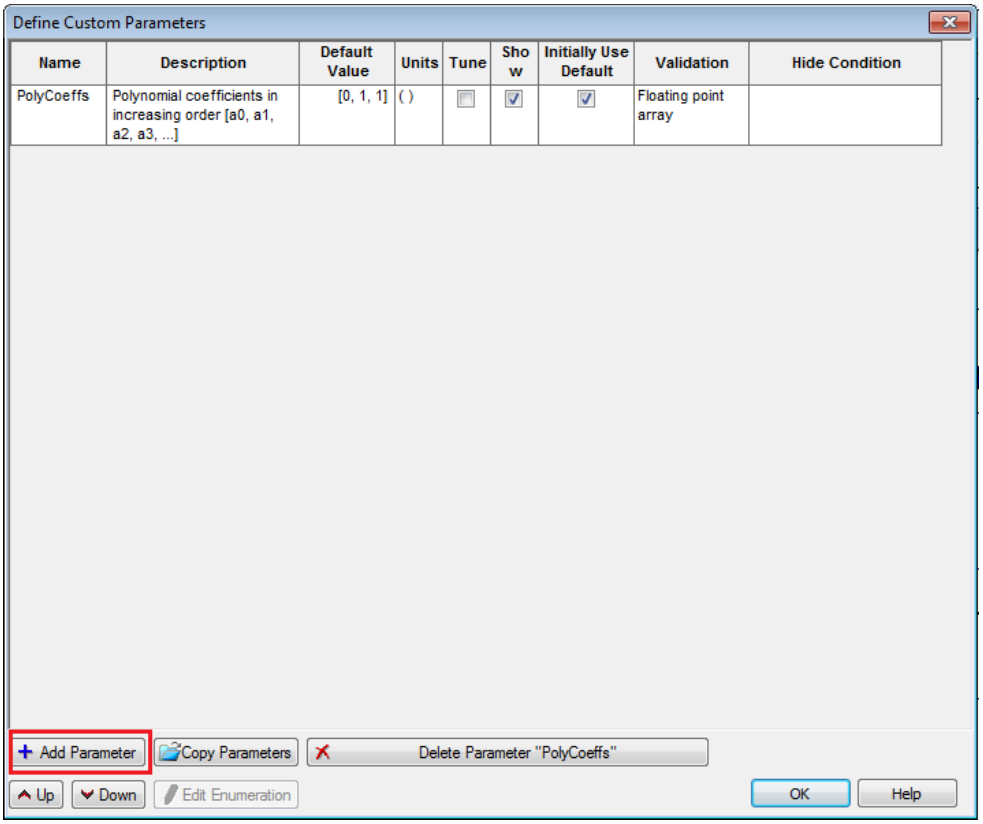
Description: MATLAB Script Block

Model: MATLAB_Script@Data Flow Models ☑ Show Model

I/O Ports:

| Symbol Port Name | Name in Equations | Direction | MultiPort | Port Rate |
|---|---|---|---|---|
| input | input | Input | □ | 1 |
| output | output | Output | □ | 1 |

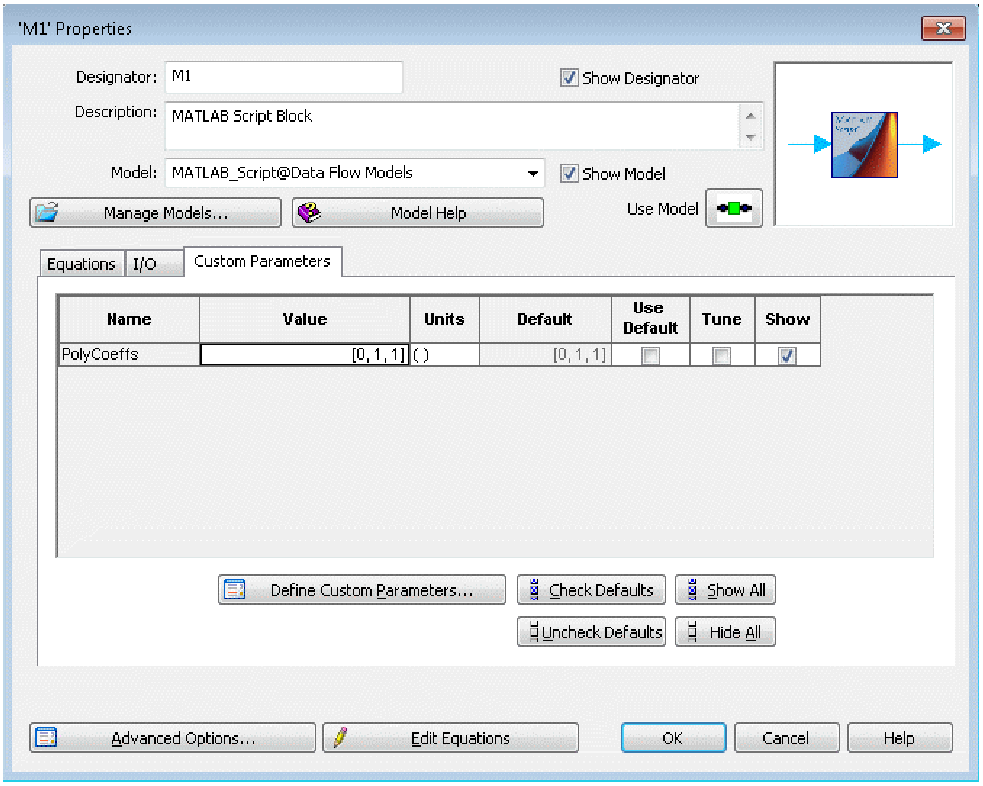
6. Go to the *Custom Parameters* tab and press the Define Custom Parameters... button. In the dialog that pops up, press the Add Parameter button to add a new parameter called *PolyCoeffs* and set its description, default value, validation, and so on, as shown in the figure.



Define Custom Parameters

| Name | Description | Default Value | Units | Tune | Show | Initially Use Default | Validation | Hide Condition |
|---|---|---|---|---|---|---|---|---|
| PolyCoeffs | Polynomial coefficients in increasing order [a0, a1, a2, a3, ...] | [0, 1, 1] | ( ) | □ | ☑ | ☑ | Floating point array | |

7. Press the OK button to apply the changes.



Designator: M1 ☑ Show Designator

Description: MATLAB Script Block

Model: MATLAB_Script@Data Flow Models ☑ Show Model

| Name | Value | Units | Default | Use Default | Tune | Show |
|---|---|---|---|---|---|---|
| PolyCoeffs | [0, 1, 1] | () | [0, 1, 1] | ☐ | ☐ | ☑ |

8. Click on the Value cell of the *PolyCoeffs* parameter and replace the default value of [0, 1, 1] with [1.2, -3.4, 1.6, 0.45, -0.6, 0.23, 0.14, -0.12] (this is the same value used in the *Coefficients* parameter of the Polynomial part in Design1).

9. Go to the *Equations* tab and try writing some MATLAB Script code that computes the *output* variable from the *input* variable and the *PolyCoeffs* array parameter using the direct polynomial computation formula

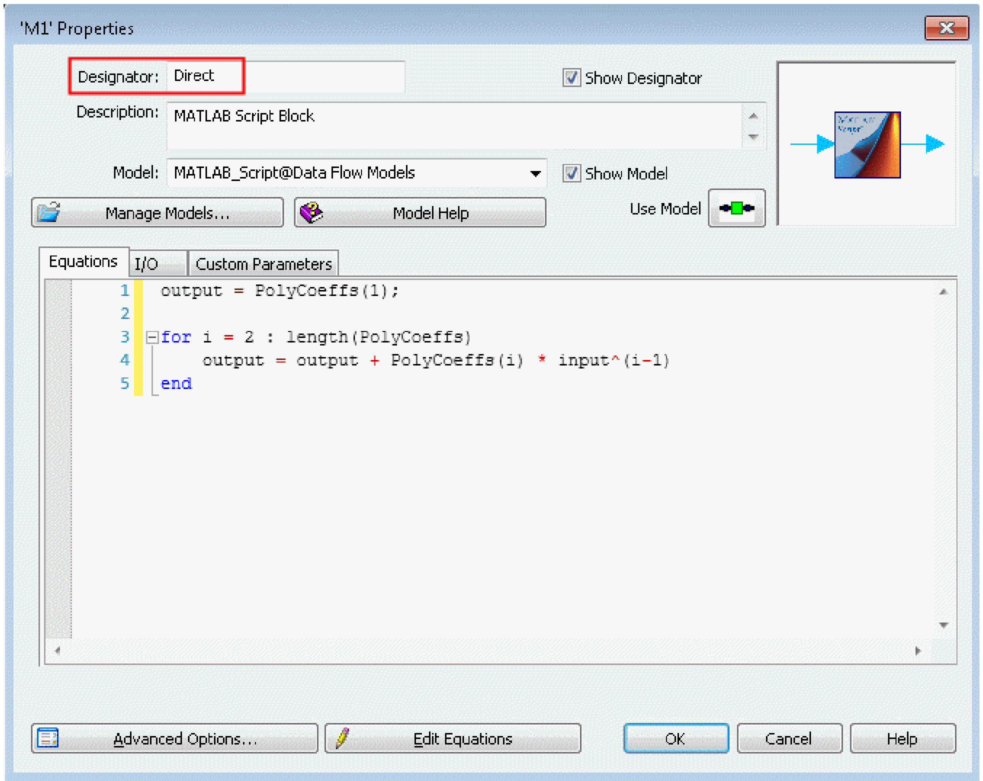
$$y = \sum_{k=0}^{N} a_k \cdot x^k$$

where *y* is *output*, *x* is *input*, and *a* is *PolyCoeffs*. Keep in mind that MATLAB Script arrays are indexed starting at 1, so PolyCoeffs(1) is the 0-th order term $a_0$. The code will look like the following:

```
output = PolyCoeffs(1);

for i = 2 : length(PolyCoeffs)
    output = output + PolyCoeffs(i) * input^(i-1)
end
```

The code first assigns *output* to the 0-th order term $a_0$, and the for loop adds the higher order terms (again keep in mind that since MATLAB Script arrays are indexed starting at 1, PolyCoeffs(i) is the coefficient of the (i-1)-th power of x).

10. Finally, change the Designator name to *Direct* (to signify that this is an implementation using a direct computation of the polynomial summation expression; different implementations will be tried later) and press the OK button.



Designator: Direct

☑ Show Designator

Description: MATLAB Script Block

Model: MATLAB_Script@Data Flow Models

☑ Show Model

```
output = PolyCoeffs(1);

for i = 2 : length(PolyCoeffs)
    output = output + PolyCoeffs(i) * input^(i-1)
end
```

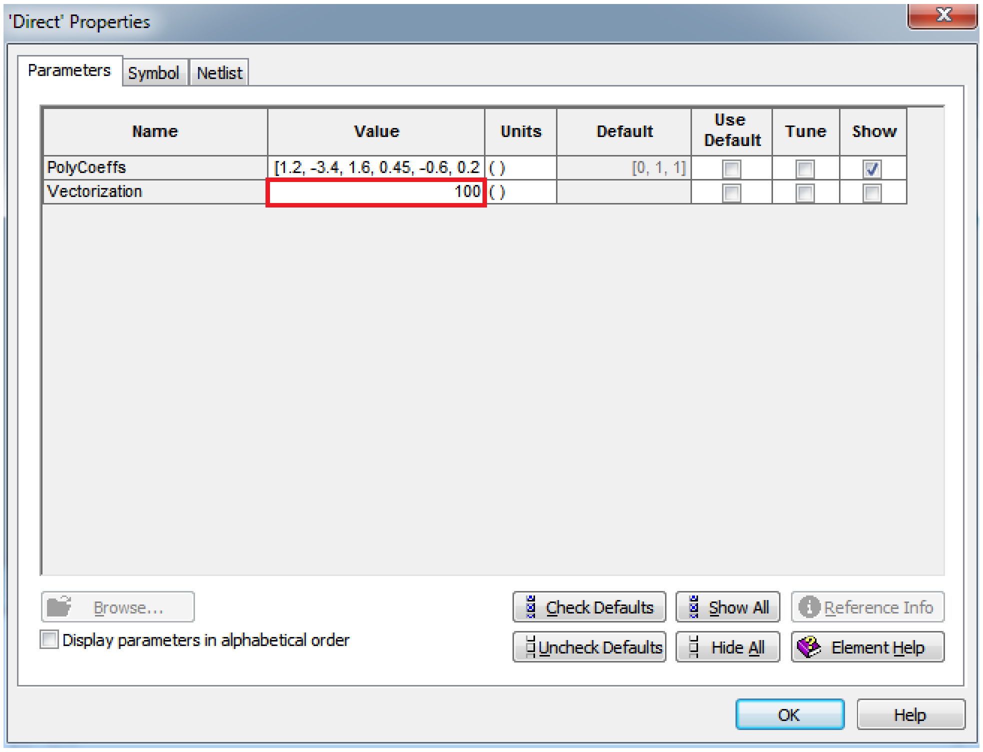
11. Run the simulation (*Design1 Analysis*) and observe the results in *Design1_Graph*. As seen, the built-in SystemVue Polynomial model and the MATLAB_Script model that was created produce the same results.

12. Now let us compare the performance. Set the *Number of Samples* in *Design1 Analysis* to 10000, deactivate (open) the MATLAB_Script part and the *MATLABScriptModel* Sink and run the simulation. Check the *Simulation Log* and find the *Execution time* reported. It should be on the order of 0.5 sec. Now deactivate (open) the Polynomial part and the *CppModel* Sink and activate the MATLAB_Script part and the *MATLABScriptModel* Sink and run the simulation again. Find the *Execution time* reported in the *Simulation Log*. It should be on the order of 5 min.

CAUTION The simulation times reported here may vary based on the CPU performance of your machine.

13. One factor that affects performance is the time it takes to execute the code in the MATLAB_Script model compared to the overhead of communicating data to and from MATLAB_Script engine for every execution of the model. The number of times this communication with the MATLAB_Script engine happens can be reduced by setting the *Vectorization* parameter of the MATLAB_Script model. For every execution of the MATLAB_Script model, the simulator will send *Port Rate * Vectorization* number of samples to the MATLAB_Script engine and then tell the engine to execute *Vectorization* number of times. At the end it will read *Port Rate * Vectorization* number of samples from every output.

14. Double click on the MATLAB_Script model and press the Advanced Options... button. The Parameters tab will show the *PolyCoeffs* parameter you defined as well as the *Vectorization* parameter. Set *Vectorization* 10 and then run the simulation again. *Execution time* reported in the *Simulation Log* should be on the order of 30 sec, which is approximately an improvement of 10 times. Set *Vectorization* 100 and then run the simulation again. *Execution time* should be on the order of 5 sec, which is approximately an improvement of 60 times (compared to the original time).

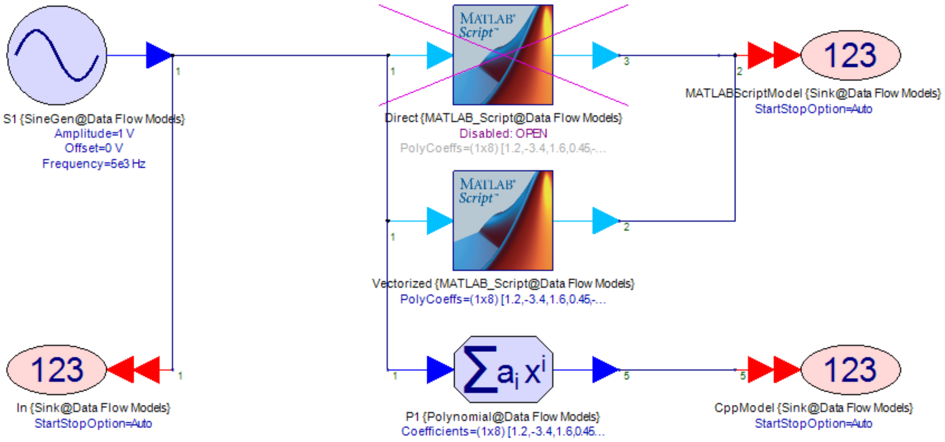


| Name | Value | Units | Default | Use Default | Tune | Show |
|---|---|---|---|---|---|---|
| PolyCoeffs | [1.2, -3.4, 1.6, 0.45, -0.6, 0.2 | ( ) | [0, 1, 1] | □ | □ | ☑ |
| Vectorization | 100 | ( ) |  | □ | □ | □ |

□ Display parameters in alphabetical order

Although this is a significant improvement, the MATLAB_Script model is still much slower than the built-in model. This difference in performance is due to the interpretive nature of MATLAB_Script in contrast to the compiled nature of the built-in C++ model. There are certain ways to improve the performance of MATLAB_Script models that will be discussed next.

15. One of the biggest performance bottlenecks in MATLAB_Script code is for loops iterating over elements of arrays and performing some operation. MATLAB_Script can efficiently handle operations on entire arrays without having to iterate over each array element. If the for loop can be replaced by some vector or matrix math, then simulation performance can improve dramatically. Let us explore how to rewrite the MATLAB_Script code without a for loop. Make a copy of the *Direct* MATLAB Script part, call it *Vectorized* and connect it between the SineGen source and the *MATLABScriptModel* Sink. Deactivate (open) the *Direct* MATLAB_Script part and activate the Polynomial part and the *CppModel* Sink. Reset the *Number of Samples* in *Design1 Analysis* to 1000 and the *Vectorization* parameter of the *Vectorized* MATLAB_Script part to 1.



S1 {SineGen@Data Flow Models}
Amplitude=1 V
Offset=0 V
Frequency=5e3 Hz

Direct {MATLAB_Script@Data Flow Models}
Disabled: OPEN
PolyCoeffs=(1x8) [1.2,-3.4,1.6,0.45,-...

MATLABScriptModel {Sink@Data Flow Models}
StartStopOption=Auto

Vectorized {MATLAB_Script@Data Flow Models}
PolyCoeffs=(1x8) [1.2,-3.4,1.6,0.45,-...

In {Sink@Data Flow Models}
StartStopOption=Auto

P1 {Polynomial@Data Flow Models}
Coefficients=(1x8) [1.2,-3.4,1.6,0.45...

CppModel {Sink@Data Flow Models}
StartStopOption=Auto

16. The polynomial evaluation sum

$$\sum_{k=0}^{N} a_k \cdot x^k$$

looks like the dot product of two vectors. MATLAB_Script can efficiently compute the dot product of two vectors *V1* and *V2* by simply multiplying them (*V1*V2*). To make the multiplication works as a dot product, make sure that *V1* is a 1 x *N* array and *V2* is a *N* x 1 array. This way *V1*V2* is going to be a 1 x 1 array representing the dot product of *V1* and *V2*. *PolyCoeffs* is already a vector, although we do not know whether the user will enter it as a row vector (1 x *N*) or a column one (*N* x 1). We can use to colon (:) operator to convert *PolyCoeffs* in a column vector from whatever form it was entered. Now create a row vector with values [1, $x$, $x^2$, $x^3$, ..., $x^{N-1}$, $x^N$]. If $x$ is a scalar and *P* is a vector, *x.^P* will create a vector of the same shape (row or column) as *P* but with values equal to *x* raised to the power of each element of *P*. Using this information, try updating the MATLAB_Script code in the *Equations* tab of the *Vectorized* MATLAB_Script part to compute the polynomial sum without using a for loop. The code might look like this:

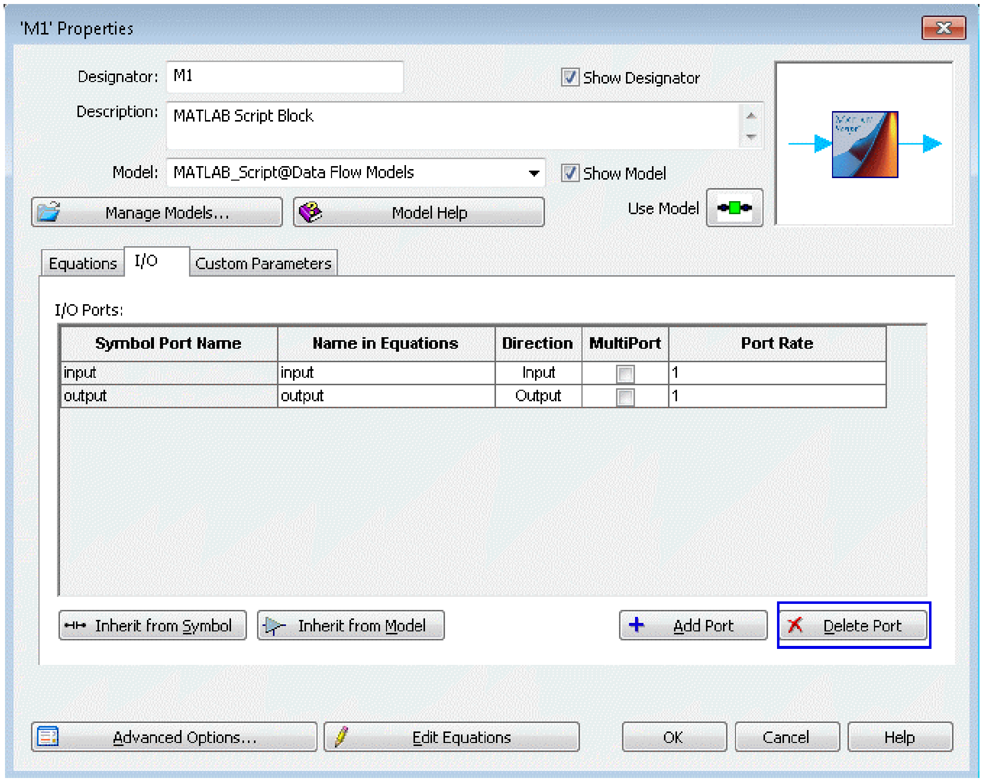
```
N = length(PolyCoeffs);
Exponents = 0:N-1;
InPowers = input.^Exponents;
output = InPowers * PolyCoeffs(:);
```

17. Run the simulation (*Design1 Analysis*) and observe the results in *Design1_Graph*. Verify that the output from the built-in SystemVue Polynomial model and the MATLAB_Script model are the same.
18. Now compare the performance of this new MATLAB_Script model to the first one. Set the *Number of Samples* in *Design1 Analysis* to 10000, deactivate (open) the Polynomial part and the *CppModel* Sink, and run the simulation. Execution time (with the *Vectorization* parameter set to 1), should be on the order of 5 min (same as with the *Direct* model). Set *Vectorization* to 100 and run the simulation again. Execution time should be on the order of 5 sec but less than what was recorded with the *Direct* model.
19. Increase the *Vectorization* parameter to 1000 and compare both models (*Direct* and *Vectorized*) again. Both models simulate faster compared to *Vectorization* = 100 and the relative speedup of the *Vectorized* model compared to the *Direct* model should be more evident. To get a more accurate estimate of the speedup increase the *Number of Samples* in *Design1 Analysis* to 100000, and simulate both models (*Direct* and *Vectorized*) with the *Vectorization* parameter to 1000. The *Vectorized* model should simulate in approximately half the time to the *Direct* one does.

## Model with State

In this tutorial example, you will create three simple MATLAB_Script models that show how to preserve state information between consecutive executions. One of the models is going to be a source.

1. Open the workspace <SystemVue Installation Directory>\Examples\Tutorials\Algorithm_Design\MATLAB_Script\MATLAB_Sc
wsv.
2. Open *Design2* in the folder *2. Model with State*.
3. First, we will create a sinusoid source. Place down the MATLAB_Script part by picking it from the Part Selector or using the shortcut key M and a left mouse click on the schematic.
4. By default, the MATLAB_Script part has one input and one output. Double click on the MATLAB_Script part, go to the *I/O* tab, and delete the input port by first selecting any editable cell in the input port row and then pressing the Delete Port button.



'M1' Properties

Designator: M1

☑ Show Designator

Description: MATLAB Script Block

Model: MATLAB_Script@Data Flow Models

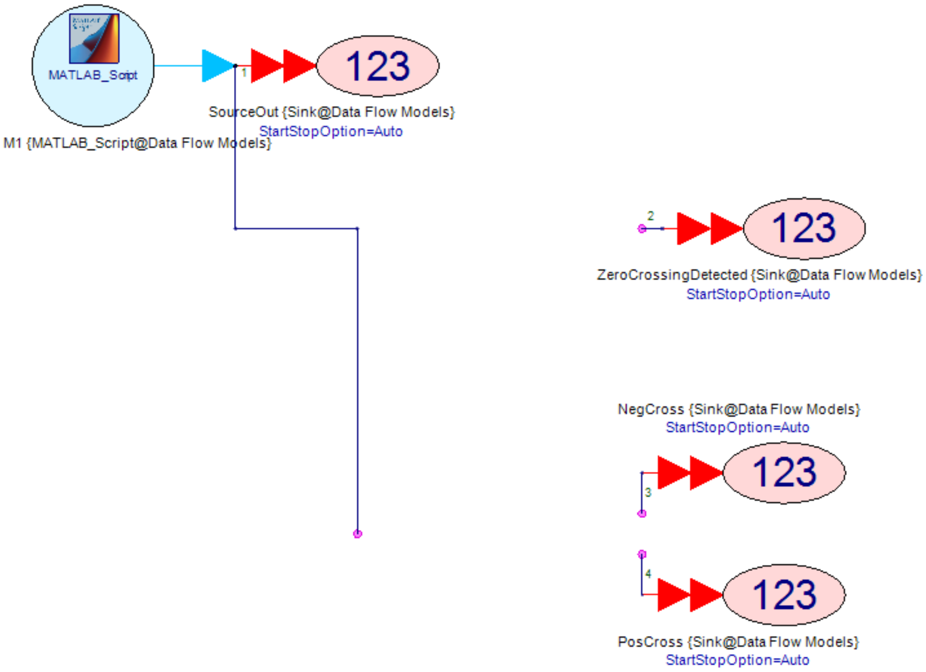
☑ Show Model

Use Model

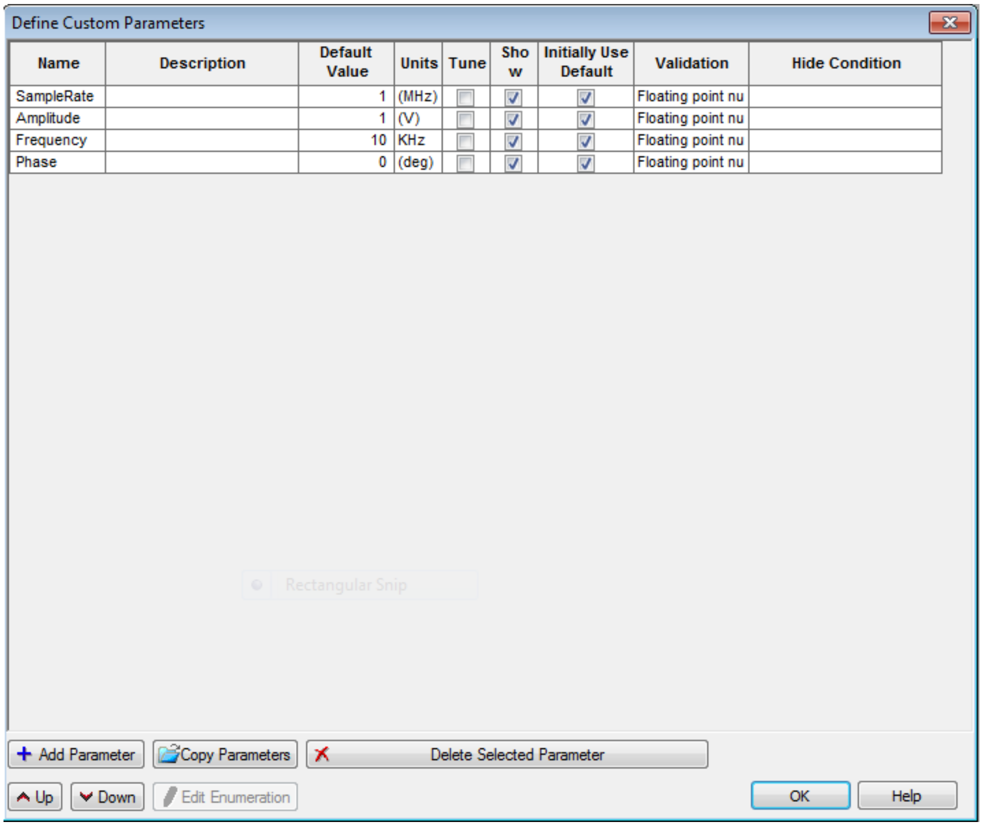
I/O Ports:

| Symbol Port Name | Name in Equations | Direction | MultiPort | Port Rate |
|---|---|---|---|---|
| input | input | Input | □ | 1 |
| output | output | Output | □ | 1 |

5. Connect the MATLAB_Script part to the *SourceOut* Sink.


MATLAB_Script
M1 {MATLAB_Script@Data Flow Models}
1
123
SourceOut {Sink@Data Flow Models}
StartStopOption=Auto
2
123
ZeroCrossingDetected {Sink@Data Flow Models}
StartStopOption=Auto
NegCross {Sink@Data Flow Models}
StartStopOption=Auto
3
123
4
123
PosCross {Sink@Data Flow Models}
StartStopOption=Auto

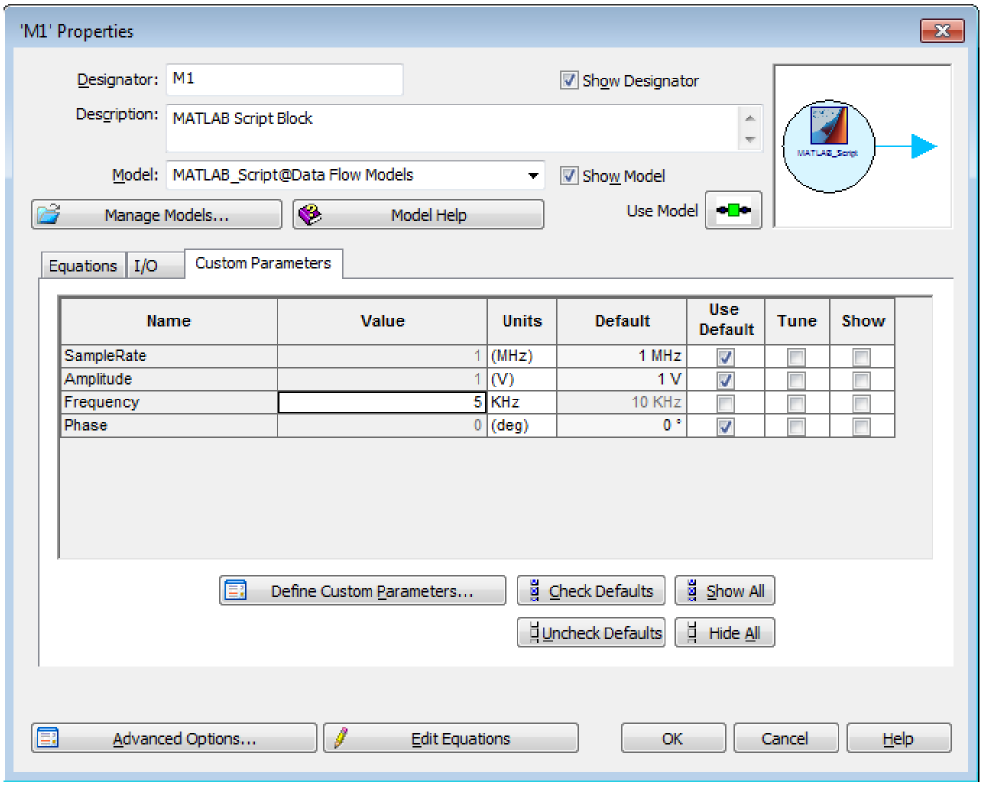
6. Go to the *Custom Parameters* tab and press the Define Custom Parameters... button. In the dialog that pops up, press the Add Parameter button to add parameters, as shown. These parameters specify the *Amplitude*, *Frequency*, and *Phase* of the generated sinusoid waveform. The *SampleRate* parameter specifies at what time intervals (1/*SampleRate*) samples of the sinusoid waveform will be generated.



Define Custom Parameters

| Name | Description | Default Value | Units | Tune | Show | Initially Use Default | Validation | Hide Condition |
|---|---|---|---|---|---|---|---|---|
| SampleRate | | 1 | (MHz) | □ | ☑ | ☑ | Floating point nu | |
| Amplitude | | 1 | (V) | □ | ☑ | ☑ | Floating point nu | |
| Frequency | | 10 | KHz | □ | ☑ | ☑ | Floating point nu | |
| Phase | | 0 | (deg) | □ | ☑ | ☑ | Floating point nu | |

7. Press the OK button to apply the changes.



Designator: M1 ☑ Show Designator

Description: MATLAB Script Block

Model: MATLAB_Script@Data Flow Models ☑ Show Model

| Name | Value | Units | Default | Use Default | Tune | Show |
|---|---|---|---|---|---|---|
| SampleRate | 1 | (MHz) | 1 MHz | ☑ | □ | □ |
| Amplitude | 1 | (V) | 1 V | ☑ | □ | □ |
| Frequency | 5 | KHz | 10 KHz | □ | □ | □ |
| Phase | 0 | (deg) | 0 ° | ☑ | □ | □ |

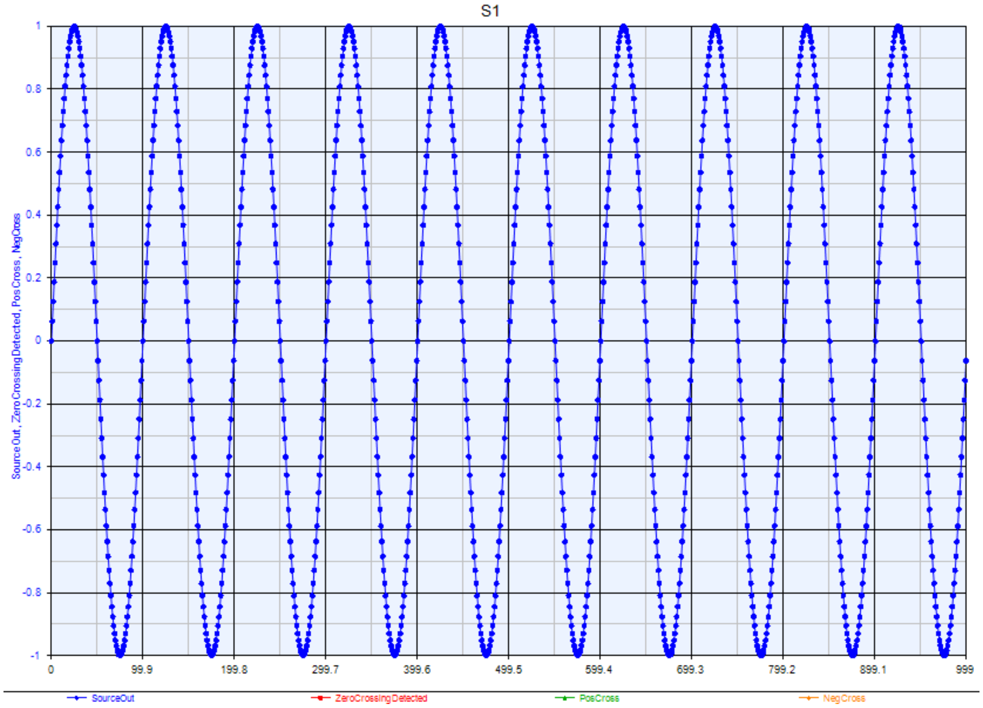
8. Go to the *Equations* tab and try writing some MATLAB_Script code that will generate a sine waveform of the form *output* = *Amplitude*·sin( 2·π·*Frequency*·*t* + *Phase* ). This expression can be used as is in the MATLAB_Script code. The only problem is we need to increment *t* by 1/*SampleRate* every time the code is executed. This means the model needs to remember the value of *t* from the previous execution, and it also needs a way to give *t* an initial value. This is what persistent variables are used for. Persistent variables preserve their value over successive executions of the MATLAB_Script model. Their initial value is the empty value [], which is used to initialize them by checking whether they are empty (use the isempty function). The code might look like this:

```
persistent MyTime;

if ( isempty( MyTime ) )
    MyTime = 0.0;
end

output = Amplitude * sin( 2 * pi * Frequency *
MyTime + Phase );
MyTime = MyTime + 1.0 / SampleRate;
```

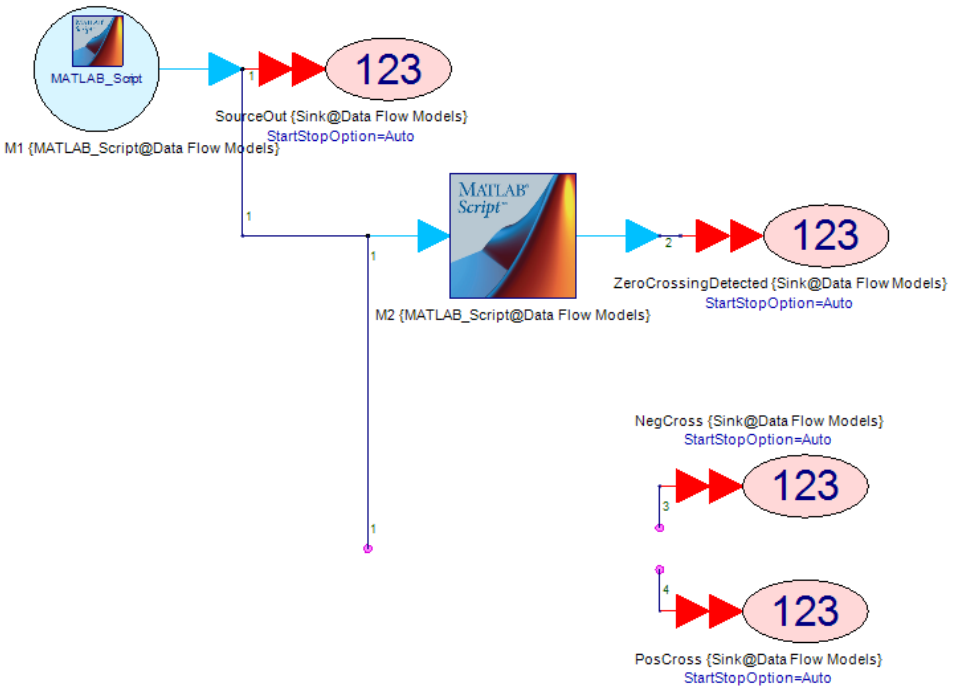
The code first declares the *MyTime* variable to be a persistent one. It then checks whether it is empty and initializes to 0. This will happen only in the first execution of the model. After the first execution, *MyTime* will no longer be empty, the isempty( ) function will return a false status, and the line

*MyTime* = 0.0 will not be executed again. The rest of the code just evaluates the sinusoid waveform at *t* = *MyTime* and then updates *MyTime* with the correct value for the next execution of the model.

9. Press the OK button to apply the changes you made.
10. Deactivate (open) the Sinks *ZeroCrossingDetected*, *PosCross*, and *NegCross*, and run *Design2 Analysis*. Open *Design2_Graph* and observe the generated waveform (ignore the errors about undefined function or variable for 'ZeroCrossingDetected', 'PosCross', and 'NegCross'; these errors occur because the associated Sinks were deactivated). For a *SampleRate* of 1 MHz and a sine wave *Frequency* of 10 kHz, you should receive 100 samples per period of the sine wave. Since 1000 samples were simulated (see *Number of Samples in _Design2 Analysis*), you should see 10 periods of the sine wave. This is what the graph displays. However, the x-axis represents sample number and not time (this will be addressed later in this tutorial). Play with the parameters *SampleRate*, *Amplitude*, *Frequency*, and *Phase*, observe the graph, and determine if the model is working properly. Reset the parameters to their default values before moving to the next step.



11. Now create a new MATLAB_Script model that takes as input the signal generated by the sinusoid source and tries to detect zero-crossings (outputs 1 when a zero-crossing is detected and 0 otherwise). Place a new MATLAB_Script part on the schematic and connect it between the source output and the Sink *ZeroCrossingDetected*. Activate the Sink *ZeroCrossingDetected*.

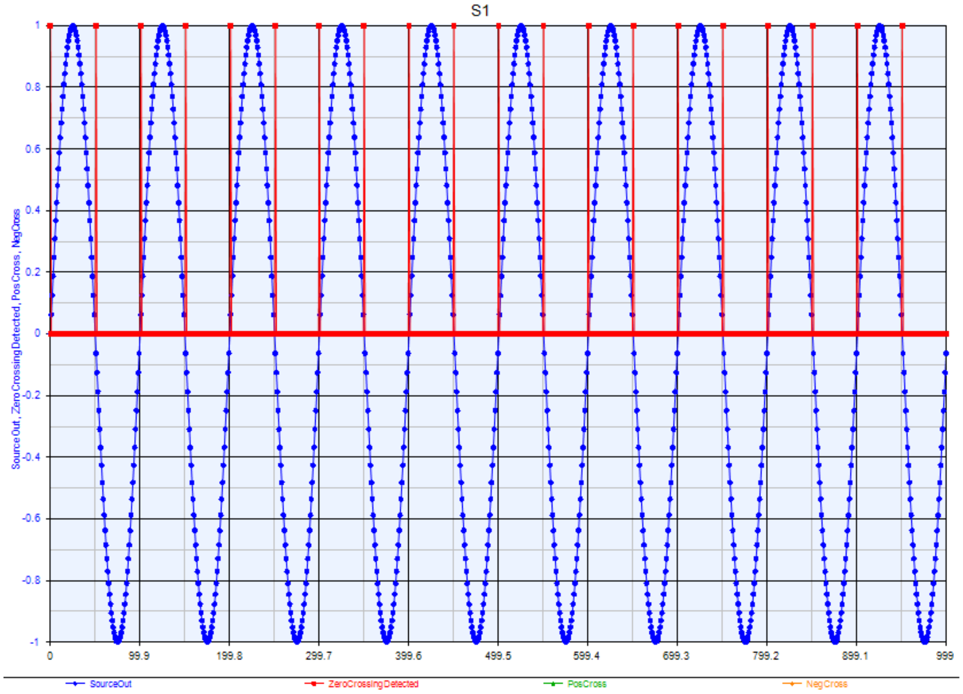

MATLAB_Script
123
SourceOut {Sink@Data Flow Models}
StartStopOption=Auto
M1 {MATLAB_Script@Data Flow Models}
MATLAB Script
123
ZeroCrossingDetected {Sink@Data Flow Models}
StartStopOption=Auto
M2 {MATLAB_Script@Data Flow Models}
NegCross {Sink@Data Flow Models}
StartStopOption=Auto
123
123
PosCross {Sink@Data Flow Models}
StartStopOption=Auto

12. The default one input and one output setup are sufficient for this MATLAB_Script model. No parameters need to be defined, so go to its Equations tab and try writing some MATLAB_Script code to detect zero-crossings. Detecting a zero-crossing requires knowledge of the current signal sample as well as the previous one. Make sure you use persistent variables to keep track of the previous sample. The code might look like this:

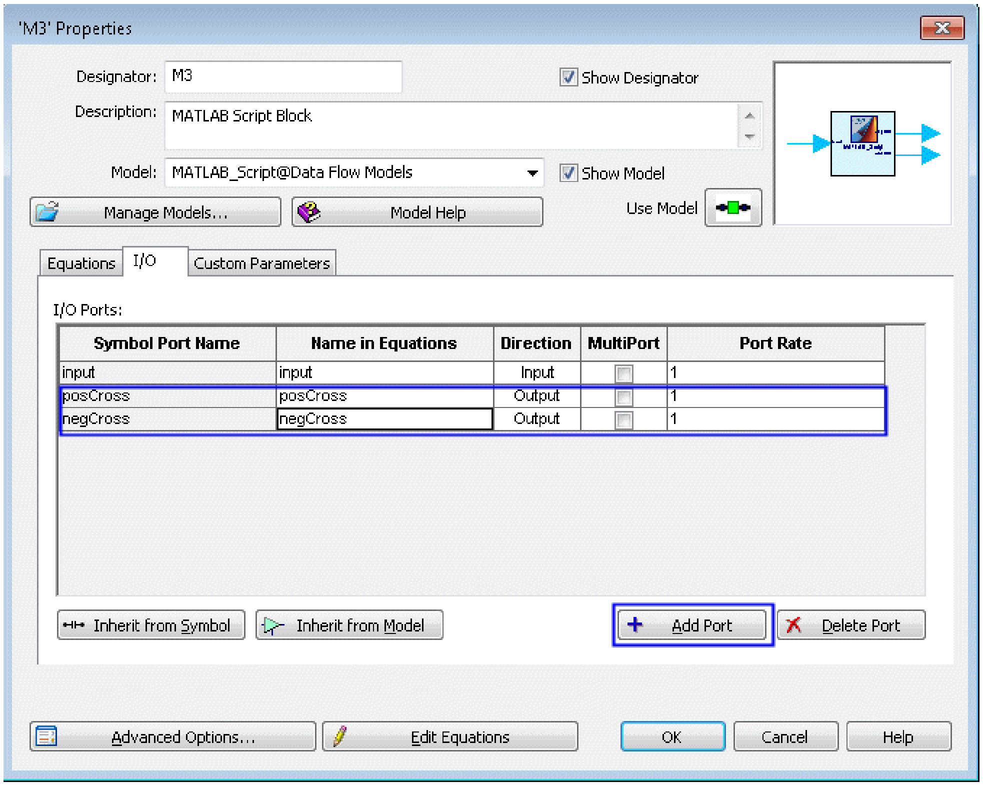
```
persistent PrevIn;

if ( isempty( PrevIn ) )
    PrevIn = 0.0;
end

if ( ( input == 0 ) || ( input * PrevIn < 0 ) )
    output = 1.0;
else
    output = 0.0;
end

PrevIn = input;
```

This code declares a persistent variable called *PrevIn*, initializes it, and uses the fact that if you multiply two numbers one of which is positive and one which is negative (in this case, a zero-crossing must have occurred between the two samples), the result is negative.

13. Run *Design2 Analysis* and observe the waveforms in *Design2_Graph*. As seen, the red waveform (*ZeroCrossingDetected*) goes high (1) every time a zero-crossing is detected



14. Finally, refine the zero-crossing detector model to distinguish between positive and negative zero-crossings. Positive zero-crossings are indicated by a value of 0.5 on a *posCross* output and negative zero-crossings are indicated by a value of -0.5 on a *negCross* output. Place a new MATLAB_Script part on the schematic. Double click on it and go to the *I/O* tab. Rename the output port to *posCross* and add a new port called *negCross* .



'M3' Properties

Designator: M3 ☑ Show Designator

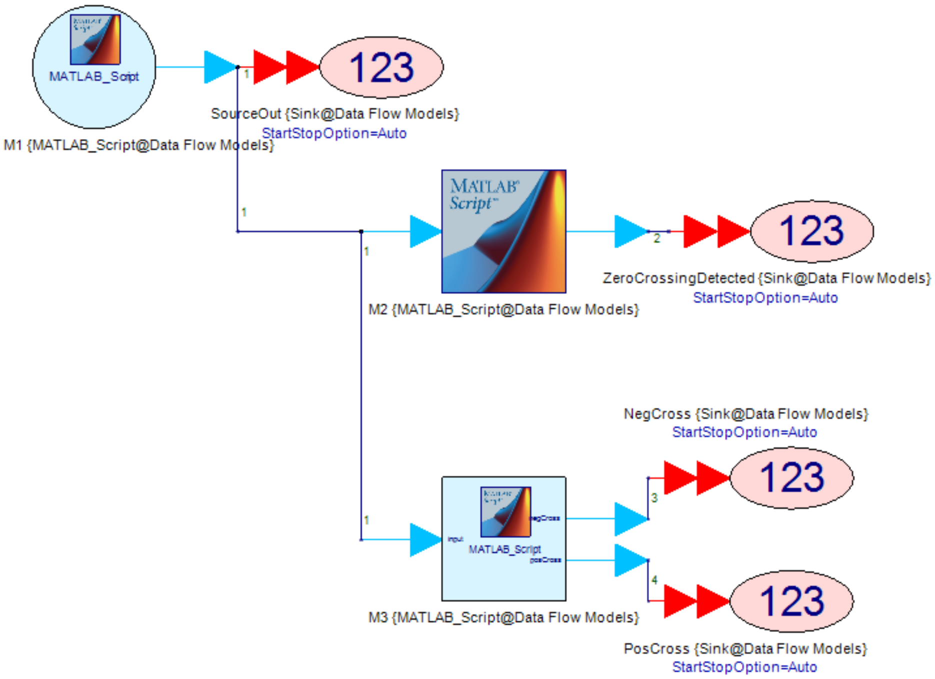
Description: MATLAB Script Block

Model: MATLAB_Script@Data Flow Models ☑ Show Model

I/O Ports:

| Symbol Port Name | Name in Equations | Direction | MultiPort | Port Rate |
|---|---|---|---|---|
| input | input | Input | □ | 1 |
| posCross | posCross | Output | □ | 1 |
| negCross | negCross | Output | □ | 1 |

15. Connect the new part between the source output and the Sinks (*PosCross* and *NegCross*). Activate both Sinks.


MATLAB_Script
M1 {MATLAB_Script@Data Flow Models}
123
SourceOut {Sink@Data Flow Models}
StartStopOption=Auto
1
1
1
MATLAB Script
M2 {MATLAB_Script@Data Flow Models}
2
123
ZeroCrossingDetected {Sink@Data Flow Models}
StartStopOption=Auto
NegCross {Sink@Data Flow Models}
StartStopOption=Auto
123
3
1
input
negCross
MATLAB_Script
posCross
M3 {MATLAB_Script@Data Flow Models}
4
123
PosCross {Sink@Data Flow Models}
StartStopOption=Auto

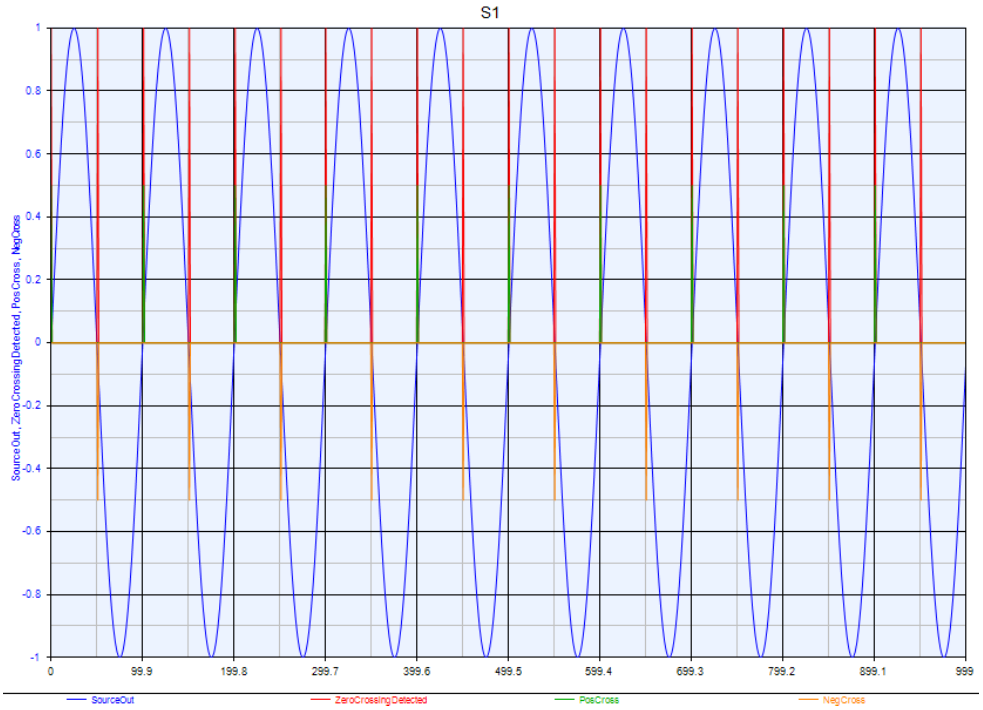
16. Although the persistent variable technique described in this tutorial example is used to track the previous input value, there is an alternative way, which relies on persistent variables but is slightly more seamless and avoids

executing the isempty( ) check every time the models executes. The MATLAB Script model allows users to define three functions: *Initialize*, *Run*, *Finalize* (see How the Simulator invokes the MATLAB Script Block. As implied by their names, *Initialize* is executed only once at the beginning of the simulation, *Run* is executed repeatedly during the simulation, and *Finalize* is executed only once at the end of the simulation. By default, if you write code in the *Equations* tab without specifying any function name, it is assumed to be the code of the *Run* function. For this refined zero-crossing detector model, use the *Initialize* function to initialize the predefined and persistent variable called *M_State*. The code might look like this:

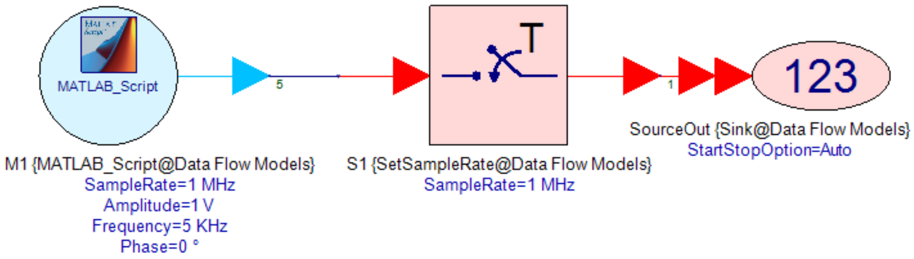
```
function Initialize
    M_State.PrevIn = 0.0;
end

function Run
    posCross = 0.0;
    negCross = 0.0;

    if ( ( input == 0 ) || ( input * M_State.PrevIn
< 0 ) )
        if ( M_State.PrevIn > 0 )
            negCross = -0.5;
        else
            posCross = 0.5;
        end
    end

    M_State.PrevIn = input;
end
```

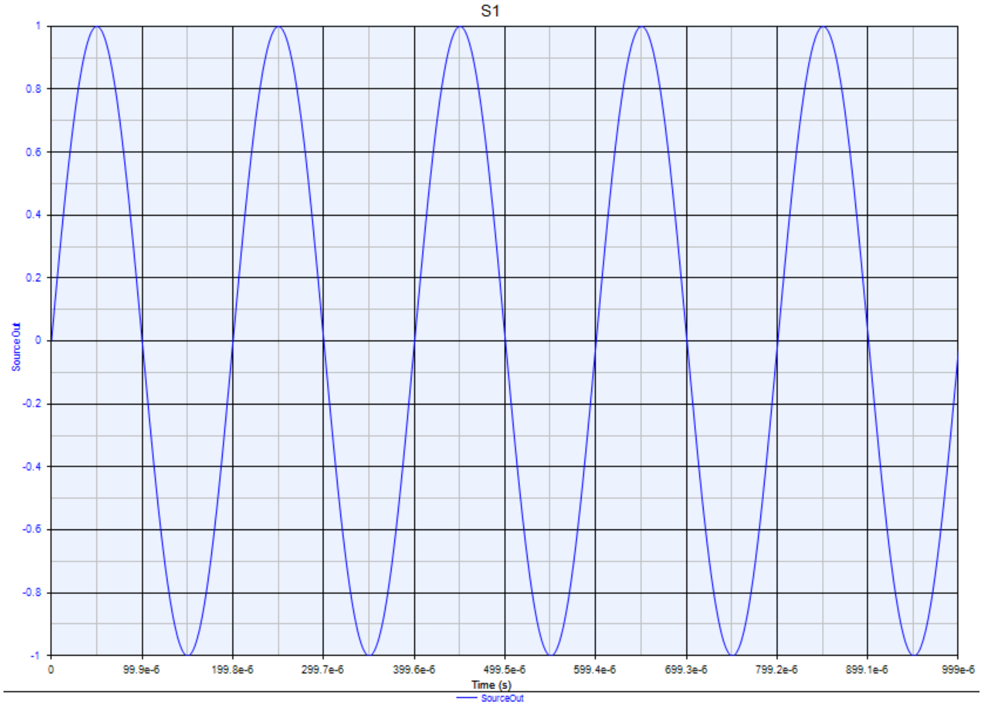
The *M_State* variable is actually a structure variable that can hold data of any type (scalar, array, cell array) and size. Just follow its name with a dot '.' and a name (for example, MyVar) and a new variable will be created automatically.

17. Run *Design2 Analysis* and observe the waveforms in *Design2_Graph*. As seen, the green waveform (*PosCross*) goes high (0.5) every time a positive zero-crossing is detected and the orange waveform (*NegCross*) goes low (-0.5) every time a negative zero-crossing is detected.



18. Before finishing this tutorial example, let us address the issue of why the signal generated from an MATLAB_Script source is plotted versus sample number and not versus time. The MATLAB_Script model is an *Untimed* model, meaning it cannot set timing properties for the simulation. The fact that a *SampleRate* parameter was defined for it does not make it a *Timed* model. If there is a need to generate a signal with time information using an MATLAB_Script source, the MATLAB_Script part needs to be followed by the SetSampleRate model, as shown below.

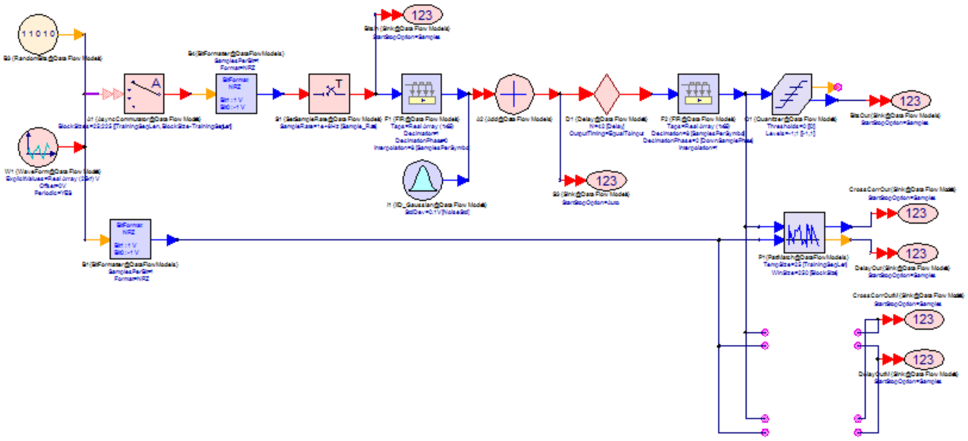




## Multirate Model

In this tutorial example, you will create a multirate (reads more than one input sample and/or writes more than one output sample at a time) MATLAB_Script model that tries to find a certain bit pattern in a block of data (the SystemVue built-in PattMatch model will be used as a reference to validate the MATLAB_Script code).

1. Open the workspace <SystemVue Installation Directory>\Examples\Tutorials\Algorithm_Design\MATLAB_Script\MATLAB_Sc wsv.
2. Open *Design3* in the folder *3. Multirate Model*.



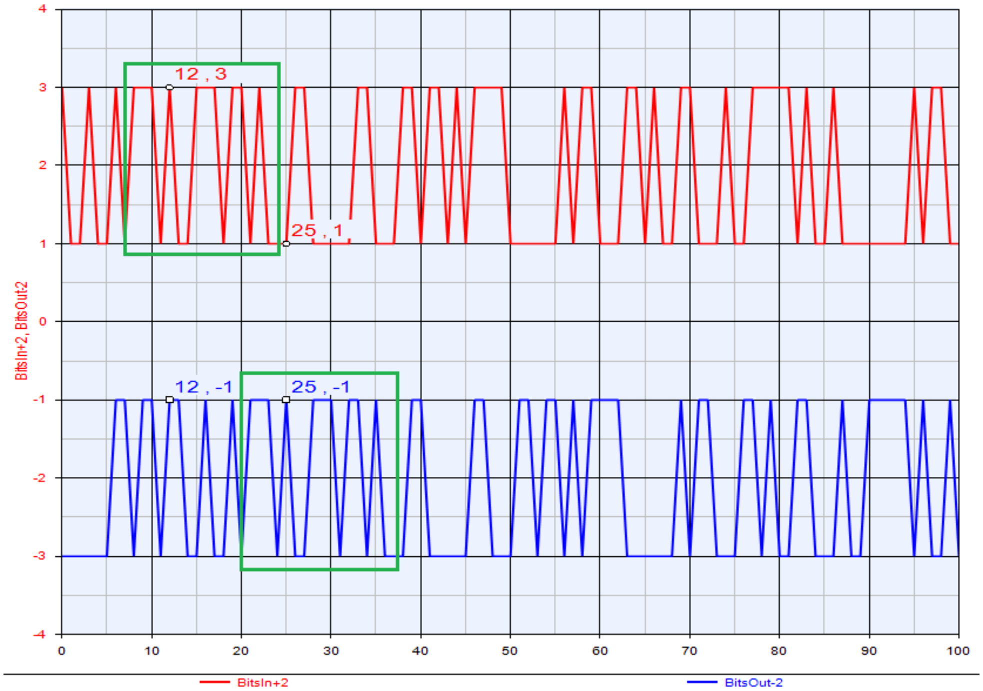
3. Here is a description of the system in *Design3*.

- The WaveForm, RandomBits, and AsyncCommutator models are used to create blocks of bits, where the first few bits are always a known bit pattern (training sequence). The exact bit pattern of the training sequence and the size of the data block can be set in the Equations page *InEqns3*.
- The bits are converted to an NRZ waveform using the BitFormatter model.
- A sample rate is assigned to the NRZ waveform using the SetSampleRate model.
- The NRZ waveform is filtered by an interpolating (interpolation factor of 8) Root-Raised-Cosine filter designed for a symbol rate of 1 MHz.
- Noise and delay are added to the signal. The noise standard deviation and the delay are set in the Equations page *InEqns3*.
- Then the signal is filtered by a decimating (decimation factor of 8) Root-Raised-Cosine filter designed for a symbol rate of 1 MHz. The decimation phase of the filter is automatically computed in the Equations page *InEqns3*.
- The output of the filter goes through a Quantizer to recover the bits (in NRZ form).
- The output of the filter is also connected to the *Window* input of the PattMatch model. The *Template* input is connected to the output of the WaveForm model (which generates the training sequence). The PattMatch model outputs the cross-correlation between the *Window* and the *Template* input as well as the index (shift) at which the cross-correlation maximum occurred.

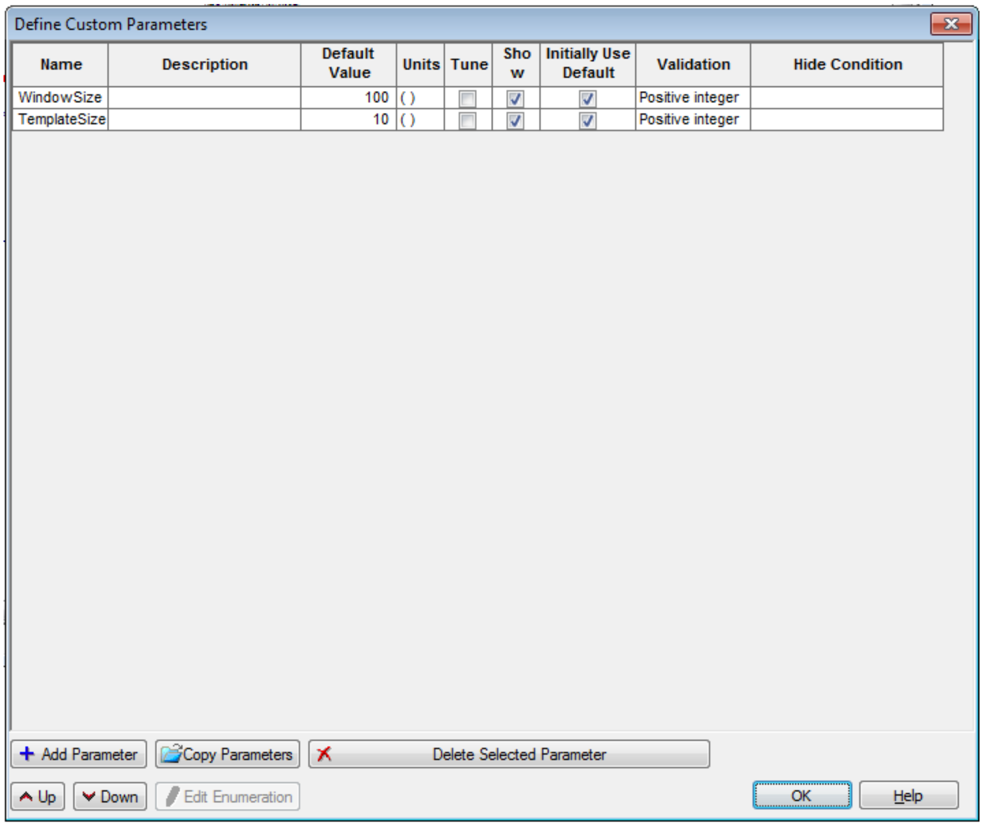
**CAUTION** **If you change the *System Sample Rate* in *Design3 Analysis* , you might be required to redesign the filters.**

4. The total delay introduced by this system is 4 + floor( *Delay* / 8 ) + 4, where the first and last 4 are the delays introduced by the FIR filters *F1* and *F2*, *Delay* is the delay in the Delay model (this is set in the Equations page *InEqns3*) and 8 is the interpolation/decimation factor. For the initial value of *Delay* = 42, the total delay is 13.
5. Deactivate (open) the Sinks *CrossCorrOutM* and *DelayOutM* and run *Design3 Analysis* (ignore any errors about undefined function or variable for 'CrossCorrOutM' and 'DelayOutM' for the moment; these occur because the associated Sinks were deactivated). Look at *Delay_Table3*. The value recorded in the *DelayOut* Sink (connected to the *index* output of PattMatch) matches the computed delay of 13. By observing the *BitsIn* (red) and *BitsOut* (blue) waveforms in *BitsIn_BitsOut_Graph3*, it can be confirmed that the introduced delay is 13. An offset of 2 and -2 is applied to *BitsIn* and *BitsOut*, respectively, to vertically separate the two waveforms so that bit patterns can be easily identified. As seen, the bit at index 12 in the *BitsIn* waveform corresponds to the bit at index 25 in the *BitsOut* waveform (a delay of 13

bits). Try a few different values for the *Delay* variable in *InEqns3* and verify that the delay recorded in the Sink *DelayOut* matches the expected one of 4 + floor( *Delay* / 8 ) + 4 = 8 + floor( *Delay* / 8 ).



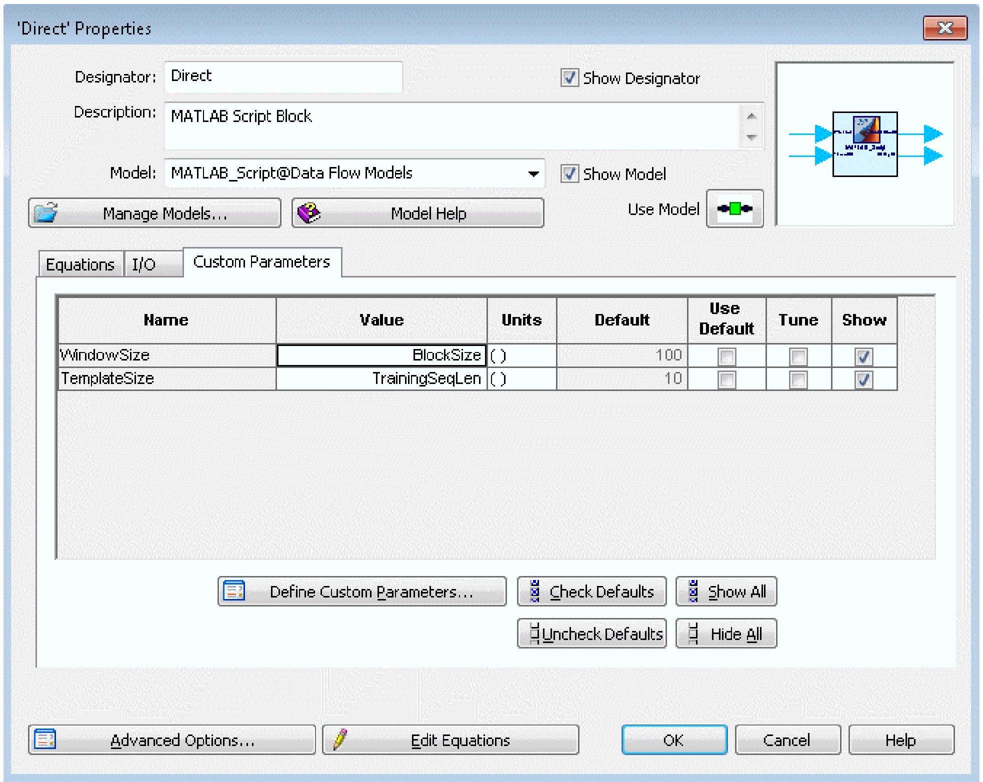
6. The goal of this tutorial is the create an MATLAB_Script model that behaves the same as the built-in SystemVue PattMatch model.
7. Place down the MATLAB_Script part by picking it from the Part Selector or using the shortcut key M and a left mouse click on the schematic.
8. Double click on the MATLAB_Script part and go to the *Custom Parameters* tab. Press the Define Custom Parameters... button. In the dialog that pops up, press the Add Parameter button to add parameters, as shown. Press the OK button to apply the changes.



Define Custom Parameters

| Name | Description | Default Value | Units | Tune | Show | Initially Use Default | Validation | Hide Condition |
|---|---|---|---|---|---|---|---|---|
| WindowSize | | 100 | ( ) | □ | ☑ | ☑ | Positive integer | |
| TemplateSize | | 10 | ( ) | □ | ☑ | ☑ | Positive integer | |

Click OK, then change the values of the custom parameters as shown:



'Direct' Properties

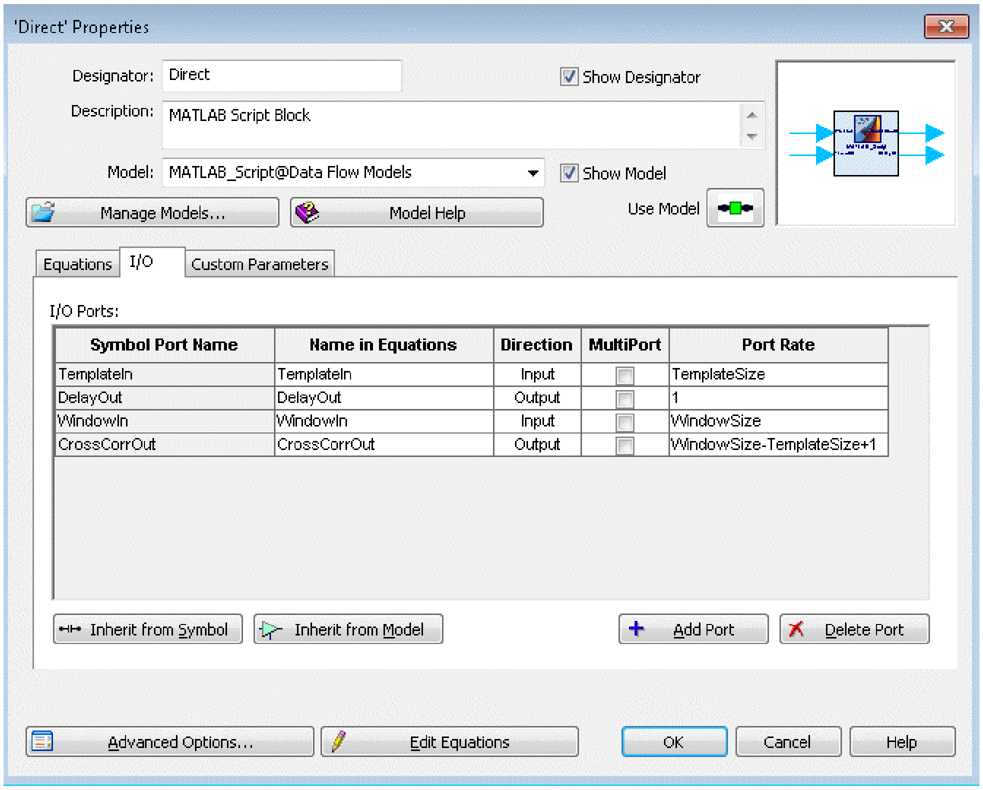
Designator: Direct ☑ Show Designator

Description: MATLAB Script Block

Model: MATLAB_Script@Data Flow Models ☑ Show Model

| Name | Value | Units | Default | Use Default | Tune | Show |
|---|---|---|---|---|---|---|
| WindowSize | BlockSize | ( ) | 100 | □ | □ | ☑ |
| TemplateSize | TrainingSeqLen | ( ) | 10 | □ | □ | ☑ |

9. Go to the *IO* tab and rename the input and output ports to TemplateIn and DelayOut, respectively. Add two more ports as shown (the names shown in the Symbol Port Name column are updated as soon as you apply the changes). Make sure you set the values in the Port Rates column, as shown (the entries in this column make use of the parameters defined in the *Custom Parameters* tab).



'Direct' Properties

Designator: Direct

☑ Show Designator

Description: MATLAB Script Block

Model: MATLAB_Script@Data Flow Models

☑ Show Model

I/O Ports:

| Symbol Port Name | Name in Equations | Direction | MultiPort | Port Rate |
|---|---|---|---|---|
| TemplateIn | TemplateIn | Input | □ | TemplateSize |
| DelayOut | DelayOut | Output | □ | 1 |
| WindowIn | WindowIn | Input | □ | WindowSize |
| CrossCorrOut | CrossCorrOut | Output | □ | WindowSize-TemplateSize+1 |

By defining port rates greater than 1, the model becomes a multirate one, meaning it will read and/or write multiple samples every time it executes. The way this model has been defined, each time it executes, it will read (consume) *TemplateSize* samples from the *TemplateIn* port and *WindowSize* samples from the *WindowIn* port, and it will write (produce) 1 sample at the *DelayOut* port and *WindowSize – TemplateSize + 1* samples at the *CrossCorrOut* port. The *WindowSize – TemplateSize + 1* samples at the *CrossCorrOut* port correspond to the cross-correlation values between the *WindowSize* data points read from the *WindowIn* port and the *TemplateSize* data points read from the *TemplateIn* port for the *WindowSize – TemplateSize + 1* different positions (shifts) the *TemplateIn* data can have in the *WindowIn* block, starting at the position where the first data point in both *WindowIn* and *TemplateIn* align (shift of 0) to the position where the last data point in both *WindowIn* and *TemplateIn* align (shift of *WindowSize – TemplateSize*). The 1 sample at the *DelayOut* port corresponds to the position (shift) where the maximum cross-correlation occurred.

When an MATLAB_Script part port is defined to have a rate greater than one:

- if it is an input port, the data collected from the port will be placed in an array whose first dimension is the same as the port rate; if the input data is not scalar but a matrix, then the data collected from the port is placed in a multi-dimensional array whose first dimension is the same as the port rate

- if it is an output port, the MATLAB_Script code you write that produces the data for the port needs to place the data in an array whose first dimension is the same as the port rate (otherwise zero padding or truncation will occur); if the output data is not scalar but a matrix, then the data produced for the port needs to be placed in a multi-dimensional array whose first dimension is the same as the port rate

  For the example, we are working on, since all input and output data are scalar real or integer numbers, we should expect that *TemplateIn* will be a *TemplateSize* x 1 real array and *WindowIn* will be a *WindowSize* x 1 real array. The MATLAB_Script code we write should produce a single value for the *DelayOut* variable (port) and a *(WindowSize – TemplateSize + 1)* x 1 array for the *CrossCorrOut* variable (port).

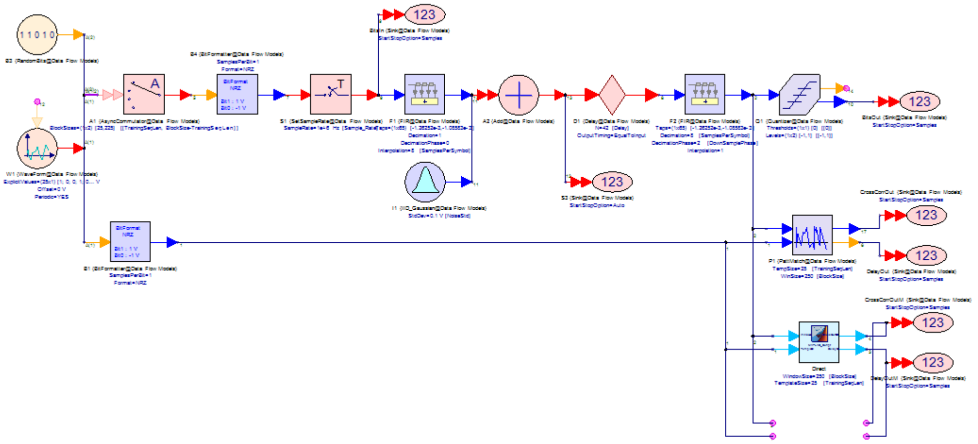
10. Go to the *Equations* tab and using the information presented above and the equations in the documentation of the PattMatch model try writing some MATLAB_Script code to implement the function of the PattMatch model. Keep in mind that MATLAB_Script arrays are indexed starting at 1. The code might look like this:

```
CrossCorrOut = zeros(WindowSize-TemplateSize+1,1);

for n=0:WindowSize-TemplateSize

    Numerator = 0.0;
    Denominator = 0.0;

    for m=1:TemplateSize
        Numerator = Numerator + TemplateIn(m) *
WindowIn(m+n);
        Denominator = Denominator + WindowIn(m+n) *
WindowIn(m+n);
    end

    if ( Denominator > 0.0 )
        CrossCorrOut(n+1) = Numerator / Denominator;
    else
        CrossCorrOut(n+1) = 0.0;
    end
end
```

```
DelayOut = find( CrossCorrOut == max(CrossCorrOut),
1 ) - 1;
```

The *CrossCorrOut* variable is first initialized to an all zero array with the correct dimensions *(WindowSize – TemplateSize + 1)* x 1. Then for every possible shift *n* (0 to *WindowSize – TemplateSize*) we compute the numerator and denominator of the expression found in the PattMatch doc and if the denominator is non-zero we divide the two to get the cross-correlation. Notice that the computed cross-correlation value is assigned to the *n* + 1 element of the *CrossCorrOut* array. This is because 0 ≤ *n* ≤ WindowSize – TemplateSize, but the index for an MATLAB_Script array has to be greater than 0. Finally, the *DelayOut* is computed using the find() function to find the index of the maximum *CrossCorrOut* array element (1 is subtracted to account for the fact the first *CrossCorrOut* array element correspond to a shift (delay) of 0).

11. Set the Designator name to *Direct*, press the OK button, and connect the MATLAB_Script part as shown below.



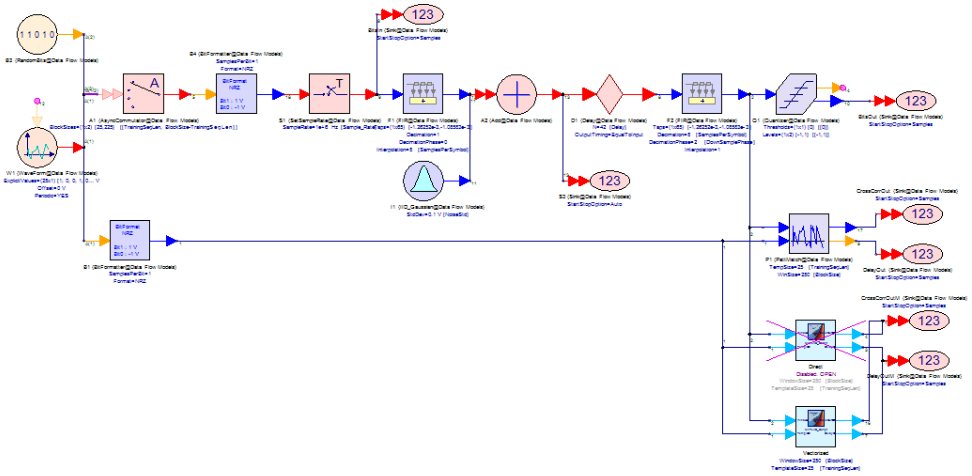
1 1 0 1 0
B3 (RandomBits@Data Flow Models)
A1 (AsyncCommutator@Data Flow Models)
BlockSizes=(1x2) [25,225] [[TrainingSeqLen, BlockSize-TrainingSeqLen]]
W1 (WaveForm@Data Flow Models)
ExplicitValues=(25x1) [1; 0; 0; 1; 0.... V
Offset=0 V
Periodic=YES
B4 (BitFormatter@Data Flow Models)
SamplesPerBit=1
Format=NRZ
BitFormat
NRZ
Bit1 : 1 V
Bit0 : -1 V
B1 (BitFormatter@Data Flow Models)
SamplesPerBit=1
Format=NRZ
S1 (SetSampleRate@Data Flow Models)
SampleRate=1e+6 Hz [Sample_Rate]
F1 (FIR@Data Flow Models)
Taps=(1x65) [-1.26252e-3,-1.05553e-3]
Decimation=1
DecimationPhase=0
Interpolation=5 [SamplesPerSymbol]
BitsIn (Sink@Data Flow Models)
StartStopOption=Samples
123
I1 (IID_Gaussian@Data Flow Models)
StdDev=0.1 V [NoiseStd]
A2 (Add@Data Flow Models)
D1 (Delay@Data Flow Models)
N=42 [Delay]
OutputTiming=EqualToInput
S3 (Sink@Data Flow Models)
StartStopOption=Auto
F2 (FIR@Data Flow Models)
Taps=(1x65) [-1.26252e-3,-1.05553e-3]
Decimation=5 [SamplesPerSymbol]
DecimationPhase=2 [DownSamplePhase]
Interpolation=1
Q1 (Quantizer@Data Flow Models)
Thresholds=(1x1) [0] [[0]]
Levels=(1x2) [-1,1] [[-1,1]]
BitsOut (Sink@Data Flow Models)
StartStopOption=Samples
CrossCorrOut (Sink@Data Flow Models)
StartStopOption=Samples
P1 (PattMatch@Data Flow Models)
TempSize=25 [TrainingSeqLen]
WinSize=250 [BlockSize]
DelayOut (Sink@Data Flow Models)
StartStopOption=Samples
CrossCorrOutM (Sink@Data Flow Models)
StartStopOption=Samples
Direct
WindowSize=250 [BlockSize]
TemplateSize=25 [TrainingSeqLen]
DelayOutM (Sink@Data Flow Models)
StartStopOption=Samples

12. Activate the Sinks *CrossCorrOutM* and *DelayOutM* and run *Design3 Analysis*. Look at *Delay_Table3*. The values recorded in the *DelayOut* and *DelayOutM* Sinks should match. Look at *CrossCorr_Graph3*. The waveforms *CrossCorrOut* and *CrossCorrOutM* should also match. Play with a few different values of *Delay* in *InEqns3* and verify that the detected delays in *Delay_Table3* match and are equal to the expected delay of 8 + floor( *Delay* / 8 ). You may also want to uncheck the Repeatable Random Sequences checkbox in the *Options* tab of *Design3 Analysis*, run a few simulations and observe how the cross-correlation waveforms in *CrossCorr_Graph3* change with every simulation but always agree with each other.

13. The execution time for the above simulation is on the order of 2 sec. Let's try re-writing the MATLAB_Script code in vectorized form (removing the for loop).

CAUTION The simulation times reported here may vary based on the CPU performance of your machine.

14. Make a copy of the *Direct* MATLAB_Script part, call it *Vectorized* and connect it as shown below. Deactivate (open) the *Direct* MATLAB_Script part.


123
BitsOut {Sink@Data Flow Models}
StartStopOption=Samples
Q1 {Quantizer@Data Flow Models}
Thresholds=(1x1) [0] [[0]]
Levels=(1x2) [-1,1] [[-1,1]]
CrossCorrOut {Sink@Data Flow Models}
StartStopOption=Samples
P1 {PattMatch@Data Flow Models}
TempSize=25 [TrainingSeqLen]
WinSize=250 [BlockSize]
DelayOut {Sink@Data Flow Models}
StartStopOption=Samples
CrossCorrOutM {Sink@Data Flow Models}
StartStopOption=Samples
Direct
Disabled: OPEN
WindowSize=250 [BlockSize]
TemplateSize=25 [TrainingSeqLen]
DelayOutM {Sink@Data Flow Models}
StartStopOption=Samples
Vectorized
WindowSize=250 [BlockSize]
TemplateSize=25 [TrainingSeqLen]

15. The sums for the numerator and denominator look very much like the dot product of two vectors so they can be efficiently computed using vector math, eliminating the inner for loop. The code might look like this:

```
CrossCorrOut = zeros(WindowSize-TemplateSize+1,1);

for n=0:WindowSize-TemplateSize

    WindowBlock = WindowIn(n+1:n+TemplateSize).';
    Numerator = WindowBlock*TemplateIn;
    Denominator = WindowBlock*WindowBlock.';

    if ( Denominator > 0.0 )
        CrossCorrOut(n+1) = Numerator / Denominator;
    else
        CrossCorrOut(n+1) = 0.0;
    end
end

DelayOut = find( CrossCorrOut == max(CrossCorrOut),
1 ) - 1;
```

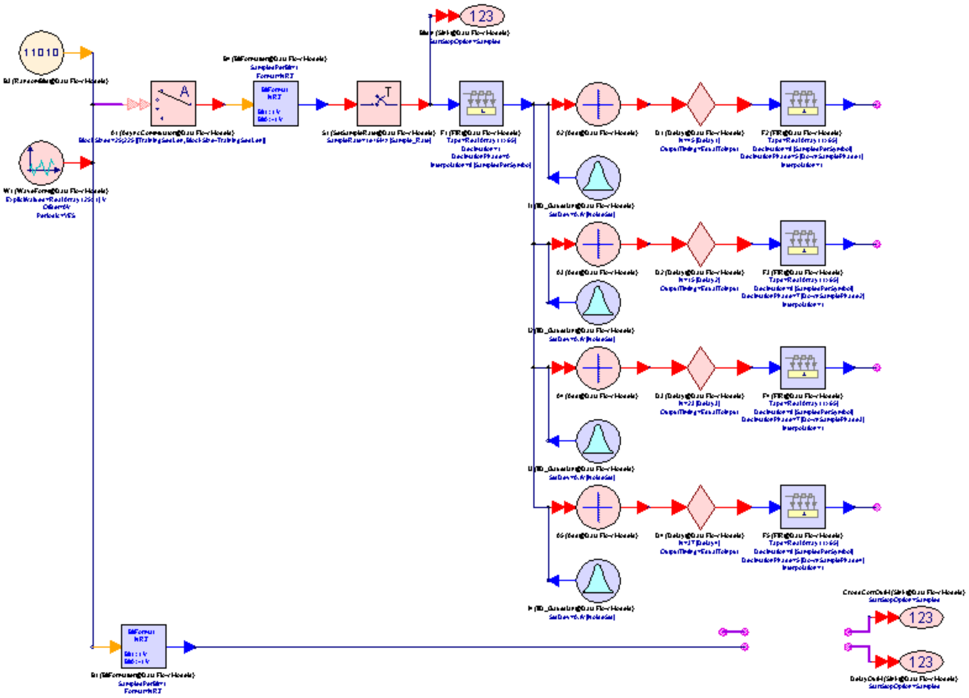
The *CrossCorrOut* variable is first initialized to an all zero array with the correct dimensions *(WindowSize – TemplateSize + 1)* x 1 as in the *Direct* implementation. Then for every possible shift *n* (0 to *WindowSize – TemplateSize*) we first extract the *TemplateSize*-element-long portion of the *WindowIn* array that will overlap with the *TemplateIn* and we transpose it (.' operator). The *WindowIn* and *TemplateIn* arrays are both column vectors (*N* x 1) but for the dot product to work we need to multiply a row vector (1 x *N*) by a column vector (*N* x 1). This is why we transpose the *WindowBlock* extracted subvector. Then we use the regular array MATLAB_Script multiplication to compute the *Numerator* (*Denominator*) as dot product of the vectors *WindowBlock* and *TemplateIn* (*WindowBlock* and *WindowBlock* transposed). The rest of the code is the same as before.

16. Run *Design3 Analysis* and verify that the new MATLAB_Script model gives the same results as the built-in SystemVue PattMatch one. The simulation time is still on the order of 2 sec. You may have to run the simulation much longer (set variable *N* in *InEqns* to a bigger number) to see the speedup benefits of writing code in vectorized form.

## Multirate Model with Bus IO

In this tutorial example, we extend the model created in Multirate Model to process a set of *WindowIn* input signals.

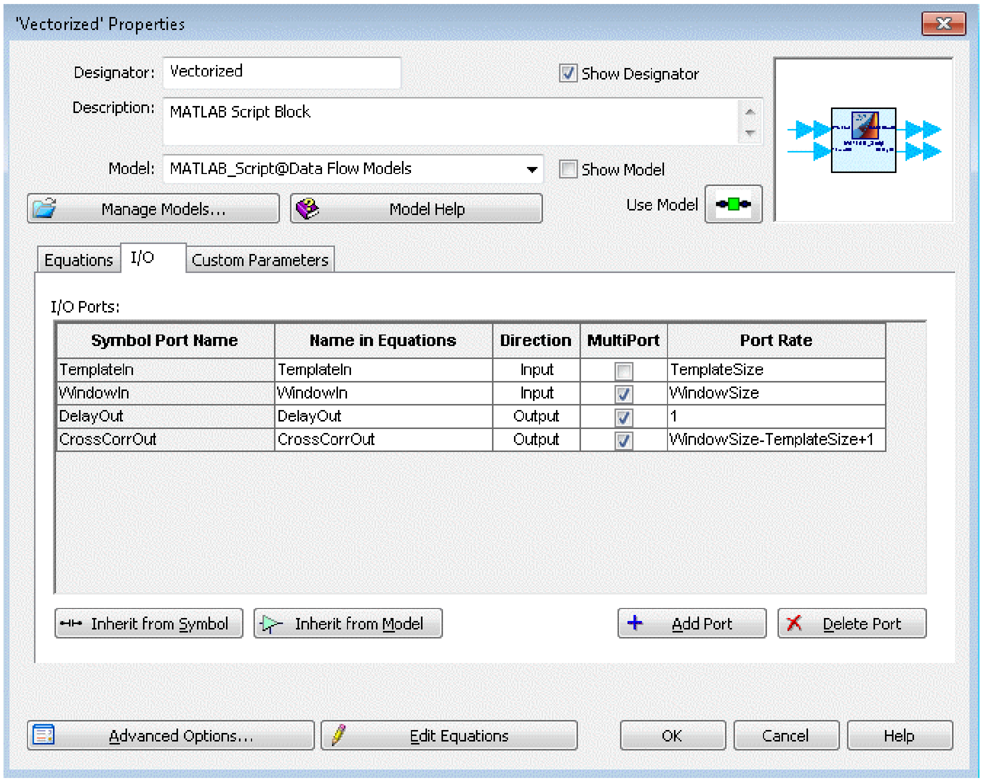
1. Open the workspace <SystemVue Installation Directory>\Examples\Tutorials\Algorithm_Design\MATLAB_Script\MATLAB_Sc wsv.
2. Open *Design4* in the folder *4. Multirate Model with Bus IO*.



3. The system in *Design4* is very similar to the system in *Design3* (see Multirate Model for a detailed description) but instead of having one path (noise addition and signal delay) between the "transmitter" (the interpolating FIR filter *F1*) and the "receiver" (the decimating FIR filter *F2*), the transmitted signal goes through four paths each of which has its own delay (settable in the Equations page *InEqns4*). If we wanted to detect the delay for each path using the PattMatch model we would need four instances of it since this model has single port (not multi-port) inputs. In contrast, the MATLAB_Script part can be configured to have multi-port inputs/outputs so that it can

process multiple signals connected at one input. This is the reason the four signals at the outputs of the decimating FIR filters have been combined into a bus of size four.

4. Copy the *Vectorized* MATLAB_Script part created in the Multirate Model tutorial and paste it in *Design4*. Connect it in the "place holder" area at the bottom right corner of the design (its outputs should be connected to the Sinks *DelayOutM* and *CrossCorrM*).
5. Double click on it and go to the *I/O* tab. Check the checkboxes in the MultiPort column for the *WindowIn*, *DelayOut*, and *CrossCorrOut* ports.



Designator: Vectorized

☑ Show Designator

Description: MATLAB Script Block

Model: MATLAB_Script@Data Flow Models

☐ Show Model

I/O Ports:

| Symbol Port Name | Name in Equations | Direction | MultiPort | Port Rate |
|---|---|---|---|---|
| TemplateIn | TemplateIn | Input | ☐ | TemplateSize |
| WindowIn | WindowIn | Input | ☑ | WindowSize |
| DelayOut | DelayOut | Output | ☑ | 1 |
| CrossCorrOut | CrossCorrOut | Output | ☑ | WindowSize-TemplateSize+1 |

When an MATLAB_Script part port is defined to be a multi-port

- if it is an input port, the data collected from the port will be placed in a cell array with as many top level cells as the size of (number of signals connected to) the multi-port
    - if the port is unirate, then each top level cell will contain a single input sample (could be scalar or matrix depending on the data type of the input signal)
    - if the port is multirate, then each top level cell will contain an *N* x 1 array of input samples, where *N* is the port rate (again each input sample could be scalar or matrix depending on the data type of the input signal)
- if it is an output port, the MATLAB_Script code you write that produces the data for the port needs to place the data in a cell array with as many top level cells as the size of (number of signals connected to) the multi-port

    - if the port is unirate, then each top level cell should contain a single output sample (could be scalar or matrix depending on the data type of the input signal)
    - if the port is multirate, then each top level cell should contain an *N* x 1 array of output samples, where *N* is the port rate (again each input sample could be scalar or matrix depending on the data type of the input signal)

For the example we are working on, we should expect that *TemplateIn* will be a *TemplateSize* x 1 real array (*TemplateIn* has not been defined to be a multi-port) and *WindowIn* will be a cell array with four cells, each of which will be a *WindowSize* x 1 real array. The MATLAB_Script code we write should produce a cell array of four cells (each cell containing a single value) for the *DelayOut* variable (port) and a cell array of four cells (each cell containing a *(WindowSize – TemplateSize + 1)* x 1 array) for the *CrossCorrOut* variable (port).

6. Go to the *Equations* tab and update the code to handle the multi-port input /output signals. Remember that accessing the individual cells of a cell array in MATLAB_Script can be done using curly braces {} and that indices for cell arrays also start at 1. The code might look like this:

```
CellArraySize = size( WindowIn );
NumChannels = CellArraySize(1);

for Channel = 1:NumChannels

    CrossCorrOut{Channel} = zeros(WindowSize-
TemplateSize+1,1);

    for i=0:WindowSize-TemplateSize

        WindowBlock = WindowIn{Channel}(i+1:
i+TemplateSize).';
        Numerator = WindowBlock*TemplateIn;
        Denominator = WindowBlock*WindowBlock.';

        if ( Denominator > 0.0 )
            CrossCorrOut{Channel}(i+1) = Numerator
/ Denominator;
        else
            CrossCorrOut{Channel}(i+1) = 0.0;
        end
    end

    DelayOut{Channel} = find( CrossCorrOut{Channel}
== max(CrossCorrOut{Channel}), 1 ) - 1;
end
```

This code is almost identical to the one in the *Vectorized* implementation in the tutorial Multirate Model with the addition of an outer for loop to process each signal in the multi-port and the use of the {} to access the individual cells in a cell array.

**CAUTION** **Unfortunately, there is no way to *vectorize* operations on cell arrays so there is no way to eliminate the outer for loop that iterates over the individual cells of the cell arrays.**

7. Press the OK button to apply all the changes you made.
8. Run *Design4 Analysis*. Look at *Delay_Table4* and verify that values listed match the expected delays for each path 8 + floor( $Delay_i$ / 8 ), where $Delay_i$ is the delay in the i-th path (set in *InEqns4*).

## Model with Array IO

In this tutorial example, you will create simple MATLAB_Script models that process arrays. With the MATLAB_Script model one can change the rate of the incoming stream of arrays (e.g. produce *N* arrays at the output for each *M* arrays read), change the number of array dimensions, or keep the same number of dimensions but increase/decrease their size. The format of this tutorial is slightly different than the previous ones: the focus is not in trying to write efficient MATLAB_Script code (although one of the examples in this section is a case where efficiency is discussed) but in trying to understand how arrays are processed, how multirate affects the input and output arrays, etc.

The important thing to remember is that when a port is defined to have a rate *R* greater than 1 then

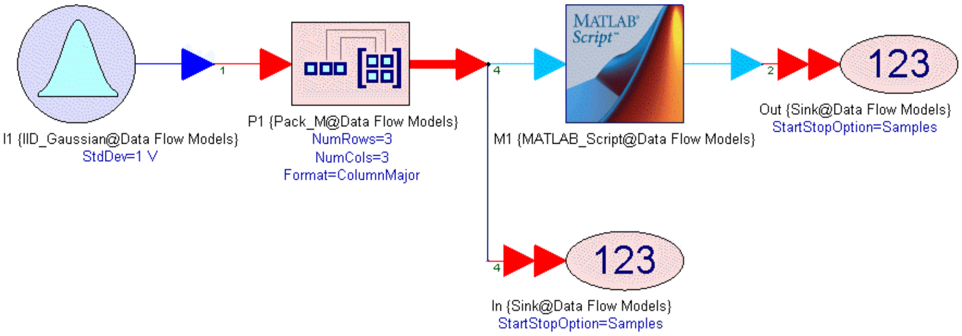
- for an input port the MATLAB_Script model will collect *R* samples from its input and put them in an array whose first dimension is *R*; if the input samples happen to be arrays then a multi-dimensional array is created, e.g. if the input is an *M* x *N* array then the MATLAB_Script code will see an *R* x *M* x *N* array.
- for an output port the MATLAB_Script model expects to receive an array whose first dimension is *R*; if the desired output is an array then the MATLAB_Script code should create a multi-dimensional array for this output, e.g. if the desired output is an *M* x *N* array then the MATLAB_Script code should create an *R* x *M* x *N* array (if the first dimension of the output array does not match the rate *R* truncation or zero-padding will occur).

All the examples shown below are located in the workspace <SystemVue Installation Directory>\Examples\Tutorials\Algorithm_Design\MATLAB_Script\MATLAB_Script_M wsv inside the workspace folder *5. Model with Array IO*. All the examples in this folder use MATLAB_Script parts with one input and one output. This is not a limitation; it is only done for simplicity so that the basic concepts can be better understood. You can have MATLAB_Script parts processing arrays with multiple inputs and/or outputs each of which can have its own rate.

**CAUTION** When you run these examples you will see different values than the ones are shown in the tables in this section. The reason is that the source is a random Gaussian one and the **Repeatable Random Sequences** checkbox in the Options tab of the Data Flow Analysis is not checked.

## Unirate Example

1. Open design *1. Unirate*.


I1 {IID_Gaussian@Data Flow Models}
StdDev=1 V
1
P1 {Pack_M@Data Flow Models}
NumRows=3
NumCols=3
Format=ColumnMajor
4
MATLAB Script
M1 {MATLAB_Script@Data Flow Models}
2
123
Out {Sink@Data Flow Models}
StartStopOption=Samples
4
123
In {Sink@Data Flow Models}
StartStopOption=Samples

2. In this design, we create a 3 x 3 array of random normally distributed numbers and pass it to an MATLAB_Script part. The MATLAB_Script part is configured with one input and one output both of which have a rate of 1 (this is the default configuration). The MATLAB_Script code is the single line

```
output = input(1:2:end,end:-2:1);
```

Can you figure out what this code will produce at the output?

Since the input rate is 1, the input is going to be a 3 x 3 array. input(1:2:end, end:-2:1) takes a subset of the rows and columns of the array. More specifically, the *range* 1:2:end (start at 1, use a step of 2, and finish at the last available index (*end*) of the associated dimension) will evaluate (when associated with the first dimension of a 3 x 3 array) to [1, 3] and the *range* end:-2:1 (start at the last available index (*end*) of the associated dimension, use a step of -2, and finish at 1) will evaluate (when associated with the first dimension of a 3 x 3 array) to [3, 1]. The length of both ranges is 2, so the resulting sub-array (which is going to be assigned to output) is a 2 x 2 array. So we are picking the array elements of rows 1 and 3 and columns 1 and 3 from the original array and assigning them to the output array. In addition, since the column *range* goes backwards ([3, 1] instead of [1, 3]) the elements of column 3 are going to be assigned to column 1 in the output array and the elements of column 1 are going to be assigned to column 2 in the output array. Let's verify this by looking at *Table1*.

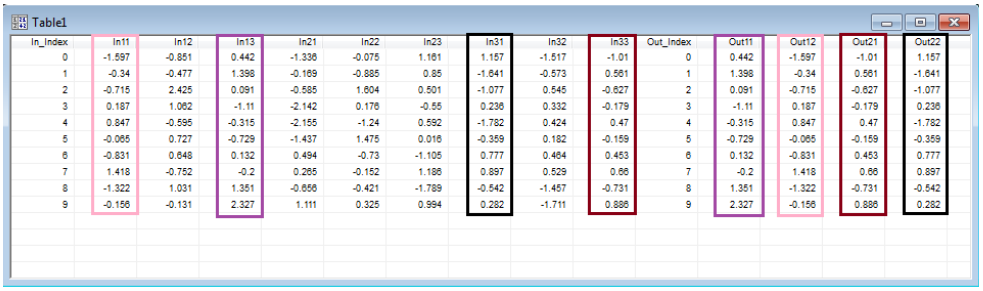


Table1

| In_Index | In11 | In12 | In13 | In21 | In22 | In23 | In31 | In32 | In33 | Out_Index | Out11 | Out12 | Out21 | Out22 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | -1.597 | -0.851 | 0.442 | -1.336 | -0.075 | 1.161 | 1.157 | -1.517 | -1.01 | 0 | 0.442 | -1.597 | -1.01 | 1.157 |
| 1 | -0.34 | -0.477 | 1.398 | -0.169 | -0.885 | 0.85 | -1.641 | -0.573 | 0.561 | 1 | 1.398 | -0.34 | 0.561 | -1.641 |
| 2 | -0.715 | 2.425 | 0.091 | -0.585 | 1.604 | 0.501 | -1.077 | 0.545 | -0.627 | 2 | 0.091 | -0.715 | -0.627 | -1.077 |
| 3 | 0.187 | 1.062 | -1.11 | -2.142 | 0.176 | -0.55 | 0.236 | 0.332 | -0.179 | 3 | -1.11 | 0.187 | -0.179 | 0.236 |
| 4 | 0.847 | -0.595 | -0.315 | -2.155 | -1.24 | 0.592 | -1.782 | 0.424 | 0.47 | 4 | -0.315 | 0.847 | 0.47 | -1.782 |
| 5 | -0.065 | 0.727 | -0.729 | -1.437 | 1.475 | 0.016 | -0.359 | 0.182 | -0.159 | 5 | -0.729 | -0.065 | -0.159 | -0.359 |
| 6 | -0.831 | 0.648 | 0.132 | 0.494 | -0.73 | -1.105 | 0.777 | 0.464 | 0.453 | 6 | 0.132 | -0.831 | 0.453 | 0.777 |
| 7 | 1.418 | -0.752 | -0.2 | 0.265 | -0.152 | 1.186 | 0.897 | 0.529 | 0.66 | 7 | -0.2 | 1.418 | 0.66 | 0.897 |
| 8 | -1.322 | 1.031 | 1.351 | -0.656 | -0.421 | -1.789 | -0.542 | -1.457 | -0.731 | 8 | 1.351 | -1.322 | -0.731 | -0.542 |
| 9 | -0.156 | -0.131 | 2.327 | 1.111 | 0.325 | 0.994 | 0.282 | -1.711 | 0.886 | 9 | 2.327 | -0.156 | 0.886 | 0.282 |

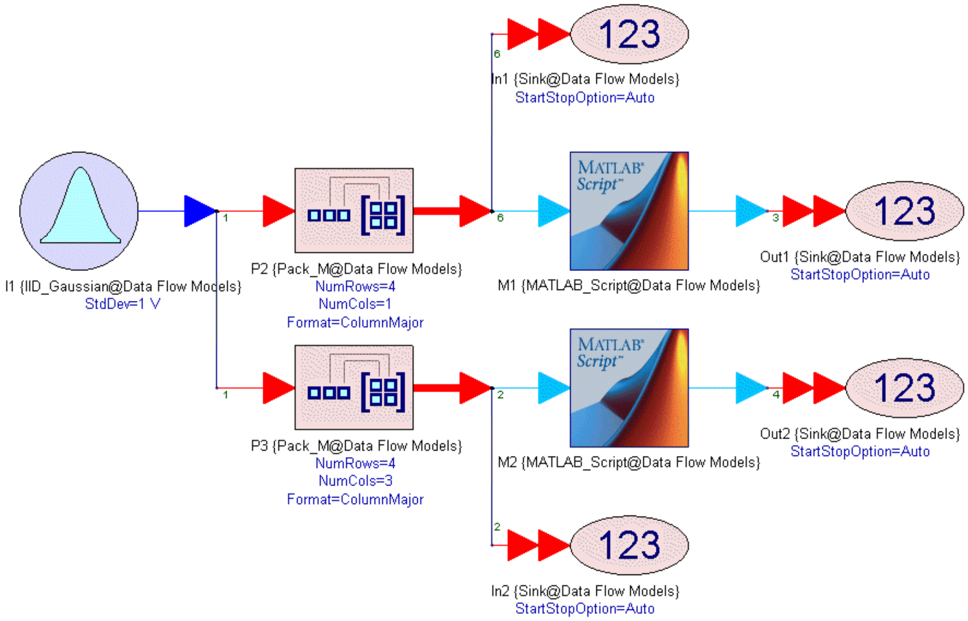
Indeed,

- the ( 1 , 1 ) element in the output array is the ( 1 , 3 ) element in the input array ( 1 is element 1 in the [ 1 , 3] array evaluated from the row *range* 1:2:end and 3 is element 1 in the [ 3 , 1] array evaluated from the column *range* end:-2:1) (see columns marked in purple in *Table1*)
- the ( 1 , 2 ) element in the output array is the ( 1 , 1 ) element in the input array ( 1 is element 1 in the [ 1 , 3] array evaluated from the row *range* 1:2:end and 1 is element 2 in the [3, 1 ] array evaluated from the column *range* end:-2:1) (see columns marked in pink in *Table1*)
- the ( 2 , 1 ) element in the output array is the ( 3 , 3 ) element in the input array ( 3 is element 2 in the [1, 3 ] array evaluated from the row *range* 1:2:end and 3 is element 1 in the [ 3 , 1] array evaluated from the column *range* end:-2:1) (see columns marked in brown in *Table1*)
- the ( 2 , 2 ) element in the output array is the ( 3 , 1 ) element in the input array ( 3 is element 2 in the [1, 3 ] array evaluated from the row *range* 1:2:end and 1 is element 2 in the [3, 1 ] array evaluated from the column *range* end:-2:1) (see columns marked in black in *Table1*)

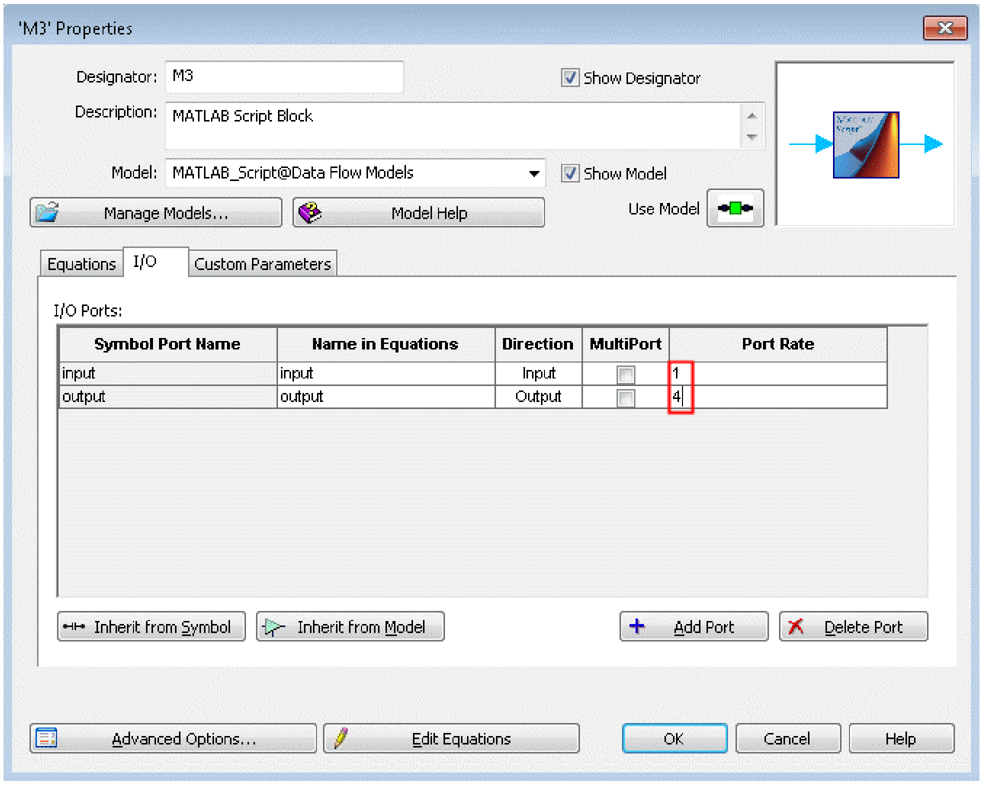
3. In this example,
   - the input rate stayed the same (there is one output array generated for each input array processed)
   - the output array has the same number of dimensions as the input array
   - the output array dimensions (2 x 2) are smaller than the input ones (3 x 3)

## Multirate Output Example

1. Open design *2. Multirate Output*.


123
In1 {Sink@Data Flow Models}
StartStopOption=Auto
MATLAB Script
123
Out1 {Sink@Data Flow Models}
StartStopOption=Auto
I1 {IID_Gaussian@Data Flow Models}
StdDev=1 V
P2 {Pack_M@Data Flow Models}
NumRows=4
NumCols=1
Format=ColumnMajor
M1 {MATLAB_Script@Data Flow Models}
MATLAB Script
123
P3 {Pack_M@Data Flow Models}
NumRows=4
NumCols=3
Format=ColumnMajor
M2 {MATLAB_Script@Data Flow Models}
Out2 {Sink@Data Flow Models}
StartStopOption=Auto
123
In2 {Sink@Data Flow Models}
StartStopOption=Auto

2. In this design, we create two arrays, a 4 x 1 and a 4 x 3, of random normally distributed numbers and pass them to two MATLAB_Script parts. The MATLAB_Script parts are identical (one input with rate 1 and one output with rate 4).


'M3' Properties

Designator: M3 ☑ Show Designator

Description: MATLAB Script Block

Model: MATLAB_Script@Data Flow Models ☑ Show Model

Manage Models... | Model Help | Use Model

Equations | I/O | Custom Parameters

I/O Ports:

| Symbol Port Name | Name in Equations | Direction | MultiPort | Port Rate |
|---|---|---|---|---|
| input | input | Input | □ | 1 |
| output | output | Output | □ | 4 |

Inherit from Symbol | Inherit from Model | Add Port | Delete Port

Advanced Options... | Edit Equations | OK | Cancel | Help

The MATLAB_Script code is the single line

```
output = input;
```

Can you figure out what this code will produce at the output?

For the MATLAB_Script part with the 4 x 1 array input, since the input rate is 1, the input is going to be a 4 x 1 array. This is assigned to output, which has a rate of 4. Remember that the first dimension of an array corresponding to an output port is associated with the port's rate, that is, how many output samples will be produced at each execution on the MATLAB_Script model. In this case, if we "remove" the first dimension we end up with a scalar. So the output of this MATLAB_Script part is going to be scalar values, which come out at 4 times the rate the input 4 x 1 arrays are consumed. Let's verify this by looking at *Table2_1*.

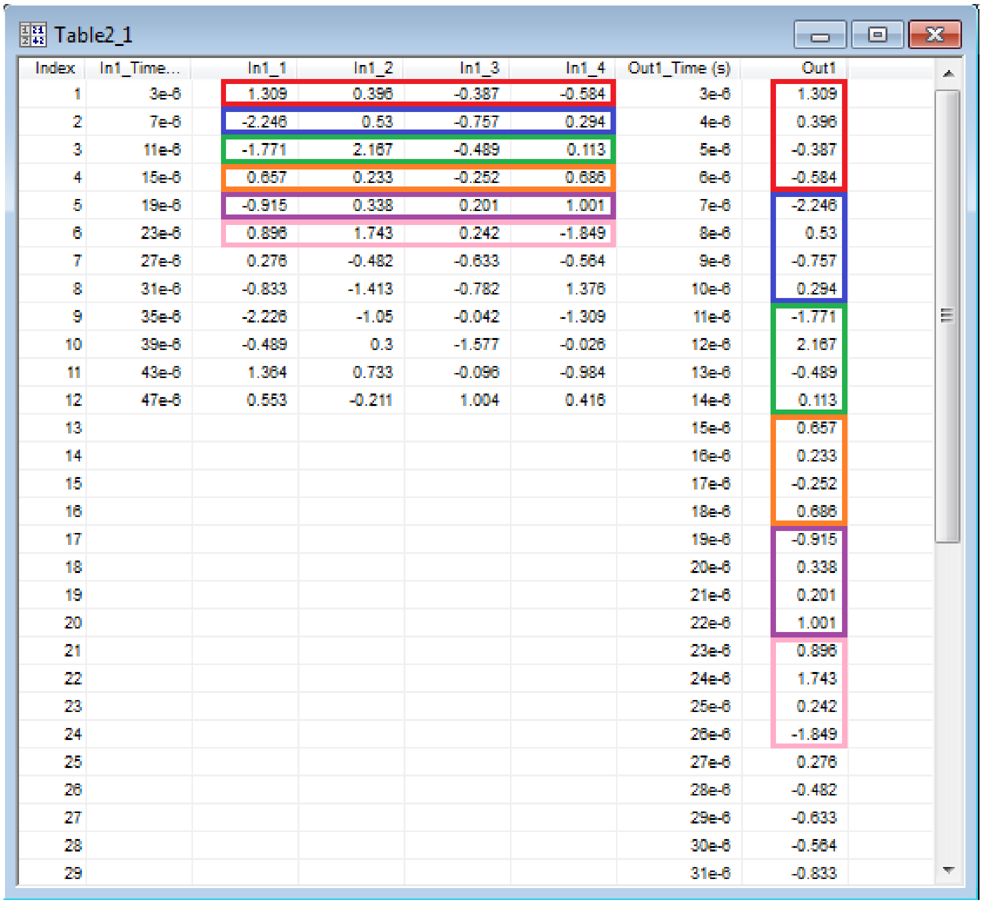


Table2_1

| Index | In1_Time... | In1_1 | In1_2 | In1_3 | In1_4 | Out1_Time (s) | Out1 |
|---|---|---|---|---|---|---|---|
| 1 | 3e-6 | 1.309 | 0.396 | -0.387 | -0.584 | 3e-6 | 1.309 |
| 2 | 7e-6 | -2.246 | 0.53 | -0.757 | 0.294 | 4e-6 | 0.396 |
| 3 | 11e-6 | -1.771 | 2.167 | -0.489 | 0.113 | 5e-6 | -0.387 |
| 4 | 15e-6 | 0.657 | 0.233 | -0.252 | 0.686 | 6e-6 | -0.584 |
| 5 | 19e-6 | -0.915 | 0.338 | 0.201 | 1.001 | 7e-6 | -2.246 |
| 6 | 23e-6 | 0.896 | 1.743 | 0.242 | -1.849 | 8e-6 | 0.53 |
| 7 | 27e-6 | 0.276 | -0.482 | -0.633 | -0.564 | 9e-6 | -0.757 |
| 8 | 31e-6 | -0.833 | -1.413 | -0.782 | 1.376 | 10e-6 | 0.294 |
| 9 | 35e-6 | -2.226 | -1.05 | -0.042 | -1.309 | 11e-6 | -1.771 |
| 10 | 39e-6 | -0.489 | 0.3 | -1.577 | -0.026 | 12e-6 | 2.167 |
| 11 | 43e-6 | 1.364 | 0.733 | -0.096 | -0.984 | 13e-6 | -0.489 |
| 12 | 47e-6 | 0.553 | -0.211 | 1.004 | 0.416 | 14e-6 | 0.113 |
| 13 | | | | | | 15e-6 | 0.657 |
| 14 | | | | | | 16e-6 | 0.233 |
| 15 | | | | | | 17e-6 | -0.252 |
| 16 | | | | | | 18e-6 | 0.686 |
| 17 | | | | | | 19e-6 | -0.915 |
| 18 | | | | | | 20e-6 | 0.338 |
| 19 | | | | | | 21e-6 | 0.201 |
| 20 | | | | | | 22e-6 | 1.001 |
| 21 | | | | | | 23e-6 | 0.896 |
| 22 | | | | | | 24e-6 | 1.743 |
| 23 | | | | | | 25e-6 | 0.242 |
| 24 | | | | | | 26e-6 | -1.849 |
| 25 | | | | | | 27e-6 | 0.276 |
| 26 | | | | | | 28e-6 | -0.482 |
| 27 | | | | | | 29e-6 | -0.633 |
| 28 | | | | | | 30e-6 | -0.564 |
| 29 | | | | | | 31e-6 | -0.833 |

Similarly, for the MATLAB_Script part with the 4 x 3 array input, since the input rate is 1, the input is going to be a 4 x 1 array. This is assigned to output, which has a rate of 4. Again, if we "remove" the first dimension we end up with a one-dimensional vector of size 3. So the output of this MATLAB_Script part is going to be dimensional vectors of size 3, which come out at 4 times the rate the input 4 x 3 arrays are consumed. Let's verify this by looking at *Table2_2*.

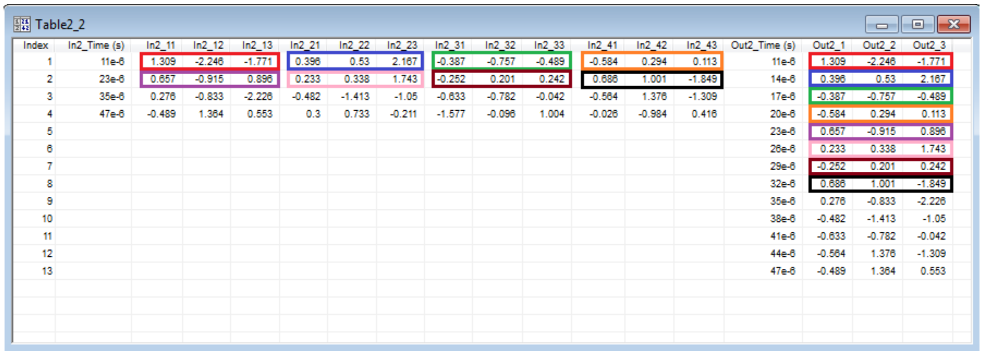


Table2_2

| Index | In2_Time (s) | In2_11 | In2_12 | In2_13 | In2_21 | In2_22 | In2_23 | In2_31 | In2_32 | In2_33 | In2_41 | In2_42 | In2_43 | Out2_Time (s) | Out2_1 | Out2_2 | Out2_3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 11e-6 | 1.309 | -2.246 | -1.771 | 0.396 | 0.53 | 2.167 | -0.387 | -0.757 | -0.489 | -0.584 | 0.294 | 0.113 | 11e-6 | 1.309 | -2.246 | -1.771 |
| 2 | 23e-6 | 0.657 | -0.915 | 0.896 | 0.233 | 0.338 | 1.743 | -0.252 | 0.201 | 0.242 | 0.686 | 1.001 | -1.849 | 14e-6 | 0.396 | 0.53 | 2.167 |
| 3 | 35e-6 | 0.276 | -0.833 | -2.226 | -0.482 | -1.413 | -1.05 | -0.633 | -0.782 | -0.042 | -0.564 | 1.376 | -1.309 | 17e-6 | -0.387 | -0.757 | -0.489 |
| 4 | 47e-6 | -0.489 | 1.364 | 0.553 | 0.3 | 0.733 | -0.211 | -1.577 | -0.096 | 1.004 | -0.026 | -0.984 | 0.416 | 20e-6 | -0.584 | 0.294 | 0.113 |
| 5 | | | | | | | | | | | | | | 23e-6 | 0.657 | -0.915 | 0.896 |
| 6 | | | | | | | | | | | | | | 26e-6 | 0.233 | 0.338 | 1.743 |
| 7 | | | | | | | | | | | | | | 29e-6 | -0.252 | 0.201 | 0.242 |
| 8 | | | | | | | | | | | | | | 32e-6 | 0.686 | 1.001 | -1.849 |
| 9 | | | | | | | | | | | | | | 35e-6 | 0.276 | -0.833 | -2.226 |
| 10 | | | | | | | | | | | | | | 38e-6 | -0.482 | -1.413 | -1.05 |
| 11 | | | | | | | | | | | | | | 41e-6 | -0.633 | -0.782 | -0.042 |
| 12 | | | | | | | | | | | | | | 44e-6 | -0.564 | 1.376 | -1.309 |
| 13 | | | | | | | | | | | | | | 47e-6 | -0.489 | 1.364 | 0.553 |

3. In this example,
    - the output rate is 4 times the input rate
    - the output arrays have different number of dimensions compared to the input arrays
4. Now try changing the output port rate from 4 to 6 and run the simulation again. Since the output array's first dimension is still 4, the output will be padded with 2 "zero" samples. For the MATLAB_Script part with the 4 x 1 array input the padded values are scalar zeros. For the MATLAB_Script with the 4 x 3 array input the padded values are 1 x 3 arrays with all their elements set to 0.

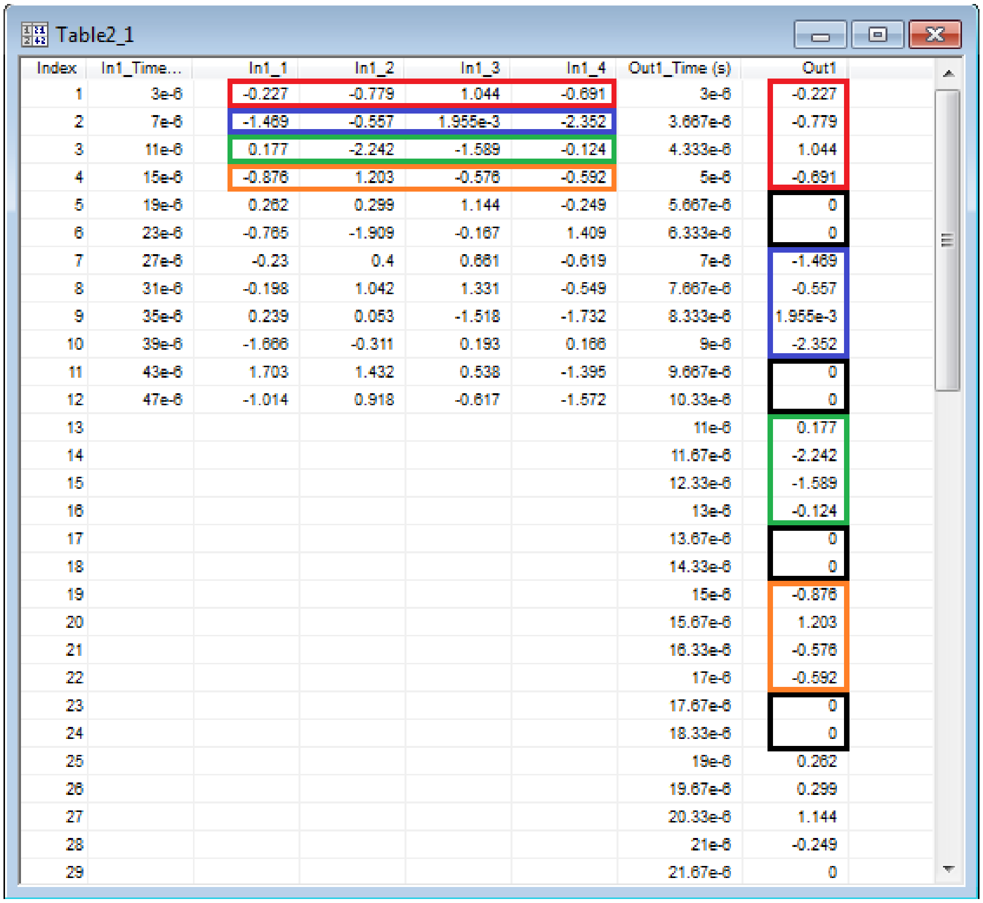


Table2_1

| Index | In1_Time... | In1_1 | In1_2 | In1_3 | In1_4 | Out1_Time (s) | Out1 |
|---|---|---|---|---|---|---|---|
| 1 | 3e-6 | -0.227 | -0.779 | 1.044 | -0.691 | 3e-6 | -0.227 |
| 2 | 7e-6 | -1.469 | -0.557 | 1.955e-3 | -2.352 | 3.667e-6 | -0.779 |
| 3 | 11e-6 | 0.177 | -2.242 | -1.589 | -0.124 | 4.333e-6 | 1.044 |
| 4 | 15e-6 | -0.876 | 1.203 | -0.576 | -0.592 | 5e-6 | -0.691 |
| 5 | 19e-6 | 0.262 | 0.299 | 1.144 | -0.249 | 5.667e-6 | 0 |
| 6 | 23e-6 | -0.765 | -1.909 | -0.167 | 1.409 | 6.333e-6 | 0 |
| 7 | 27e-6 | -0.23 | 0.4 | 0.661 | -0.619 | 7e-6 | -1.469 |
| 8 | 31e-6 | -0.198 | 1.042 | 1.331 | -0.549 | 7.667e-6 | -0.557 |
| 9 | 35e-6 | 0.239 | 0.053 | -1.518 | -1.732 | 8.333e-6 | 1.955e-3 |
| 10 | 39e-6 | -1.666 | -0.311 | 0.193 | 0.166 | 9e-6 | -2.352 |
| 11 | 43e-6 | 1.703 | 1.432 | 0.538 | -1.395 | 9.667e-6 | 0 |
| 12 | 47e-6 | -1.014 | 0.918 | -0.617 | -1.572 | 10.33e-6 | 0 |
| 13 | | | | | | 11e-6 | 0.177 |
| 14 | | | | | | 11.67e-6 | -2.242 |
| 15 | | | | | | 12.33e-6 | -1.589 |
| 16 | | | | | | 13e-6 | -0.124 |
| 17 | | | | | | 13.67e-6 | 0 |
| 18 | | | | | | 14.33e-6 | 0 |
| 19 | | | | | | 15e-6 | -0.876 |
| 20 | | | | | | 15.67e-6 | 1.203 |
| 21 | | | | | | 16.33e-6 | -0.576 |
| 22 | | | | | | 17e-6 | -0.592 |
| 23 | | | | | | 17.67e-6 | 0 |
| 24 | | | | | | 18.33e-6 | 0 |
| 25 | | | | | | 19e-6 | 0.262 |
| 26 | | | | | | 19.67e-6 | 0.299 |
| 27 | | | | | | 20.33e-6 | 1.144 |
| 28 | | | | | | 21e-6 | -0.249 |
| 29 | | | | | | 21.67e-6 | 0 |

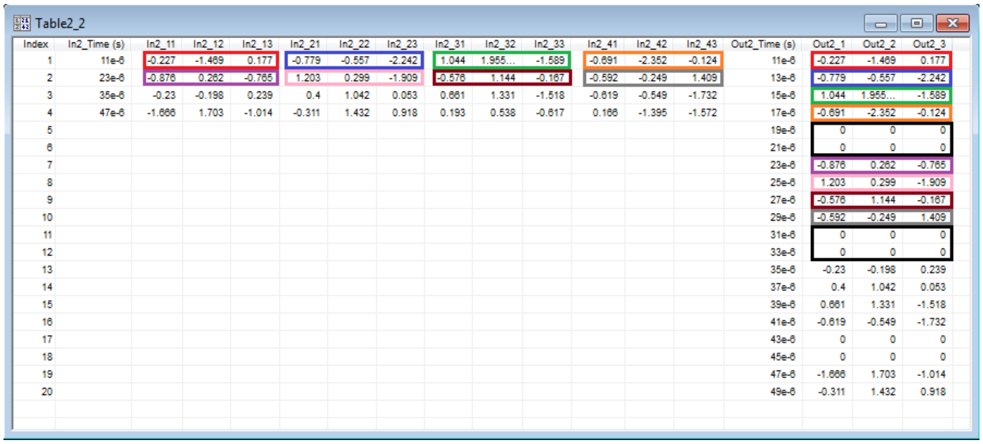


Table2_2

| Index | In2_Time (s) | In2_11 | In2_12 | In2_13 | In2_21 | In2_22 | In2_23 | In2_31 | In2_32 | In2_33 | In2_41 | In2_42 | In2_43 | Out2_Time (s) | Out2_1 | Out2_2 | Out2_3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 11e-6 | -0.227 | -1.469 | 0.177 | -0.779 | -0.557 | -2.242 | 1.044 | 1.955... | -1.589 | -0.691 | -2.352 | -0.124 | 11e-6 | -0.227 | -1.469 | 0.177 |
| 2 | 23e-6 | -0.876 | 0.262 | -0.765 | 1.203 | 0.299 | -1.909 | -0.576 | 1.144 | -0.167 | -0.592 | -0.249 | 1.409 | 13e-6 | -0.779 | -0.557 | -2.242 |
| 3 | 35e-6 | -0.23 | -0.198 | 0.239 | 0.4 | 1.042 | 0.053 | 0.661 | 1.331 | -1.518 | -0.619 | -0.549 | -1.732 | 15e-6 | 1.044 | 1.955... | -1.589 |
| 4 | 47e-6 | -1.666 | 1.703 | -1.014 | -0.311 | 1.432 | 0.918 | 0.193 | 0.538 | -0.617 | 0.166 | -1.395 | -1.572 | 17e-6 | -0.691 | -2.352 | -0.124 |
| 5 | | | | | | | | | | | | | | 19e-6 | 0 | 0 | 0 |
| 6 | | | | | | | | | | | | | | 21e-6 | 0 | 0 | 0 |
| 7 | | | | | | | | | | | | | | 23e-6 | -0.876 | 0.262 | -0.765 |
| 8 | | | | | | | | | | | | | | 25e-6 | 1.203 | 0.299 | -1.909 |
| 9 | | | | | | | | | | | | | | 27e-6 | -0.576 | 1.144 | -0.167 |
| 10 | | | | | | | | | | | | | | 29e-6 | -0.592 | -0.249 | 1.409 |
| 11 | | | | | | | | | | | | | | 31e-6 | 0 | 0 | 0 |
| 12 | | | | | | | | | | | | | | 33e-6 | 0 | 0 | 0 |
| 13 | | | | | | | | | | | | | | 35e-6 | -0.23 | -0.198 | 0.239 |
| 14 | | | | | | | | | | | | | | 37e-6 | 0.4 | 1.042 | 0.053 |
| 15 | | | | | | | | | | | | | | 39e-6 | 0.661 | 1.331 | -1.518 |
| 16 | | | | | | | | | | | | | | 41e-6 | -0.619 | -0.549 | -1.732 |
| 17 | | | | | | | | | | | | | | 43e-6 | 0 | 0 | 0 |
| 18 | | | | | | | | | | | | | | 45e-6 | 0 | 0 | 0 |
| 19 | | | | | | | | | | | | | | 47e-6 | -1.666 | 1.703 | -1.014 |
| 20 | | | | | | | | | | | | | | 49e-6 | -0.311 | 1.432 | 0.918 |

5. Now try changing the output port rate from 4 to 3 and run the simulation again. Since the output array's first dimension is still 4, the last row in output will be omitted.

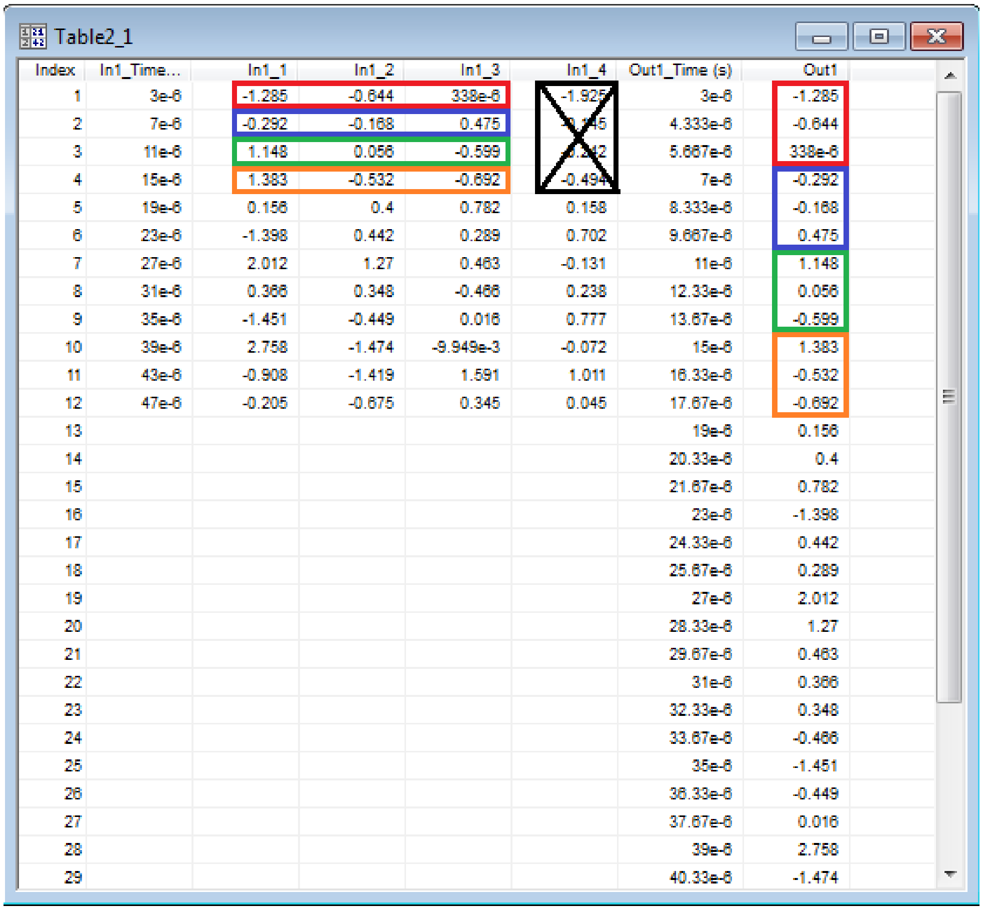


Table2_1

| Index | In1_Time... | In1_1 | In1_2 | In1_3 | In1_4 | Out1_Time (s) | Out1 |
|---|---|---|---|---|---|---|---|
| 1 | 3e-6 | -1.285 | -0.644 | 338e-6 | ~~-1.925~~ | 3e-6 | -1.285 |
| 2 | 7e-6 | -0.292 | -0.168 | 0.475 | ~~-0.145~~ | 4.333e-6 | -0.644 |
| 3 | 11e-6 | 1.148 | 0.056 | -0.599 | ~~-0.242~~ | 5.667e-6 | 338e-6 |
| 4 | 15e-6 | 1.383 | -0.532 | -0.692 | ~~-0.494~~ | 7e-6 | -0.292 |
| 5 | 19e-6 | 0.156 | 0.4 | 0.782 | 0.158 | 8.333e-6 | -0.168 |
| 6 | 23e-6 | -1.398 | 0.442 | 0.289 | 0.702 | 9.667e-6 | 0.475 |
| 7 | 27e-6 | 2.012 | 1.27 | 0.463 | -0.131 | 11e-6 | 1.148 |
| 8 | 31e-6 | 0.366 | 0.348 | -0.466 | 0.238 | 12.33e-6 | 0.056 |
| 9 | 35e-6 | -1.451 | -0.449 | 0.016 | 0.777 | 13.67e-6 | -0.599 |
| 10 | 39e-6 | 2.758 | -1.474 | -9.949e-3 | -0.072 | 15e-6 | 1.383 |
| 11 | 43e-6 | -0.908 | -1.419 | 1.591 | 1.011 | 16.33e-6 | -0.532 |
| 12 | 47e-6 | -0.205 | -0.675 | 0.345 | 0.045 | 17.67e-6 | -0.692 |
| 13 | | | | | | 19e-6 | 0.156 |
| 14 | | | | | | 20.33e-6 | 0.4 |
| 15 | | | | | | 21.67e-6 | 0.782 |
| 16 | | | | | | 23e-6 | -1.398 |
| 17 | | | | | | 24.33e-6 | 0.442 |
| 18 | | | | | | 25.67e-6 | 0.289 |
| 19 | | | | | | 27e-6 | 2.012 |
| 20 | | | | | | 28.33e-6 | 1.27 |
| 21 | | | | | | 29.67e-6 | 0.463 |
| 22 | | | | | | 31e-6 | 0.366 |
| 23 | | | | | | 32.33e-6 | 0.348 |
| 24 | | | | | | 33.67e-6 | -0.466 |
| 25 | | | | | | 35e-6 | -1.451 |
| 26 | | | | | | 36.33e-6 | -0.449 |
| 27 | | | | | | 37.67e-6 | 0.016 |
| 28 | | | | | | 39e-6 | 2.758 |
| 29 | | | | | | 40.33e-6 | -1.474 |

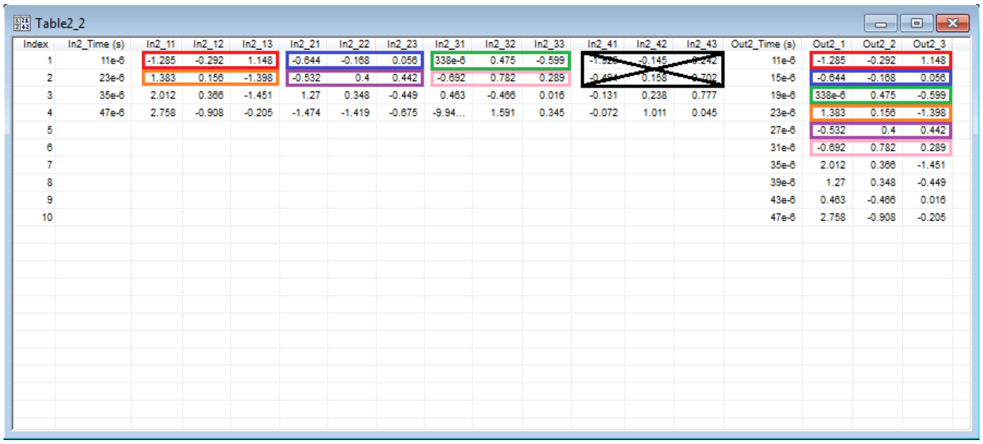
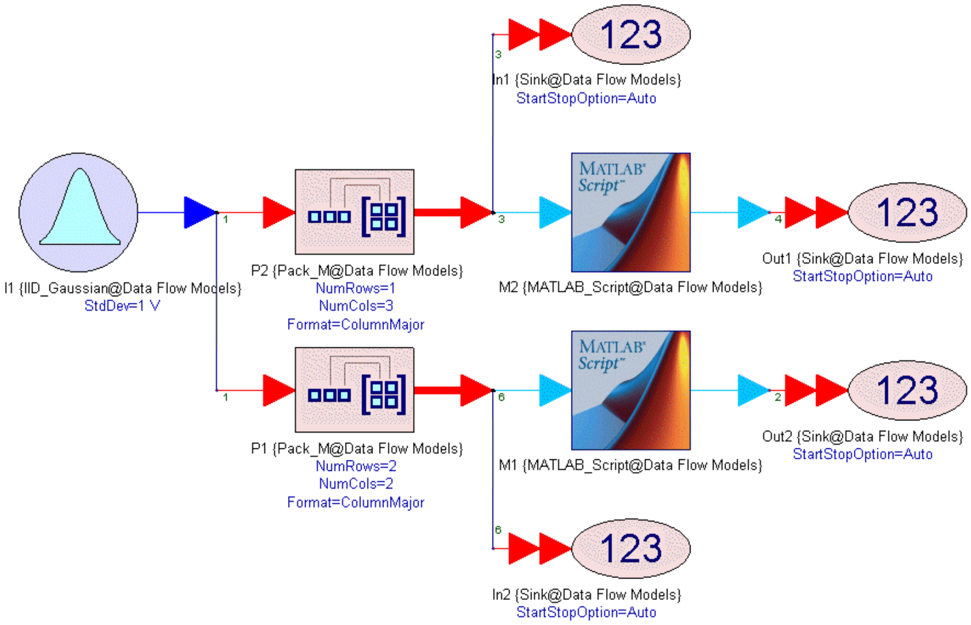


Table2_2

| Index | In2_Time (s) | In2_11 | In2_12 | In2_13 | In2_21 | In2_22 | In2_23 | In2_31 | In2_32 | In2_33 | In2_41 | In2_42 | In2_43 | Out2_Time (s) | Out2_1 | Out2_2 | Out2_3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 11e-6 | -1.285 | -0.292 | 1.148 | -0.644 | -0.168 | 0.056 | 338e-6 | 0.475 | -0.599 | ~~-1.326~~ | ~~-0.145~~ | ~~0.242~~ | 11e-6 | -1.285 | -0.292 | 1.148 |
| 2 | 23e-6 | 1.383 | 0.156 | -1.398 | -0.532 | 0.4 | 0.442 | -0.692 | 0.782 | 0.289 | ~~-0.494~~ | ~~0.158~~ | ~~0.702~~ | 15e-6 | -0.644 | -0.168 | 0.056 |
| 3 | 35e-6 | 2.012 | 0.366 | -1.451 | 1.27 | 0.348 | -0.449 | 0.463 | -0.466 | 0.016 | -0.131 | 0.238 | 0.777 | 19e-6 | 338e-6 | 0.475 | -0.599 |
| 4 | 47e-6 | 2.758 | -0.908 | -0.205 | -1.474 | -1.419 | -0.675 | -9.94... | 1.591 | 0.345 | -0.072 | 1.011 | 0.045 | 23e-6 | 1.383 | 0.156 | -1.398 |
| 5 | | | | | | | | | | | | | | 27e-6 | -0.532 | 0.4 | 0.442 |
| 6 | | | | | | | | | | | | | | 31e-6 | -0.692 | 0.782 | 0.289 |
| 7 | | | | | | | | | | | | | | 35e-6 | 2.012 | 0.366 | -1.451 |
| 8 | | | | | | | | | | | | | | 39e-6 | 1.27 | 0.348 | -0.449 |
| 9 | | | | | | | | | | | | | | 43e-6 | 0.463 | -0.466 | 0.016 |
| 10 | | | | | | | | | | | | | | 47e-6 | 2.758 | -0.908 | -0.205 |

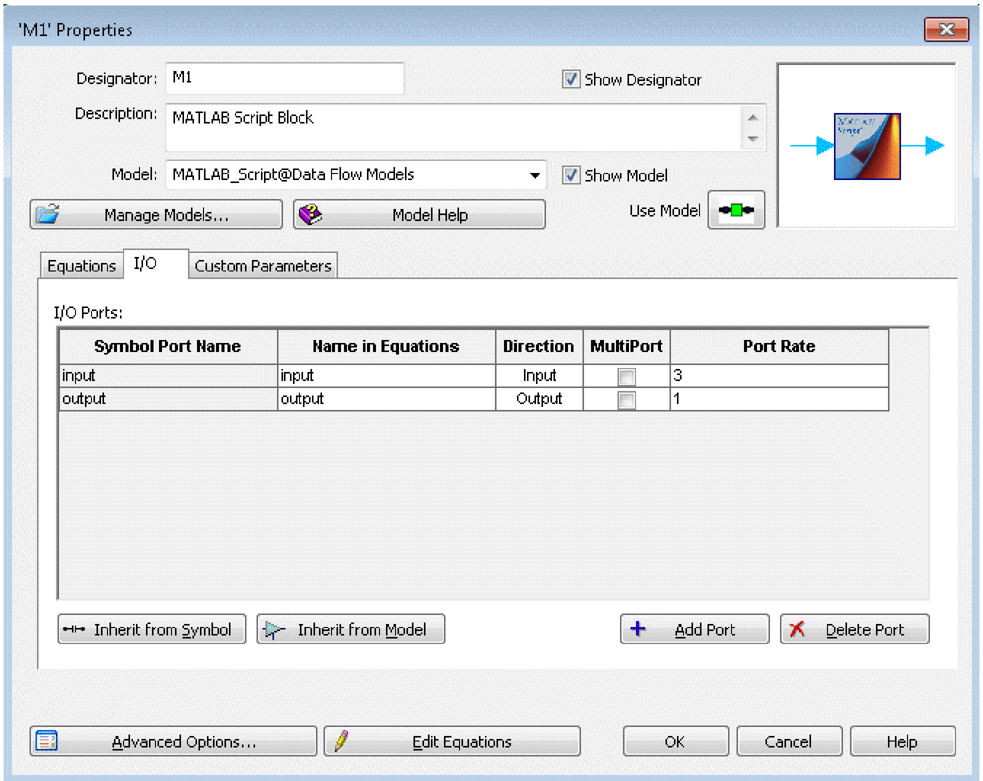
## Multirate Input Example

1. Open design *3. Multirate Input*.



123
3
In1 {Sink@Data Flow Models}
StartStopOption=Auto
1
3
MATLAB Script
4
123
I1 {IID_Gaussian@Data Flow Models}
StdDev=1 V
P2 {Pack_M@Data Flow Models}
NumRows=1
NumCols=3
Format=ColumnMajor
M2 {MATLAB_Script@Data Flow Models}
Out1 {Sink@Data Flow Models}
StartStopOption=Auto
1
6
MATLAB Script
2
123
P1 {Pack_M@Data Flow Models}
NumRows=2
NumCols=2
Format=ColumnMajor
M1 {MATLAB_Script@Data Flow Models}
Out2 {Sink@Data Flow Models}
StartStopOption=Auto
6
123
In2 {Sink@Data Flow Models}
StartStopOption=Auto

2. In this design, we create two arrays, a 1 x 3 and a 2 x 2, of random normally distributed numbers and pass them to two MATLAB_Script parts. The MATLAB_Script parts are identical (one input with rate 3 and one output with rate 1).



Designator: M1 ☑ Show Designator

Description: MATLAB Script Block

Model: MATLAB_Script@Data Flow Models ☑ Show Model

I/O Ports:

| Symbol Port Name | Name in Equations | Direction | MultiPort | Port Rate |
|---|---|---|---|---|
| input | input | Input | □ | 3 |
| output | output | Output | □ | 1 |

The MATLAB_Script code is the single line

```
output = input;
```

Can you figure out what this code will produce at the output?

For the MATLAB_Script part with the 1 x 3 array input, since the input rate is 3, the input is going to be a 3 x 1 x 3 array. This is assigned to output, which has a rate of 1. Therefore, the output of this MATLAB_Script part is going to be 3 x 1 x 3 arrays, which come out at 1/3 times the rate the input 1 x 3 arrays are consumed. Let's verify this by looking at *Table3_1*.

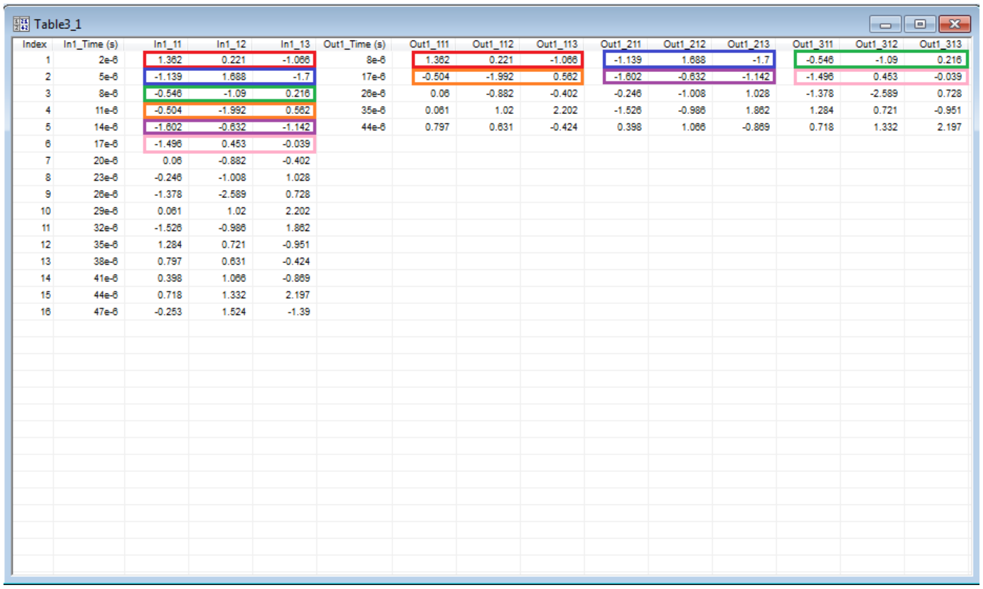


Table3_1

| Index | In1_Time (s) | In1_11 | In1_12 | In1_13 | Out1_Time (s) | Out1_111 | Out1_112 | Out1_113 | Out1_211 | Out1_212 | Out1_213 | Out1_311 | Out1_312 | Out1_313 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2e-6 | 1.362 | 0.221 | -1.066 | 8e-6 | 1.362 | 0.221 | -1.066 | -1.139 | 1.688 | -1.7 | -0.546 | -1.09 | 0.216 |
| 2 | 5e-6 | -1.139 | 1.688 | -1.7 | 17e-6 | -0.504 | -1.992 | 0.562 | -1.602 | -0.632 | -1.142 | -1.496 | 0.453 | -0.039 |
| 3 | 8e-6 | -0.546 | -1.09 | 0.216 | 26e-6 | 0.06 | -0.882 | -0.402 | -0.246 | -1.008 | 1.028 | -1.378 | -2.589 | 0.728 |
| 4 | 11e-6 | -0.504 | -1.992 | 0.562 | 35e-6 | 0.061 | 1.02 | 2.202 | -1.526 | -0.986 | 1.862 | 1.284 | 0.721 | -0.951 |
| 5 | 14e-6 | -1.602 | -0.632 | -1.142 | 44e-6 | 0.797 | 0.631 | -0.424 | 0.398 | 1.066 | -0.869 | 0.718 | 1.332 | 2.197 |
| 6 | 17e-6 | -1.496 | 0.453 | -0.039 | | | | | | | | | | |
| 7 | 20e-6 | 0.06 | -0.882 | -0.402 | | | | | | | | | | |
| 8 | 23e-6 | -0.246 | -1.008 | 1.028 | | | | | | | | | | |
| 9 | 26e-6 | -1.378 | -2.589 | 0.728 | | | | | | | | | | |
| 10 | 29e-6 | 0.061 | 1.02 | 2.202 | | | | | | | | | | |
| 11 | 32e-6 | -1.526 | -0.986 | 1.862 | | | | | | | | | | |
| 12 | 35e-6 | 1.284 | 0.721 | -0.951 | | | | | | | | | | |
| 13 | 38e-6 | 0.797 | 0.631 | -0.424 | | | | | | | | | | |
| 14 | 41e-6 | 0.398 | 1.066 | -0.869 | | | | | | | | | | |
| 15 | 44e-6 | 0.718 | 1.332 | 2.197 | | | | | | | | | | |
| 16 | 47e-6 | -0.253 | 1.524 | -1.39 | | | | | | | | | | |

Similarly, for the MATLAB_Script part with the 2 x 2 array input, since the input rate is 3, the input is going to be a 3 x 2 x 2 array. This is assigned to output, which has a rate of 1. Therefore, the output of this MATLAB_Script part is going to be 3 x 2 x 2 arrays, which come out at 1/3 times the rate the input 2 x 2 arrays are consumed. Let's verify this by looking at *Table3_2*.

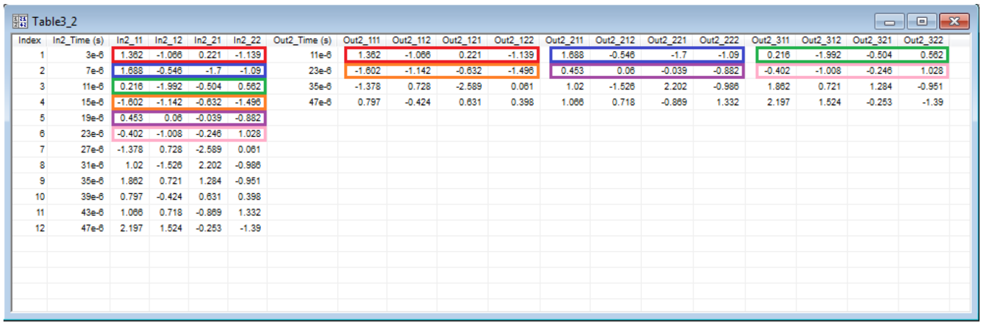


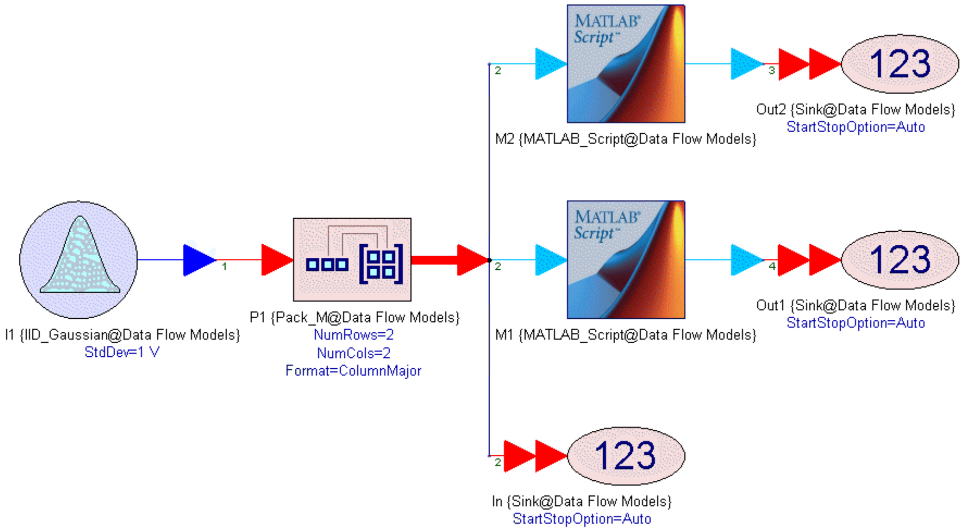
Table3_2

| Index | In2_Time (s) | In2_11 | In2_12 | In2_21 | In2_22 | Out2_Time (s) | Out2_111 | Out2_112 | Out2_121 | Out2_122 | Out2_211 | Out2_212 | Out2_221 | Out2_222 | Out2_311 | Out2_312 | Out2_321 | Out2_322 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 3e-6 | 1.362 | -1.066 | 0.221 | -1.139 | 11e-6 | 1.362 | -1.066 | 0.221 | -1.139 | 1.688 | -0.546 | -1.7 | -1.09 | 0.216 | -1.992 | -0.504 | 0.562 |
| 2 | 7e-6 | 1.688 | -0.546 | -1.7 | -1.09 | 23e-6 | -1.602 | -1.142 | -0.632 | -1.496 | 0.453 | 0.06 | -0.039 | -0.882 | -0.402 | -1.008 | -0.246 | 1.028 |
| 3 | 11e-6 | 0.216 | -1.992 | -0.504 | 0.562 | 35e-6 | -1.378 | 0.728 | -2.589 | 0.061 | 1.02 | -1.526 | 2.202 | -0.986 | 1.862 | 0.721 | 1.284 | -0.951 |
| 4 | 15e-6 | -1.602 | -1.142 | -0.632 | -1.496 | 47e-6 | 0.797 | -0.424 | 0.631 | 0.398 | 1.066 | 0.718 | -0.869 | 1.332 | 2.197 | 1.524 | -0.253 | -1.39 |
| 5 | 19e-6 | 0.453 | 0.06 | -0.039 | -0.882 | | | | | | | | | | | | | |
| 6 | 23e-6 | -0.402 | -1.008 | -0.246 | 1.028 | | | | | | | | | | | | | |
| 7 | 27e-6 | -1.378 | 0.728 | -2.589 | 0.061 | | | | | | | | | | | | | |
| 8 | 31e-6 | 1.02 | -1.526 | 2.202 | -0.986 | | | | | | | | | | | | | |
| 9 | 35e-6 | 1.862 | 0.721 | 1.284 | -0.951 | | | | | | | | | | | | | |
| 10 | 39e-6 | 0.797 | -0.424 | 0.631 | 0.398 | | | | | | | | | | | | | |
| 11 | 43e-6 | 1.066 | 0.718 | -0.869 | 1.332 | | | | | | | | | | | | | |
| 12 | 47e-6 | 2.197 | 1.524 | -0.253 | -1.39 | | | | | | | | | | | | | |

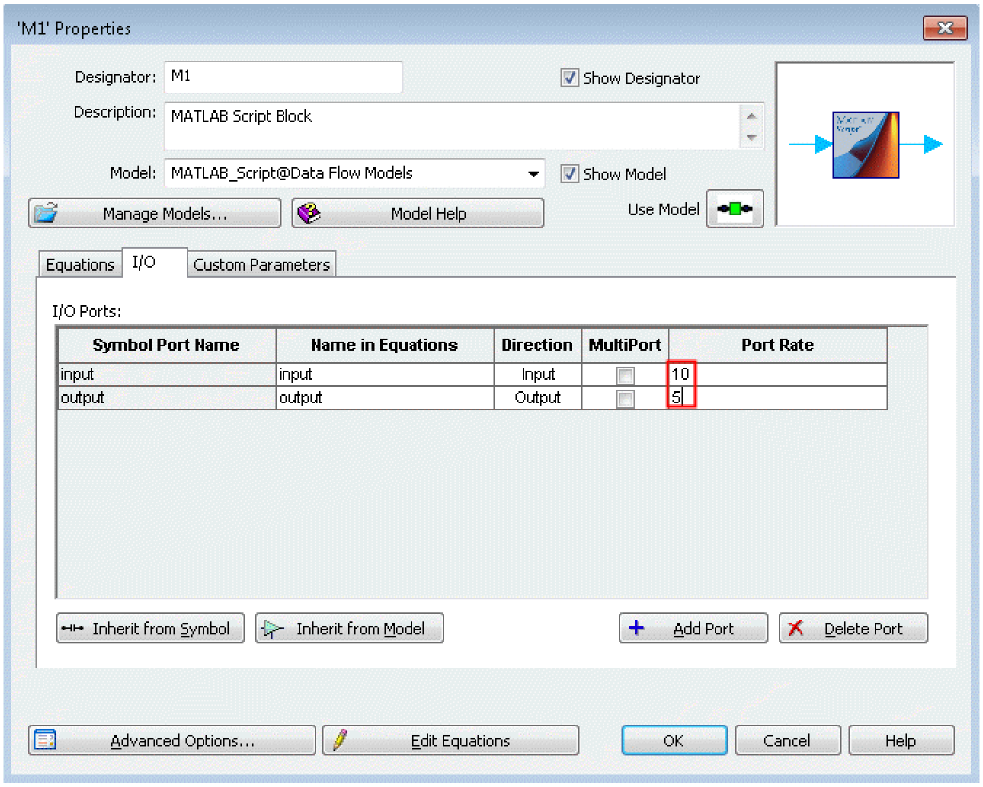
3. In this example,
   - the output rate is 1/3 times the input rate
   - the output arrays have a different number of dimensions compared to the input arrays

## Multirate Input and Output Example

1. Open design *4. Multirate Input and Output*.


MATLAB Script
2
3
123
M2 {MATLAB_Script@Data Flow Models}
Out2 {Sink@Data Flow Models}
StartStopOption=Auto
1
2
4
I1 {IID_Gaussian@Data Flow Models}
StdDev=1 V
P1 {Pack_M@Data Flow Models}
NumRows=2
NumCols=2
Format=ColumnMajor
M1 {MATLAB_Script@Data Flow Models}
Out1 {Sink@Data Flow Models}
StartStopOption=Auto
In {Sink@Data Flow Models}
StartStopOption=Auto

2. In this design, we create a 2 x 2 array of random normally distributed numbers and pass it to two MATLAB_Script parts. The MATLAB_Script parts are identical (one input with rate 10 and one output with rate 5).



'M1' Properties

Designator: M1 ☑ Show Designator

Description: MATLAB Script Block

Model: MATLAB_Script@Data Flow Models ☑ Show Model

Use Model

I/O Ports:

| Symbol Port Name | Name in Equations | Direction | MultiPort | Port Rate |
|---|---|---|---|---|
| input | input | Input | □ | 10 |
| output | output | Output | □ | 5 |

The MATLAB_Script code in the MATLAB_Script part *M1* is

```
output = zeros( 5, 2, 2 );
```

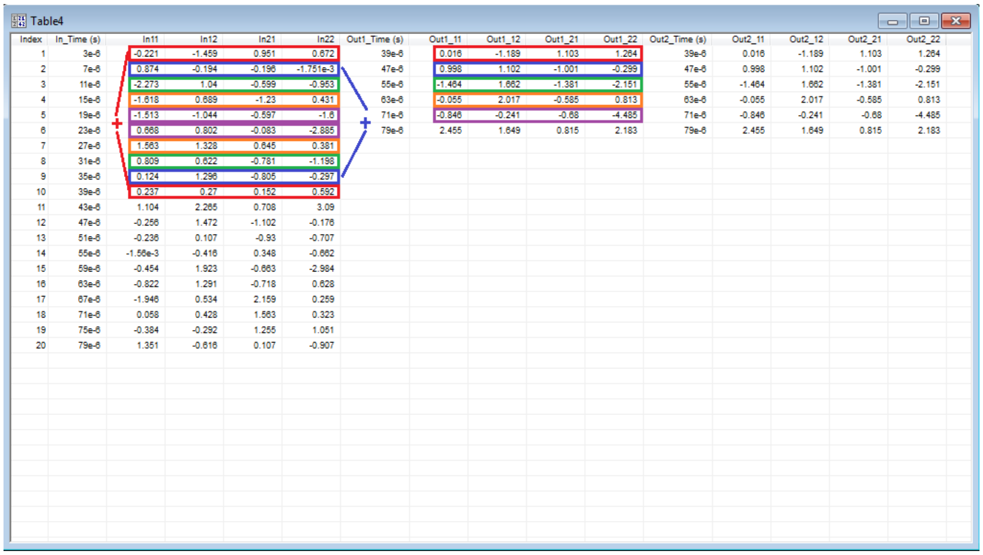
```
output( 1, :, : ) = input( 1, :, : ) + input( 10,
:, : );
output( 2, :, : ) = input( 2, :, : ) + input(  9,
:, : );
output( 3, :, : ) = input( 3, :, : ) + input(  8,
:, : );
output( 4, :, : ) = input( 4, :, : ) + input(  7,
:, : );
output( 5, :, : ) = input( 5, :, : ) + input(  6,
:, : );
```

Can you figure out what this code will produce at the output?

For the MATLAB_Script part *M1*, since the input rate is 10, the input is going to be a 10 x 2 x 2 array. The output is first initialized to a 5 x 2 x 2 array of zeros. Since the output rate is 5, the output of this MATLAB_Script part is going to be 2 x 2 arrays, which come out at 1/2 times the rate the input 2 x 2 arrays are consumed. The rest of the code computes the 5 output 2 x 2 matrices.

- the 1st output 2 x 2 matrix is the sum of the 1st and 10th 2 x 2 input matrices
- the 2nd output 2 x 2 matrix is the sum of the 2nd and 9th 2 x 2 input matrices
- the 3rd output 2 x 2 matrix is the sum of the 3rd and 8th 2 x 2 input matrices
- the 4th output 2 x 2 matrix is the sum of the 4th and 7th 2 x 2 input matrices
- the 5th output 2 x 2 matrix is the sum of the 5th and 6th 2 x 2 input matrices

3. Let's verify this by looking at *Table4*.



| Index | In_Time (s) | In11 | In12 | In21 | In22 | Out1_Time (s) | Out1_11 | Out1_12 | Out1_21 | Out1_22 | Out2_Time (s) | Out2_11 | Out2_12 | Out2_21 | Out2_22 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 3e-6 | -0.221 | -1.459 | 0.951 | 0.672 | 39e-6 | 0.016 | -1.189 | 1.103 | 1.264 | 39e-6 | 0.016 | -1.189 | 1.103 | 1.264 |
| 2 | 7e-6 | 0.874 | -0.194 | -0.196 | -1.751e-3 | 47e-6 | 0.998 | 1.102 | -1.001 | -0.299 | 47e-6 | 0.998 | 1.102 | -1.001 | -0.299 |
| 3 | 11e-6 | -2.273 | 1.04 | -0.599 | -0.953 | 55e-6 | -1.464 | 1.662 | -1.381 | -2.151 | 55e-6 | -1.464 | 1.662 | -1.381 | -2.151 |
| 4 | 15e-6 | -1.618 | 0.689 | -1.23 | 0.431 | 63e-6 | -0.055 | 2.017 | -0.585 | 0.813 | 63e-6 | -0.055 | 2.017 | -0.585 | 0.813 |
| 5 | 19e-6 | -1.513 | -1.044 | -0.597 | -1.6 | 71e-6 | -0.846 | -0.241 | -0.68 | -4.485 | 71e-6 | -0.846 | -0.241 | -0.68 | -4.485 |
| 6 | 23e-6 | 0.668 | 0.802 | -0.083 | -2.885 | 79e-6 | 2.455 | 1.649 | 0.815 | 2.183 | 79e-6 | 2.455 | 1.649 | 0.815 | 2.183 |
| 7 | 27e-6 | 1.563 | 1.328 | 0.645 | 0.381 | | | | | | | | | | |
| 8 | 31e-6 | 0.809 | 0.622 | -0.781 | -1.198 | | | | | | | | | | |
| 9 | 35e-6 | 0.124 | 1.296 | -0.805 | -0.297 | | | | | | | | | | |
| 10 | 39e-6 | 0.237 | 0.27 | 0.152 | 0.592 | | | | | | | | | | |
| 11 | 43e-6 | 1.104 | 2.265 | 0.708 | 3.09 | | | | | | | | | | |
| 12 | 47e-6 | -0.256 | 1.472 | -1.102 | -0.176 | | | | | | | | | | |
| 13 | 51e-6 | -0.236 | 0.107 | -0.93 | -0.707 | | | | | | | | | | |
| 14 | 55e-6 | -1.56e-3 | -0.416 | 0.348 | -0.662 | | | | | | | | | | |
| 15 | 59e-6 | -0.454 | 1.923 | -0.663 | -2.984 | | | | | | | | | | |
| 16 | 63e-6 | -0.822 | 1.291 | -0.718 | 0.628 | | | | | | | | | | |
| 17 | 67e-6 | -1.946 | 0.534 | 2.159 | 0.259 | | | | | | | | | | |
| 18 | 71e-6 | 0.058 | 0.428 | 1.563 | 0.323 | | | | | | | | | | |
| 19 | 75e-6 | -0.384 | -0.292 | 1.255 | 1.051 | | | | | | | | | | |
| 20 | 79e-6 | 1.351 | -0.616 | 0.107 | -0.907 | | | | | | | | | | |

4. The operation of this MATLAB_Script part can be generalized to "the $i^{th}$ output 2 x 2 matrix is the sum of the $i^{th}$ and $i^{th}$from the end 2 x 2 input matrices." Can you write a for loop that does that?

```
for i = 1:5;
    output( i, :, : ) = input( i, :, : ) + input( 11
-i, :, : );
end
```

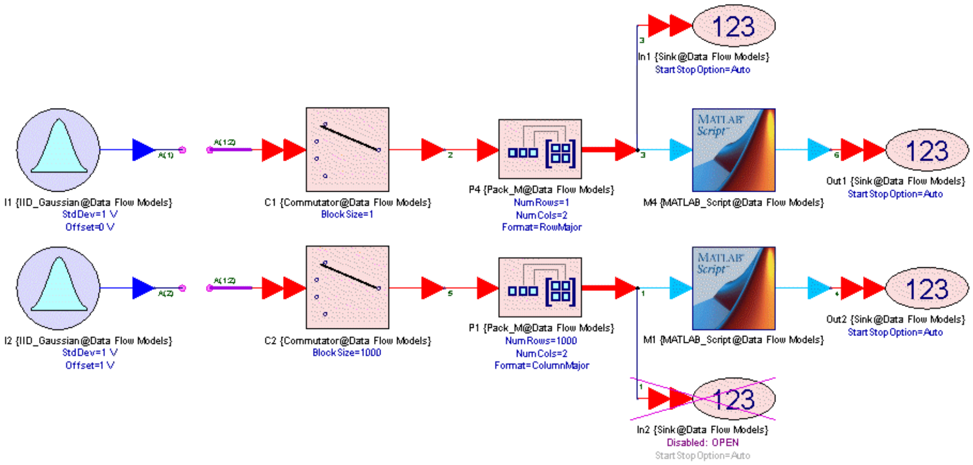
Since the for loop index is used to index in an array can you write this code in vectorized form (without the use of the for loop? Double click on the MATLAB_Script part *M2* to see this implementation.

5. In this example,

   - the output rate is 1/2 times the input rate
   - the output arrays have the same number of dimensions compared to the input arrays
   - the output array dimensions (2 x 2) are the same as the input ones (2 x 2)

## Array Averaging

1. Open design *5. Array Averaging*.



123
In1 {Sink@Data Flow Models}
StartStopOption=Auto
A(1)
A(12)
2
3
6
I1 {IID_Gaussian@Data Flow Models}
StdDev=1 V
Offset=0 V
C1 {Commutator@Data Flow Models}
BlockSize=1
P4 {Pack_M@Data Flow Models}
NumRows=1
NumCols=2
Format=RowMajor
MATLAB Script
M4 {MATLAB_Script@Data Flow Models}
123
Out1 {Sink@Data Flow Models}
StartStopOption=Auto
A(2)
A(12)
5
1
4
I2 {IID_Gaussian@Data Flow Models}
StdDev=1 V
Offset=1 V
C2 {Commutator@Data Flow Models}
BlockSize=1000
P1 {Pack_M@Data Flow Models}
NumRows=1000
NumCols=2
Format=ColumnMajor
MATLAB Script
M1 {MATLAB_Script@Data Flow Models}
123
Out2 {Sink@Data Flow Models}
StartStopOption=Auto
1
123
In2 {Sink@Data Flow Models}
Disabled: OPEN
StartStopOption=Auto

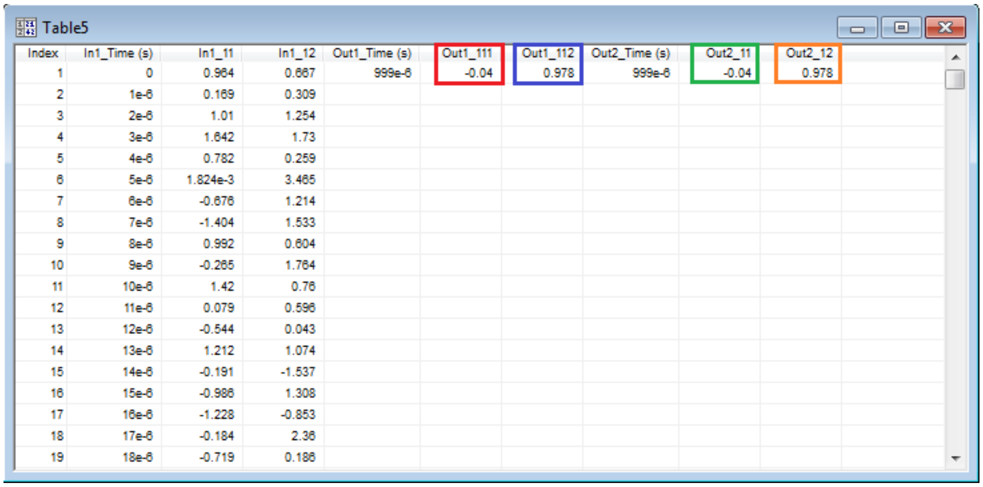
2. In this design, we have two random sources that generate a random number with mean (*Offset*) 0 (instance *I1*) and 1 (instance *I2*) and standard deviation ( *StdDev*) 1. In the top path, we interleave samples from these sources using a Commutator. Since source *I1* is connected to the first input of the Commutator *C1* and source *I2* is connected to its second input and the *BlockSize* parameter is set to 1, the samples at the output of *C1* will be alternating between samples from *I1* and I2_ starting with *I1*. Then we pack the samples at the output of *C1* into 1 x 2 arrays in *RowMajor* form. What this means is that the (1,1) elements in the arrays coming out of *P4* are random numbers normally distributed with mean 0 and standard deviation of

1 and the (1,2) elements in the arrays coming out of *P4*are random numbers normally distributed with mean 1 and standard deviation of 1. These arrays are then sent to an MATLAB_Script part, which has one input with rate 1000 and one output with rate 1. The code in the MATLAB_Script part is the single line:

```
output = mean( input );
```

Can you figure out what this code will produce at the output?

Since the input rate is 1000, the input is going to be a 1000 x 1 x 2 array. The mean function averages along the first non-singleton array dimension (in our case this is the first dimension of 1000) and reduces this dimension to 1. So the output is going to be a 1 x 1 x 2 array. Since we are averaging random numbers we expect the average to be close to their mean. Therefore, the (1,1,1) output array element should be close to 0 and the (1,1,2) output array element should be close to 1. Let's verify this by looking at *Table5*.



| Index | In1_Time (s) | In1_11 | In1_12 | Out1_Time (s) | Out1_111 | Out1_112 | Out2_Time (s) | Out2_11 | Out2_12 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0.964 | 0.667 | 999e-6 | -0.04 | 0.978 | 999e-6 | -0.04 | 0.978 |
| 2 | 1e-6 | 0.169 | 0.309 | | | | | | |
| 3 | 2e-6 | 1.01 | 1.254 | | | | | | |
| 4 | 3e-6 | 1.642 | 1.73 | | | | | | |
| 5 | 4e-6 | 0.782 | 0.259 | | | | | | |
| 6 | 5e-6 | 1.824e-3 | 3.465 | | | | | | |
| 7 | 6e-6 | -0.676 | 1.214 | | | | | | |
| 8 | 7e-6 | -1.404 | 1.533 | | | | | | |
| 9 | 8e-6 | 0.992 | 0.604 | | | | | | |
| 10 | 9e-6 | -0.265 | 1.764 | | | | | | |
| 11 | 10e-6 | 1.42 | 0.76 | | | | | | |
| 12 | 11e-6 | 0.079 | 0.596 | | | | | | |
| 13 | 12e-6 | -0.544 | 0.043 | | | | | | |
| 14 | 13e-6 | 1.212 | 1.074 | | | | | | |
| 15 | 14e-6 | -0.191 | -1.537 | | | | | | |
| 16 | 15e-6 | -0.986 | 1.308 | | | | | | |
| 17 | 16e-6 | -1.228 | -0.853 | | | | | | |
| 18 | 17e-6 | -0.184 | 2.36 | | | | | | |
| 19 | 18e-6 | -0.719 | 0.186 | | | | | | |

Indeed, the *Out1* array (see columns marked red and blue) has three dimensions (there are three indices following its name in the columns showing its contents), the first dimension has size 1 (the first index only takes the value of 1), the second dimension has size 1 (the second index only takes the value of 1), and the third dimension has size 2 (the third index takes the values 1 and 2). The value of the (1,1,1) element is close to 0 (the mean of the samples from source *I1*) and the value of the (1,1,2) element is close to 1 (the mean of the samples from source *I2*).

3. In the bottom path, we again interleave samples from the two random sources using a Commutator. However, the *BlockSize* parameter of the Commutator *C2* is set to 1000. Therefore, its output will interleave blocks of 1000 samples from the two source starting with source *I1*. The output of *C2* is packed into 1000 x 2 arrays in *ColumnMajor* form. What this means is that

the first column (:,1) in the arrays coming out of *P1* contains the block of 1000 samples coming out of *I1* and the second column (:,2) in the arrays coming out of *P1* contains the block of 1000 samples coming out of *I2*. The arrays at the output of *P1*are sent to an MATLAB_Script part, which has one input with rate 1 and one output with rate 1. The code in the MATLAB_Script part is again the single line:

```
output = mean( input );
```

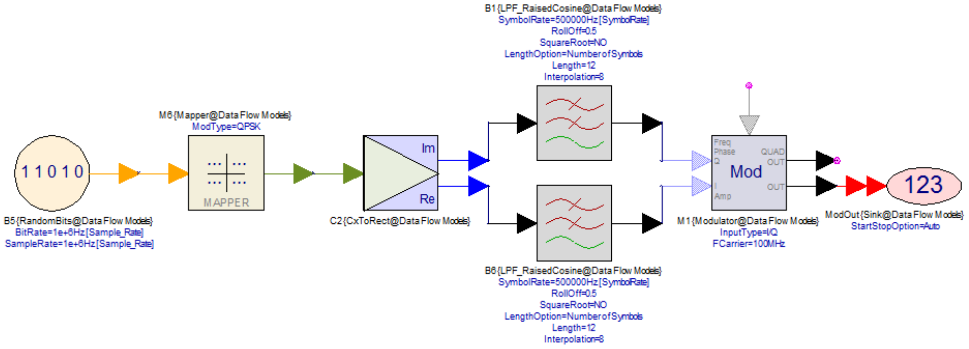
Can you figure out what this code will produce at the output?

Since the input rate is 1, the input is going to be a 1000 x 2 array. The mean function averages along the first non-singleton array dimension (in our case this is the first dimension of 1000) and reduces this dimension to 1. So the output is going to be a 1 x 2 array. Since we are averaging random numbers we expect the average to be close to their mean. Therefore, the (1,1) output array element should be close to 0 and the (1,2) output array element should be close to 1. This can be verified by looking at *Table5* above. The *Out2* array (see columns marked green and orange) has two dimensions (there are two indices following its name in the columns showing its contents), the first dimension has size 1 (the first index only takes the value of 1), and the second dimension has size 2 (the second index takes the values 1 and 2). The value of the (1,1) element is close to 0 (the mean of the samples from source *I1*) and the value of the (1,2) element is close to 1 (the mean of the samples from source *I2*).

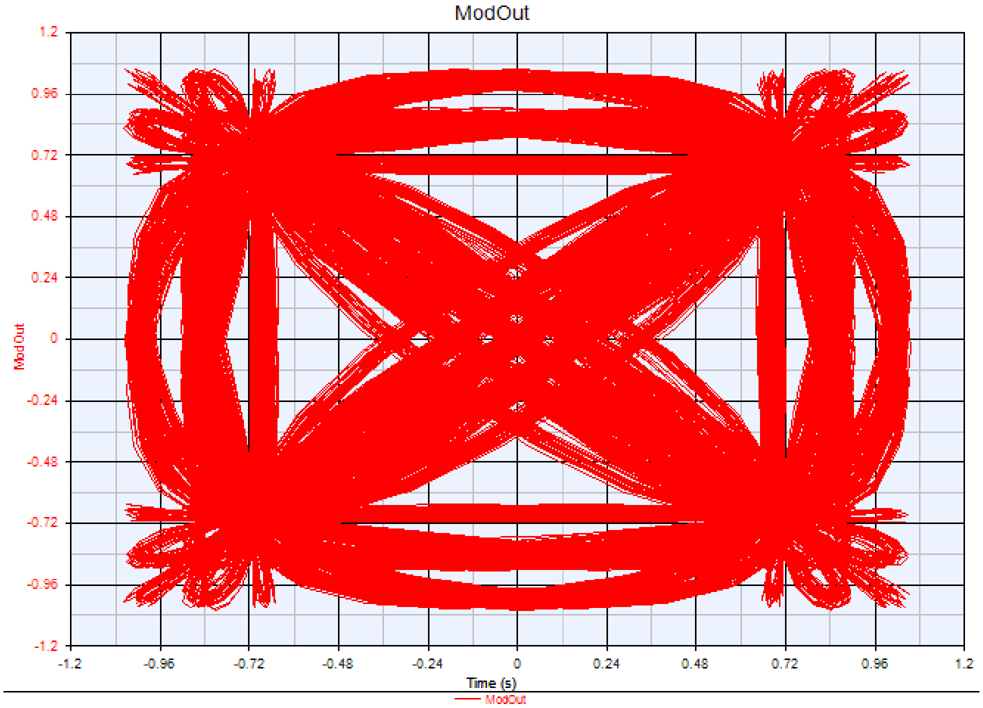
### Time Domain Power Measurements

In this tutorial example, we will post-process simulation data to compute some useful power measurements.

1. Open the workspace *PostProcessingSimulationData.wsv* under <SystemVue Installation Directory>\Examples\Tutorials\Algorithm_Design\MATLAB_Script.
2. Open *Design1* in the folder *1. Time Domain Power Measurements*. This design generates a QPSK signal with a symbol rate of 0.5 MHz sampled at 4 MHz.


B1{LPF_RaisedCosine@Data Flow Models}
SymbolRate=500000Hz[SymbolRate]
RollOff=0.5
SquareRoot=NO
LengthOption=NumberofSymbols
Length=12
Interpolation=8
M6{Mapper@Data Flow Models}
ModType=QPSK
1 1 0 1 0
MAPPER
Im
Re
Freq
Phase
Q
I
Amp
QUAD OUT
OUT
Mod
123
B5{RandomBits@Data Flow Models}
BitRate=1e+6Hz[Sample_Rate]
SampleRate=1e+6Hz[Sample_Rate]
C2{CxToRect@Data Flow Models}
M1{Modulator@Data Flow Models}
InputType=I/Q
FCarrier=100MHz
ModOut{Sink@Data Flow Models}
StartStopOption=Auto
B6{LPF_RaisedCosine@Data Flow Models}
SymbolRate=500000Hz[SymbolRate]
RollOff=0.5
SquareRoot=NO
LengthOption=NumberofSymbols
Length=12
Interpolation=8

The trajectory plot is shown in *Graph1*



3. Add an Equations page in folder *1. Time Domain Power Measurements* and call it *PostProcessingEqns1*. Try writing some MATLAB Script code that computes the instantaneous power of the signal at the output of the modulator. This signal is saved in the variable *ModOut* in the dataset *Desing1_Data*. Assume the reference resistance is 50 Ohms. The signal at the output of the modulator is represented in a complex envelope form. For a complex envelope signal *v(t)* the power is $|v(t)|^2$ / ( 2 · *R*), where *R* is the reference resistance and the scaling factor of 2 is used because *v(t)* represents the signal envelope (not the real signal) and the mean power of the carrier is 0.5.

**NOTE** In order to be able to access dataset variables from an Equations page, you need to use the using function.

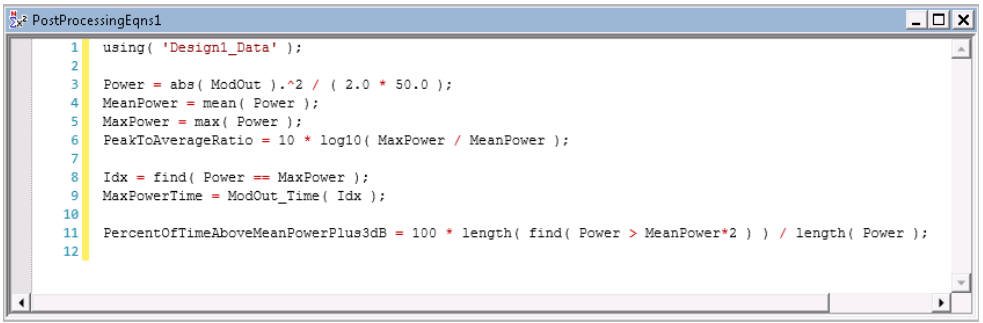
Add some more code to compute the mean power, the maximum instantaneous power, the peak to average ratio, the time instant the maximum instantaneous power occurs, and the percentage of time the instantaneous power exceeds the average power by 3 dB. The code might look like this:

```
using( 'Design1_Data' );

Power = abs( ModOut ).^2 / ( 2.0 * 50.0 );
MeanPower = mean( Power );
MaxPower = max( Power );
```

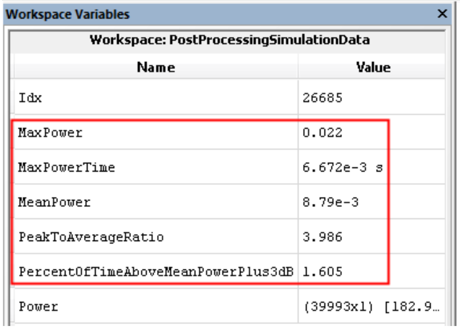
```
PeakToAverageRatio = 10 * log10( MaxPower /
MeanPower );

Idx = find( Power == MaxPower );
MaxPowerTime = ModOut_Time( Idx );

PercentOfTimeAboveMeanPowerPlus3dB = 100 * length(
find( Power > MeanPower*2 ) ) / length( Power );
```

First the using function is used to get access to dataset variables. The instantaneous power is computed using the expression $|v(t)|^2 / ( 2 \cdot R )$. The abs function is used to get the magnitude of the signal and the element-wise power operator (.^) is used to compute the squares of all elements in the variables *ModOut*. *MeanPower* and *MaxPower* are easily computed by calling the mean and max functions respectively. The *PeakToAverageRatio* in dB can then be computed from *MeanPower* and *MaxPower*. To find the time instant where the maximum instantaneous power occurs, we use the find function with the argument *Power == PowerMax*. This returns the index of the *Power* array element that is equal to *MaxPower*. We then use this index to extract the time value the maximum occurred by indexing into the *ModOut_Time* dataset variable (the independent time associated with *ModOut*). Finally, to find the percentage of time the instantaneous power exceeds the average power by 3 dB, we again use the find function with the argument *Power > MeanPower* * 2. This returns all the indices (in a row vector) of the *Power* array elements whose values are greater than *MeanPower* * 2 (3 dB higher than the average power). *PercentOfTimeAboveMeanPowerPlus3dB* is then computed by taking the ratio of the number of elements (length) in the row vector returned by find to the number of elements of *Power*.

4. Press Ctrl+G or click the Run Equations button from the Equation Toolbar to execute the code you have written. The values of the variables will be shown in the Variable Viewer. The power values are in *Watts* (since *ModOut* values are in *Volts*).



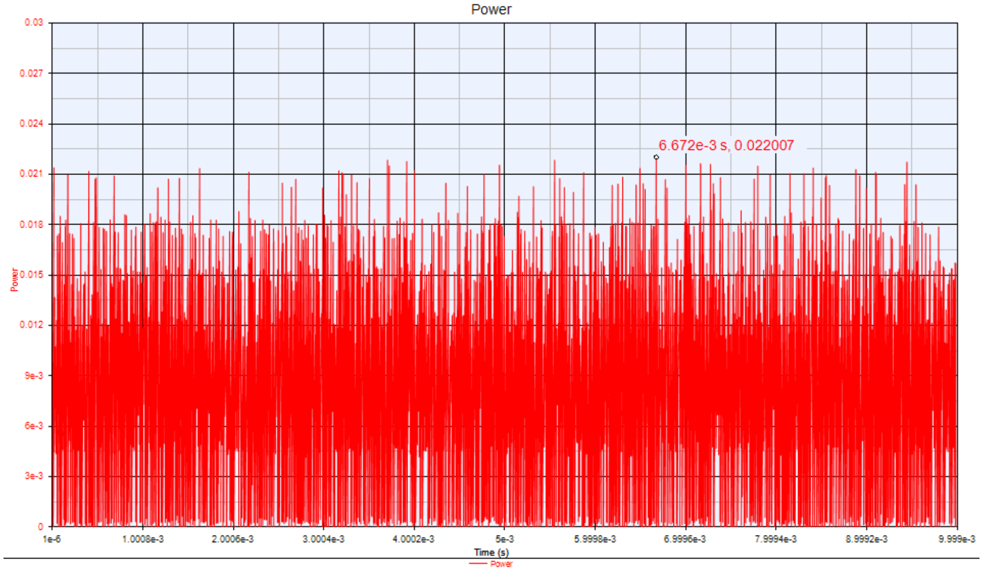
```
using( 'Design1_Data' );

Power = abs( ModOut ).^2 / ( 2.0 * 50.0 );
MeanPower = mean( Power );
MaxPower = max( Power );
PeakToAverageRatio = 10 * log10( MaxPower / MeanPower );

Idx = find( Power == MaxPower );
MaxPowerTime = ModOut_Time( Idx );

PercentOfTimeAboveMeanPowerPlus3dB = 100 * length( find( Power > MeanPower*2 ) ) / length( Power );
```



Workspace: PostProcessingSimulationData

| Name | Value |
|---|---|
| Idx | 26685 |
| MaxPower | 0.022 |
| MaxPowerTime | 6.672e-3 s |
| MeanPower | 8.79e-3 |
| PeakToAverageRatio | 3.986 |
| PercentOfTimeAboveMeanPowerPlus3dB | 1.605 |
| Power | (39993x1) [182.9… |

5. Right click on the *Power* variable and select *Add to Graph* > *New Graph*. A graph showing the instantaneous power vs time is created. Try placing a marker at the time point *MaxPowerTime* sec and visually verify that it is where the maximum occurs.

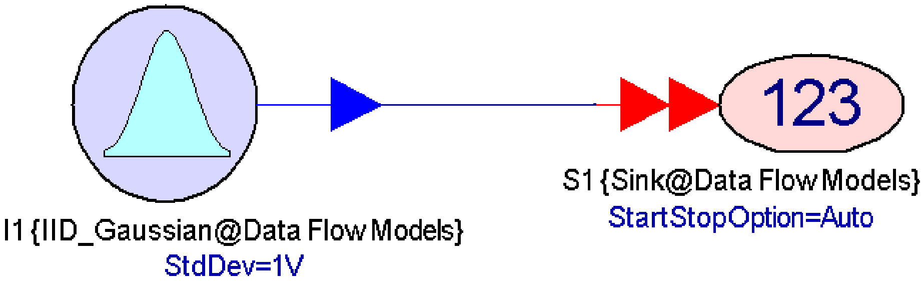


6. Experiment with different values of the Raised Cosine filters' *RollOff* factor and see how the *MeanPower*, *MaxPower*, *PeakToAverageRatio*, and *PercentOfTimeAboveMeanPowerPlus3dB* vary.

## Histogram

In this tutorial example, we will post-process simulation data to create a histogram. There is already a histogram function in SystemVue, which we will use as a reference to validate our code.

1. Open the workspace *PostProcessingSimulationData.wsv* under <SystemVue Installation Directory>\Examples\Tutorials\Algorithm_Design\MATLAB_Script.
2. Open *Design2* in the folder *2. Histogram*. This design generates a random number that is normally distributed with zero mean and standard deviation of 1. The simulation generates 10000 numbers saved in the array variable *S1* in the dataset *Design2_Data*.

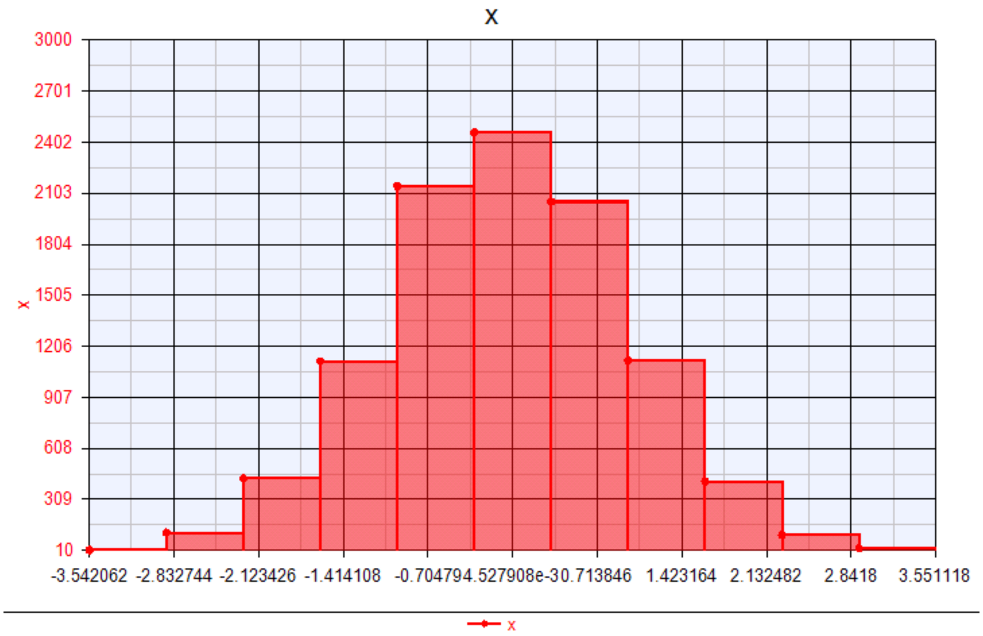

123
S1 {Sink@Data Flow Models}
StartStopOption=Auto
I1 {IID_Gaussian@Data Flow Models}
StdDev=1V

3. Add an Equations page in folder *2. Histogram* and call it *PostProcessingEqns2*. Use the using function to get access to the variables in the *Design2_Data* dataset and call the histogram function to create a histogram of *S1*with 11 bins.

```
using( 'Design2_Data' );

x = histogram( S1, 11 );
```

The histogram function will create a histogram with 11 equal width bins starting from the minimum value of *S1* to the maximum value of *S1*. If the boundaries of the bins are *bins(i)*, *i* = 1, 2, ..., 12, the histogram values *x(i)*, *i* = 1, 2, ..., 11, are the number of values *v* in *S1*, such that *bins(i)* ≤ *v* < *bins(i+1)*. The number of values *v* in *S1* that is exactly equal to *bins(*12*)* (the maximum value of *S1*) are counted in the last histogram value *x(*11*)*.

4. Click the Run Equations button in the Equations page window to run the code you have written. Right-click on the *x* variable and select *Add to Graph > New Graph*. A graph showing the histogram of the *S1* values is created.



5. Try writing some MATLAB Script code that computes the histogram of *S1* the same way the histogram function does.

NOTE To get a vector of *N* equally spaced values (bin boundaries) starting at *a* and ending at *b* used the linspace function.

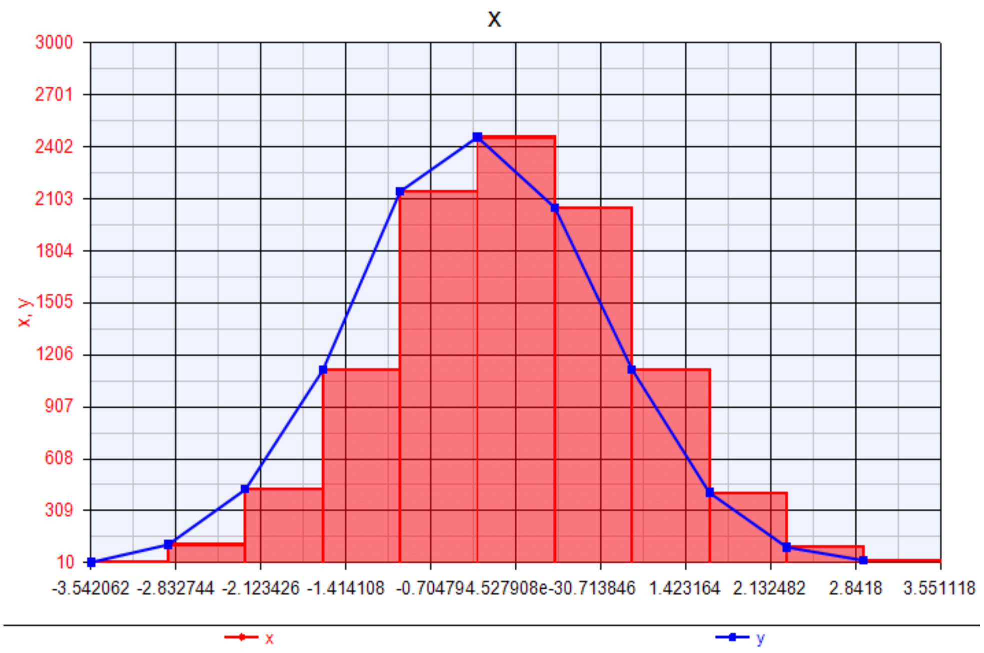
The code might look like this:

```
using( 'Design2_Data' );

x = histogram( S1, 11 );

DataMin = min( S1 );
DataMax = max( S1 );

bins = linspace( DataMin, DataMax, 12 );

y = zeros( 11, 1 );

for i = 1:length( y );
    indices = find( bins(i) <= S1 & S1 < bins(i+1)
);
    y(i) = length( indices );
end

indices = find( S1 == bins(i+1) );
y(i) = y(i) + length( indices );

setindep( 'y', 'bins' );
```

We first find the minimum and maximum values of *S1* and create the bin boundaries using the linspace function. Then we initialize our histogram *y* to a zero array of 11 elements (it is important to define *y* as a column vector; otherwise it will not plot properly on graphs). Then we use the find function to get the indices of the values in *S1* that satisfy the condition *bins(i)* ≤ *v* < *bins(i+1)* (values inside bin *i*). To get a count of these values we use the length function on the *indices* vector. Finally, using find and length again we get the number of values in *S1* that are exactly equal to the right boundary of the last bin (*bins(i+1)* = *bins*(12*)* = max(*S1*)) and we add it to the last histogram value *y(i)* = *y*(11*)* (at the end of the for loop the value of the loop index *i* is the last value it got, that is, length( y ) = 11). The last call to the setindep function creates a dependent/independent relationship between the variables *y* and *bins* so that *y* can be properly plotted on a graph.

6. Click the Run Equations button in the Equations page window to run the code you have written. Right click on the *y* variable and select *Add to Graph > Add to 'PostProcessEqn2_x'* to plot *y* on the same graph as *x*. Verify that the values of *x* and *y* match.

**CAUTION** *y* is not going to be plotted as a bar graph but as a line graph. The histogram function sets a certain property on the variable it returns to make it plot in a bar graph form. Setting such a property on the variable *y*, which is computed in the Equations page using a sequence of MATLAB Script statements (assignments, for loops, function calls, etc.), is not a supported feature.

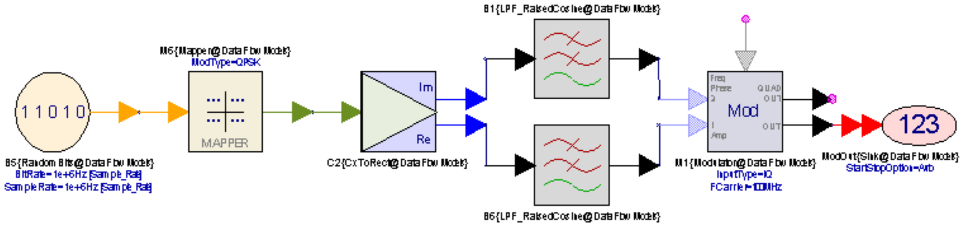


## Spectrum Averaging

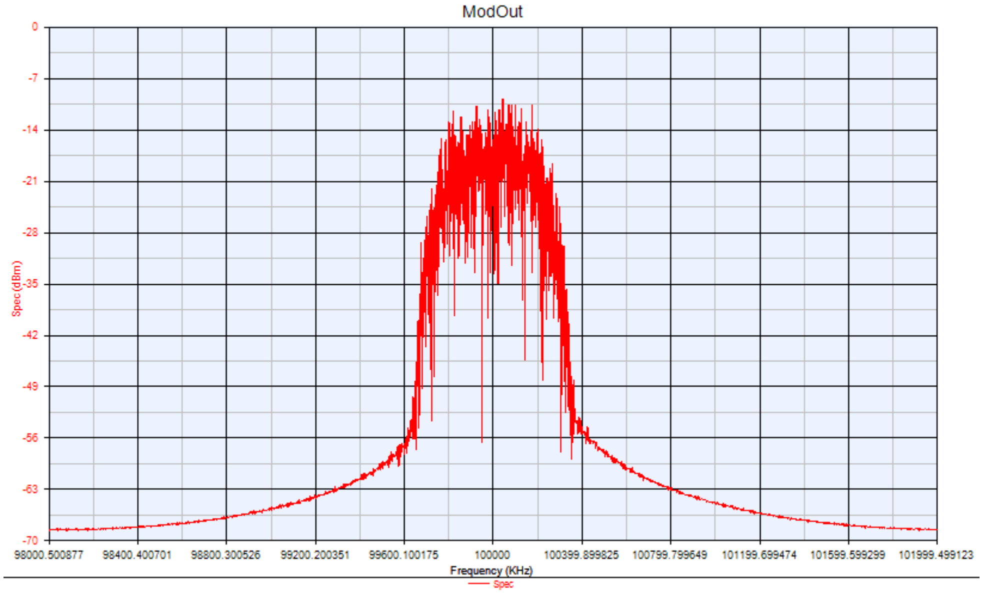
## Spectrum Averaging

In this tutorial example, we will extend the custom equations in the *Spectrum* graph to allow averaging spectra of multiple segments.

1. Open the workspace *PostProcessingSimulationData.wsv* under <SystemVue Installation Directory>\Examples\Tutorials\Algorithm_Design\MATLAB_Script.
2. Open *Design3* in the folder *3. Spectrum Averaging*. This is the same design as *Design1* in the folder *1. Time Domain Power Measurements*, which generates a QPSK signal with a symbol rate of 0.5 MHz sampled at 4 MHz.


B1{LPF_RaisedCosine@Data Flow Models}
M6{Mapper@Data Flow Models}
ModType=QPSK
1 1 0 1 0
MAPPER
Im
Re
Freq
Phase
Q
I
Amp
QUAD OUT
OUT
Mod
123
B5{Random Bits@Data Flow Models}
BitRate=1e+6Hz [Sample_Rate]
SampleRate=1e+6Hz [Sample_Rate]
C2{CxToRect@Data Flow Models}
M1{Modulator@Data Flow Models}
InputType=IQ
FCarrier=100MHz
ModOut{Sink@Data Flow Models}
StartStopOption=Arb
B6{LPF_RaisedCosine@Data Flow Models}

3. Run *Design3 Analysis*, open the dataset *Design3_Data*, right click on the variable *ModOut*, and select *Add to Graph* > *New Graph Series Wizard*. In the *Graph Series Wizard* window, select Spectrum in the *Select Type of Series* area, and press the OK button. Press the OK button again and a graph showing the spectrum of the *ModOut* signal is created.



4. Double-click on the graph, and press the Edit... button in the first row of the grid. The *Graph Series Wizard* window opens again. Press the Edit Equations button. An Equations page opens with some equations that calculate the spectrum of a time-domain signal. Dismiss all the windows that have opened

by pressing the Cancel buttons. The function *MySpectrumAnalyzer* inside the folder *3. Spectrum Averaging* is a simplified version of the equations inside the *Custom Equations* page of the *Graph Series Wizard* window wrapped into a function.

5. Add an Equations page in folder *3. Spectrum Averaging* and call it *PostProcessingEqns3*. Try writing some MATLAB Script code that breaks down the signal *ModOut* into *N* segments (you can define *N* to be a variable in this Equations page), computes the spectrum of each segment by calling the *MySpectrumAnalyzer* function, and averages the spectra.

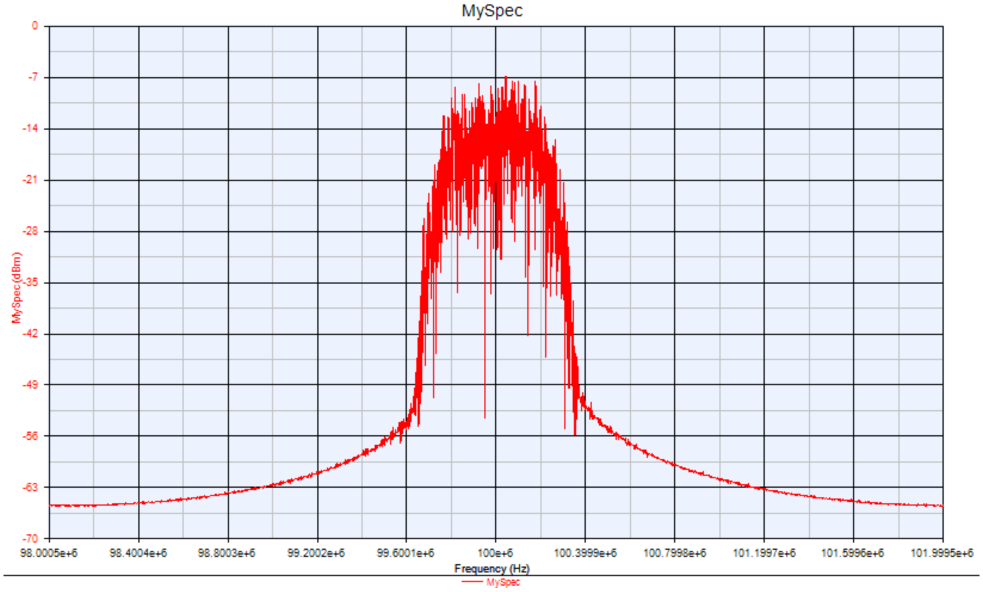
**NOTE** In order to be able to access dataset variables from an Equations page, you need to use the using function.

The code might look like this:

```
using( 'Design3_Data' );

N = 1;

BlockSize = floor( length( ModOut ) / N );

MySpec = zeros( BlockSize, 1 );

for i = 1:N
    ModOutI = ModOut( (i-1)*BlockSize+1 :
i*BlockSize );
    TimeVecI = ModOut_Time( (i-1)*BlockSize+1 :
i*BlockSize );
    [MySpecI, MyFreq] = MySpectrumAnalyzer(
ModOutI, TimeVecI );
    MySpec = MySpec + MySpecI;
end

MySpec = MySpec / N;
MyFreq = MyFreq + ModOut_Fc;

setindep( 'MySpec', 'MyFreq' );
setdisplayunit( 'MySpec', 'dBm' );
setdisplayunit( 'MyFreq', 'Hz' );
```

We first compute the number of samples (*BlockSize*) of each segment by dividing the length of the *ModOut* variable with *N* (the number of segments). We use the floor function to make sure *BlockSize* has an integer value. Then we define the *MySpec* variable that will hold the averaged spectrum and we initialize it to a vector of zeros (it is important to define this as a column vector; otherwise it will not plot properly on graphs). Next we use a for loop to get *N* non-overlapping segments of data *ModOutI* from the *ModOut* variable and the corresponding time vector *TimeVecI* from the *ModOut_Time* variable, call the *MySpectrumAnalyzer* function to compute their spectrum and add all the spectra to the *MySpec* variable. Finally, we divide *MySpec* with *N* to get the averaged value and offset *MyFreq* by *ModOut_Fc* (the characterization frequency of the *ModOut* signal). The last three lines set up a dependent/independent relationship between the variables *MySpec* and *MyFreq* and set the units for these variables so that *MySpec* can be properly plotted on a graph.

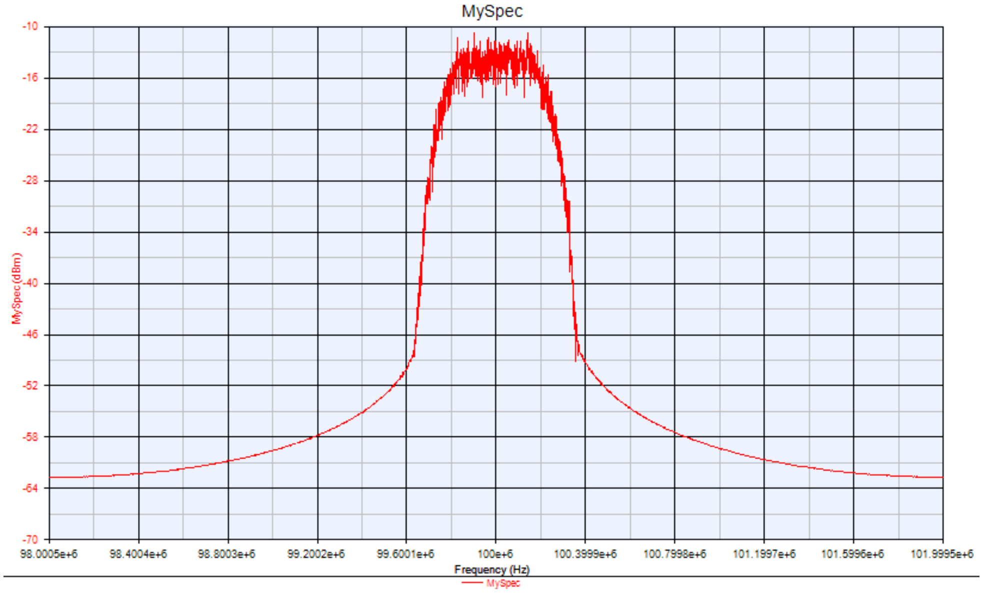
6. Click the Run Equations button in the *PostProcessingEqns3* Equations page window to run the code you have written. Right click on the *MySpec* variable and select *Add to Graph* > *New Graph*. A graph showing the spectrum of *ModOut* is created.



7. The spectrum in this graph should look the same as the one we created earlier.

8. Go to *Design3 Analysis*, set *Number of Samples* to 10000, and simulate again.

9. Go to *PostProcessingEqns3*, set *N* to 10, and click the Run Equations button. You can see the results of averaging 10 spectra in the graph plotting *MySpec* .



## Converting UFMC Simulation Script

In this tutorial, we will convert an MATLAB script, UFMC_OFDM___TransceiverChain.m, to a SystemVue model-based design using MATLAB_Script models and workspace tree equations. UFMC_OFDM___TransceiverChain.m was created and made publicly available by Alcatel-Lucent. This script demonstrates how a typical wireless communication simulation is done in MATLAB. The MATLAB script file is located in <SystemVue Installation Directory>\Examples\Tutorials\Algorithm_Design\MATLAB_Script\UFMC_OFDM___T m and the converted SystemVue workspace is in <SystemVue Installation Directory>\Examples\Tutorials\Algorithm_Design\MATLAB_Script\Converting_UFMC wsv. You need to have MATLAB R2014a or later installed on your machine in order to run the tutorial workspace in SystemVue because the script uses certain functions that are not available in MATLAB Script.

1. After examing UFMC_OFDM___TransceiverChain.m, you will understand that lines 1 to 294 is used to setup the parameters and structures (*PAR* and *PARderived*) for the main simulation loops. You can create a workspace equation, name it "Setup", and directly copy lines 1 to 294 to the workspace equation. The only change between UFMC_OFDM___TransceiverChain.m and the "Setup" equation in the tutorial workspace is line 106, where PAR.NTTIs = 10000 is changed to PAR.NTTIs = 100 to reduce the simulation time for demonstration purpose. Once the "Setup" equation is created, right click on the equation editing area, turn off "Auto-calculate", and select "Use MATLAB <version>". The reason to select the retail MATLAB version is because the script uses functions like "chebwin" which is not available in MATLAB Script mode.

```
% =======================================================================
% MatLab Script “UFMC_OFDM___TransceiverChain.m” Terms of Use
% Version: 2014-03-21
%
% This MatLab script was created and is made available free of charge
% by Alcatel-Lucent Deutschland AG, Lorenzstraße 10, 70435 Stuttgart
% (“Alcatel-Lucent”).
%
% You may use this script for any purpose, and you may modify the
% contents of this script and incorporate the script into your own
% scripts in whole or in part.
%
% If you use this script without modification, you must preserve the
% copyright notice identifying Alcatel-Lucent. If you incorporate this
% script into your own scripts, you must include into them a notice
% stating that portions of the script were created by Alcatel-Lucent.
%
% Alcatel-Lucent does not provide any technical support for the script
% or for MatLab itself. For enquiries related to the above, you may
% contact
%     Frank Schaich, e-mail frank.schaich@acatel-lucent.com, or
%     Thorsten Wild, e-mail thorsten.wild@alcatel-lucent.com.
%
% Alcatel-Lucent does not provide any representation or warranty with
% regard to the functionality of this script, and Alcatel-Lucent does
% does not assume any liability for the functionality of this script.
% Furthermore, Alcatel-Lucent does not represent, warrant, guarantee
% or otherwise assume any liability for the fitness of the script for
% any particular purpose and for any consequences the use of this
% script may have.
%
```

```
% By making available the script, Alcatel-Lucent
does not provide an
% express or implied license as to any of its or
its related companies’
% patents or patent applications . The granting of
rights embodied
% in this notice relates only to the script itself.
%
% MatLab is software licensed separately by The
MathWorks, Inc.
%
%
====================================================
====================
%
% Further technical details on UFMC can be found in
the following
% references (and references therein):
% [1] F. Schaich, T. Wild, Y. Chen , “Waveform
contenders for 5G –
%     suitability for short packet and low latency
transmissions”,
%     accepted for IEEE VTCs’14, Seoul, Korea,
April 2014
% [2] V. Vakilian, T. Wild, F. Schaich, S.t. Brink,
J.-F. Frigon,
%     "Universal-Filtered Multi-Carrier Technique fo
r Wireless Systems
%     Beyond LTE", 9th International Workshop on
Broadband Wireless Access
%    (BWA) @ IEEE Globecom'13, Atlanta, GA, USA,
December 2013.




clear all

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%
%%% simplified UFMC
chain:                                            %%%
%%% single-user, no delay, AWGN, BPSK/QPSK, ZF/MF
/MMSE and FFT based %%%
%%% detection, UFMC and CP-
OFDM                                       %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%   Parameter settings %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

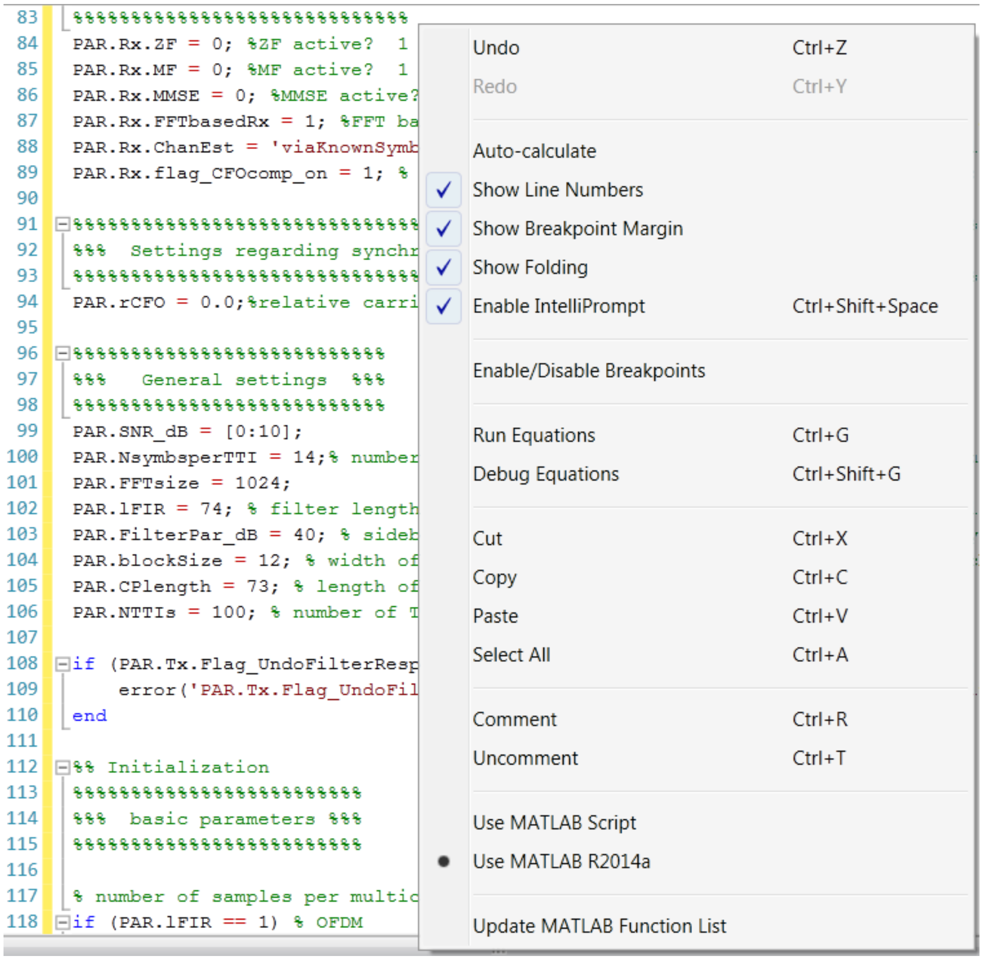
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%   burst placing, modulation   %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
PAR.blockShift = 28;% frequency shift of lowest-
frequency block in subcarriers
PAR.nPRB = 10;% Allocation width in number of
subbands (sub-band width defined in PAR.blockSize
below)
PAR.DataSource = 'QPSK'; % 'BPSK', 'QPSK'

PAR.Constellation{1} = [-1 1 ; 1 0];              %
BPSK signal constellation
PAR.Constellation{2} = [(+1+1i)/sqrt(2) 0 0 ; ... %
QPSK signal constellation (Gray mapping)
                        (+1-1i)/sqrt(2) 0 1 ; ...
                        (-1+1i)/sqrt(2) 1 0 ; ...
                        (-1-1i)/sqrt(2) 1 1 ];

PAR.Tx.Flag_UndoFilterResponse = 1; %if flag set to
1: filter response in pass-band is undone in Tx to
uniform power distribution beteen subcarriers.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%   Parameteres for Rx  %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
PAR.Rx.ZF = 0; %ZF active?  1 --> active, 0 --> not
active
PAR.Rx.MF = 0; %MF active?  1 --> active, 0 --> not
active
PAR.Rx.MMSE = 0; %MMSE active?   1 --> active, 0 --
> not active
PAR.Rx.FFTbasedRx = 1; %FFT based detection
active?   1 --> active, 0 --> not active
PAR.Rx.ChanEst = 'viaKnownSymbs'; %'viaKnownSymbs'
(ideal yet, i.e. all symbols used as pilots)
PAR.Rx.flag_CFOcomp_on = 1; % CFO compensation
(time domain) done in receiver: 1 = yes; 0 = no

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  Settings regarding synchronization
misalignments per alloc and per layer   %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
PAR.rCFO = 0.0;%relative carrier frequency offset
in subcarrier spacings

%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%   General settings  %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%
PAR.SNR_dB = [0:10];
PAR.NsymbsperTTI = 14;% number of Multi-carrier
symbols per TTI (i.e. per simulation drop)
```

```
PAR.FFTsize = 1024;
PAR.lFIR = 74; % filter length: 1 means OFDM, >1
uses a Dolph-Chebychev FIR filter
PAR.FilterPar_dB = 40; % sideband attenuation
(design parameter of Dolph-Chebychev filters)
PAR.blockSize = 12; % width of subband in number of
subcarriers (needs to match to Filterbandwidth)
PAR.CPlength = 73; % length of OFDM CP in samples
PAR.NTTIs = 100; % number of TTIs/drops

if (PAR.Tx.Flag_UndoFilterResponse && (PAR.Rx.ZF ||
PAR.Rx.MF || PAR.Rx.MMSE))
    error('PAR.Tx.Flag_UndoFilterResponse and at
least one of the linear receivers is active. PAR.Tx.
Flag_UndoFilterResponse is only applicable to FFT
based Rx!')
end

%% Initialization
%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  basic parameters %%%
%%%%%%%%%%%%%%%%%%%%%%%%%

% number of samples per multicarrier symbol
if (PAR.lFIR == 1) % OFDM
    PARderived.lMCsym = PAR.FFTsize + PAR.CPlength;
else
    PARderived.lMCsym = PAR.FFTsize + PAR.lFIR -1;
end

switch PAR.DataSource
        case {'BPSK'}
            PARderived.Bit_per_Symbol = 1;
        case {'QPSK'}
            PARderived.Bit_per_Symbol = 2;
end

PARderived.nSNR = length(PAR.SNR_dB);

%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Signal generation  %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%

% OFDM CP and CP removal
% matrix for cyclic prefix addition
CPadd = zeros(PAR.CPlength+PAR.FFTsize,PAR.FFTsize);
CPadd(1:PAR.CPlength,(PAR.FFTsize-PAR.CPlength+1):
end) = diag(ones(1,PAR.CPlength));
CPadd((PAR.CPlength+1):end,:) = diag(ones(1,PAR.
FFTsize));
PARderived.Tx.CPadd=CPadd;
% matrix for cyclic prefix removal
CPrem = zeros(PAR.FFTsize,PAR.CPlength+PAR.FFTsize);
```

```
CPrem(:,(PAR.CPlength+1):end) = diag(ones(1,PAR.
FFTsize));
PARderived.Rx.CPrem=CPrem;

% Allocation widths
PARderived.nUsedCarr = PAR.nPRB*PAR.blockSize;
% allocated subcarriers
PARderived.allocatedSubcarriers = [1 : PARderived.
nUsedCarr] + PAR.blockShift;

% Generation of IDFT spreading matrices carrying
the relevant columns of the IDFT matrice with size
PAR.FFTsize
% Dimension of the matrices: [PAR.FFTsize x
PARderived.nUsedCarr]
PARderived.V = zeros(PAR.FFTsize,PARderived.
nUsedCarr);
for c = 1:PARderived.nUsedCarr   %loop through all
allocated subcarriers
    SubcarrierIndex=PARderived.allocatedSubcarriers
(c);
    PARderived.V([1:PAR.FFTsize],c) = exp(2*pi*1i*([
1:PAR.FFTsize]-1)*SubcarrierIndex/PAR.FFTsize); %
generation of the IDFT vector
end

% final multicarrier modulation matrix T
if PAR.lFIR == 1  % OFDM
    V=PARderived.V;
    T=(1/norm(V))*CPadd*V; %CP-OFDM = IDFT
spreading matrices plus CP addition
    PARderived.T = T;
else % UFMC
    f = chebwin(PAR.lFIR,PAR.FilterPar_dB); %Dolph-
Chebyshev
    % initialize helper matrices
    F_all = [];
    V_all = zeros(PAR.FFTsize*PAR.nPRB,PARderived.
nUsedCarr);
    for iPRB = 1:PAR.nPRB
        % shift to center carrier
        blockShift = PARderived.allocatedSubcarriers
(1)-1; %edge of the allocation
        carrierind = blockShift + (PAR.blockSize+1)/
2 + (iPRB-1)*PAR.blockSize; % center carrier
        centerFshift = zeros(PAR.lFIR,1);
        for k = 1:PAR.lFIR
            centerFshift(k) = exp(2*pi*1i*(k-1)
*carrierind/PAR.FFTsize);
        end
        % frequency-shifted FIR window
        f1 = f.*centerFshift;
        PARderived.Filterresponse_shifted{iPRB}=f1;
        % generate Toeplitz matrix for convolution
        F{iPRB} = toeplitz([f1;zeros(PAR.FFTsize-1,1
)],[f1(1),zeros(1,PAR.FFTsize-1)]);
        % stacked Toeplitz matrices implement
multicarrier modulation
        F_all = [F_all F{iPRB}];
        % generate expanded IDFT matrix
        V_all( (1+(iPRB-1)*PAR.FFTsize):(iPRB*PAR.
FFTsize), ...
        (1+(iPRB-1)*PAR.blockSize):(iPRB*PAR.
blockSize)) = ...
                                PARderived.V(:,(1+
(iPRB-1)*PAR.blockSize):(iPRB*PAR.blockSize));
    end
    T = F_all*V_all;
    % Final normalized multicarrier modulation
matrix
    TimeDomainSig=T*ones(PARderived.nUsedCarr,1);
    T=T/sqrt(mean(abs(TimeDomainSig).^2)/PARderived.
nUsedCarr*PAR.FFTsize);
    PARderived.T = T;
    %determine FreqResp in pass-band
    TimeDomainSig=T*ones(PARderived.nUsedCarr,1);
    FreqDomSig_oversampled=fft([TimeDomainSig.'
zeros(1,2*PAR.FFTsize-length(TimeDomainSig))])/sqrt
(PAR.FFTsize);
    FreqDomSig=FreqDomSig_oversampled(1:2:end);
    if PAR.Tx.Flag_UndoFilterResponse
        PARderived.PredistortionResponse=(FreqDomSig
(PARderived.allocatedSubcarriers+1)./(mean(abs
(FreqDomSig(PARderived.allocatedSubcarriers+1)))));
    else
        PARderived.PredistortionResponse=ones(1,
PARderived.nUsedCarr);
    end
end


%CFO generation (matrix-multiplication in time
domain), per Alloc and per Layer
T = PARderived.T; %Modulation matrix, perfectly
time and frequency alligned
Gamma = diag(exp((1j*2*pi*PAR.rCFO*(1:PARderived.
lMCsym))/PAR.FFTsize));
PARderived.Gamma = Gamma;
PARderived.GT = Gamma*T; % Modulation matrix
(Signal) and CFO matrix combined


%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Signal detection  %%%
%%%%%%%%%%%%%%%%%%%%%%%%%
% static receive filters (for AWGN)
```

```
V = PARderived.V;
Gamma = PARderived.Gamma;
T = PARderived.T;
GT = PARderived.GT;

if PAR.Rx.ZF %ZF active
    if ((PAR.CPlength ~= 0)&(PAR.lFIR == 1)) % OFDM
w/ CP
        if PAR.Rx.flag_CFOcomp_on == 0
            w_ZF = pinv((1/norm(V))*V);
        else
            w_ZF = pinv((1/norm(Gamma(PAR.CPlength+1
:end,PAR.CPlength+1:end)*V))*Gamma(PAR.CPlength+1:
end,PAR.CPlength+1:end)*V);
        end
        PARderived.w_ZF = w_ZF;
    else % UFMC
        if PAR.Rx.flag_CFOcomp_on == 0
            w_ZF = pinv(T);
        else
            w_ZF = pinv(GT);
        end
        PARderived.w_ZF = w_ZF;
    end
end
if PAR.Rx.MF % MF
    % OFDM with CP
    if ((PAR.CPlength ~= 0)&(PAR.lFIR == 1))
        if PAR.Rx.flag_CFOcomp_on == 0
            w_MF = (1/norm(V))*(V)';
        else
            w_MF = ((1/norm(Gamma(PAR.CPlength+1:
end,PAR.CPlength+1:end)*V))*Gamma(PAR.CPlength+1:
end,PAR.CPlength+1:end)*V)';
        end
        D = zeros(PARderived.nUsedCarr,PARderived.
nUsedCarr);
        for ii=1:PARderived.nUsedCarr
            D(ii,ii) = 1/(w_MF(ii,:)*w_MF(ii,:)');
        end
        PARderived.w_MF = D*w_MF;
     else%UFMC
        if PAR.Rx.flag_CFOcomp_on == 0
            w_MF = T';
        else
            w_MF = GT';
        end
        % normalize it
        %row-wise - each carrier must be normalized
        D = zeros(PARderived.nUsedCarr,PARderived.
nUsedCarr);
        for ii=1:PARderived.nUsedCarr
```

```
            D(ii,ii) = 1/(w_MF(ii,:)*w_MF(ii,:)');
        end
        PARderived.w_MF = D*w_MF;
    end
end
if PAR.Rx.MMSE %MMSE
    if ((PAR.CPlength ~= 0)&(PAR.lFIR == 1)) % OFDM
with CP
        V_normalized=1/norm(V)*V;
        GV = Gamma(PAR.CPlength+1:end,PAR.CPlength+1
:end)*V_normalized;
        for isnr = 1:PARderived.nSNR
            nvar = 1/(10^(0.1*PAR.SNR_dB(isnr))); %
perfect knowledge of noise power assumed
            if PAR.Rx.flag_CFOcomp_on == 0
                w_MMSE{isnr} = inv(V_normalized'*V_n
ormalized + nvar*diag(ones(size(V_normalized,2),1)))
*V_normalized';
            else
                w_MMSE{isnr} = inv(GV'*GV +
nvar*diag(ones(size(GV,2),1)))*GV';
            end
            PARderived.w_MMSE.SNR{isnr} = w_MMSE
{isnr};
        end
    else %UFMC
        for isnr = 1:PARderived.nSNR
            nvar = 1/(10^(0.1*PAR.SNR_dB(isnr))); %
perfect knowledge of noise power assumed
            if PAR.Rx.flag_CFOcomp_on == 0
                w_MMSE{isnr} = inv(T'*T + nvar*diag
(ones(size(T,2),1)))*T';
            else
                w_MMSE{isnr} = inv(GT'*GT +
nvar*diag(ones(size(GT,2),1)))*GT';
            end
            PARderived.w_MMSE.SNR{isnr} = w_MMSE
{isnr};
        end
    end
end
```



```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
PAR.Rx.ZF = 0; %ZF active?  1
PAR.Rx.MF = 0; %MF active?  1
PAR.Rx.MMSE = 0; %MMSE active?
PAR.Rx.FFTbasedRx = 1; %FFT ba
PAR.Rx.ChanEst = 'viaKnownSymb
PAR.Rx.flag_CFOcomp_on = 1; %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  Settings regarding synchr
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
PAR.rCFO = 0.0;%relative carri

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%   General settings  %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
PAR.SNR_dB = [0:10];
PAR.NsymbsperTTI = 14;% number
PAR.FFTsize = 1024;
PAR.lFIR = 74; % filter length
PAR.FilterPar_dB = 40; % sideb
PAR.blockSize = 12; % width of
PAR.CPlength = 73; % length of
PAR.NTTIs = 100; % number of T

if (PAR.Tx.Flag_UndoFilterResp
    error('PAR.Tx.Flag_UndoFil
end

%% Initialization
%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  basic parameters %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%

% number of samples per multic
if (PAR.lFIR == 1) % OFDM
```

2. Run the "Setup" equation; this will generate the variables needed for the simulation, especially the structure variables *PAR* and *PARderived*, in the Workspace Variables. Note that if retail MATLAB has not been launched before during the current SystemVue session, running the equation will take extra time to launch MATLAB.

3. In UFMC_OFDM___TransceiverChain.m, lines 296 to 444 are used for the main simulation loops. The top level loop starts at line 298

```
for isnr = 1:PARderived.nSNR
```

which loops over SNR (signal to noise ratio). The second level loop starts at line 305

```
for iTTI = 1:PAR.NTTIs
```

which loops over TTI (transmission time interval). The transmitter (Tx) part of the script is in lines 306 to 337 and the receiver (Rx) part of the script is in lines 342 to 442. Finally, the post-processing of the simulation results is done in lines 445 to 475. In SystemVue, the typical way to sweep over variables like SNR is to use Sweep. Data flow simulation control can be used to control the length of the simulation, which in this case is the number of TTIs. The following steps will guide you through the details.

4. To create a tunable variable for sweep, you can create a workspace tree equation called "TuneSNR", and copy lines 298 and 299 to the "TuneSNR" equation. Comment out the for loop and make *isnr* a tunable variable using the tune function, like the following script.

```
%for isnr = 1:PARderived.nSNR
isnr = tune(1); %isnr loop is moved to sweep
sqrt_nvar = 1/sqrt(10^(0.1*PAR.SNR_dB(isnr)));
```

5. By examining the variables between Tx part and Rx part of the script, you can find that the variables *s_orig* (original signal), *x* (Tx output signal without adding noise), and *y_pilots* (pilot signal assuming no distortion) are the outputs of the Tx part. The signal *y* received by Rx is actually a superposition of the noise signal *n* added to the Tx output signal *x*. In addition, variable *iEb* in line 442 of the original script

```
iEb((iTTI-1)*PAR.NsymbsperTTI+1:iTTI*PAR.
NsymbsperTTI,isnr) = diag(x'*x/(PARderived.
nUsedCarr*PARderived.Bit_per_Symbol));
```

can be treated as a Tx output because it is derived from *x* without any dependency on Rx part of the script. Based on this partition, you can start to create a Tx model in SystemVue. First, create a Schematic and name it "UFMC Design". Next, place an MATLAB_Script model on the schematic and name it "Tx". Double click on the Tx MATLAB_Script model, then copy and paste Tx part from lines 306 to 337 of the original script to the "Equations" tab. Comment out

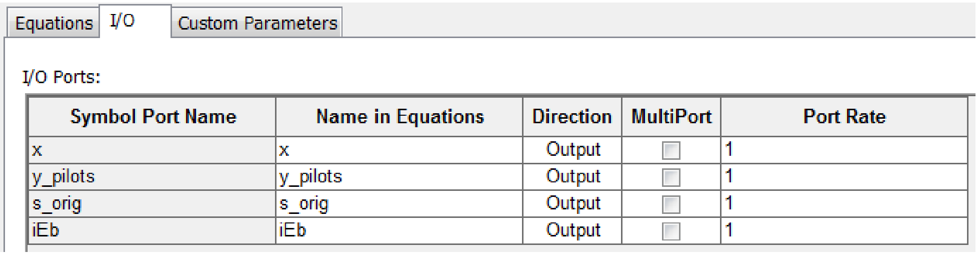
```
n = sqrt_nvar*(1/sqrt(2))*(randn(PARderived.lMCsym,
PAR.NsymbsperTTI)+j*randn(PARderived.lMCsym,PAR.
NsymbsperTTI));
y = x + n; %superimpose layers and add noise
```

in the "Equations" tab because noise addition will be handled in a separate MATLAB_Script model. Add line 442 from Rx part of the original script to the end of the "Equations" tab, as described above. After the above steps, the "Equations" tab of the Tx model should look like

```
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Symbol vector generation %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
        switch PAR.DataSource
            case {'BPSK'}  %BPSK
                s_orig = sign(randn(PARderived.
nUsedCarr,PAR.NsymbsperTTI));
            case {'QPSK'}  %QPSK
                s_orig = (1/sqrt(2))*(sign(randn
(PARderived.nUsedCarr,PAR.NsymbsperTTI))+j*sign
(randn(PARderived.nUsedCarr,PAR.NsymbsperTTI)));
        end
        %s_pilots=s_orig; % ideal chanest so far,
each data symbol known and used as pilot

        %Predistortion to undo FilterResponse

        if (PAR.lFIR ~= 1 && PAR.Tx.
Flag_UndoFilterResponse && ~(PAR.Rx.ZF || PAR.Rx.MF
|| PAR.Rx.MMSE))%UFMC
            s = s_orig./repmat(PARderived.
PredistortionResponse.',1,PAR.NsymbsperTTI);
            s_pilots = s_orig./repmat(PARderived.
PredistortionResponse.',1,PAR.NsymbsperTTI);
        else
            s=s_orig;
            s_pilots=s_orig;
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Transformation to time domain %
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                      %delay raus
        x = PARderived.GT*s; %signal of user of
interest including CFO (G)
        x_pilots = PARderived.T*s_pilots; %pilots
of user of interest (for ideal chanest, w/o CFO as
CFO compensated separately in time domain)

        % add noise
        %n = sqrt_nvar*(1/sqrt(2))*(randn
(PARderived.lMCsym,PAR.NsymbsperTTI)+j*randn
(PARderived.lMCsym,PAR.NsymbsperTTI));
        %y = x + n; %superimpose layers and add
noise
        y_pilots = x_pilots;

        %moved from Rx part of code
        iEb = diag(x'*x/(PARderived.
nUsedCarr*PARderived.Bit_per_Symbol));
```

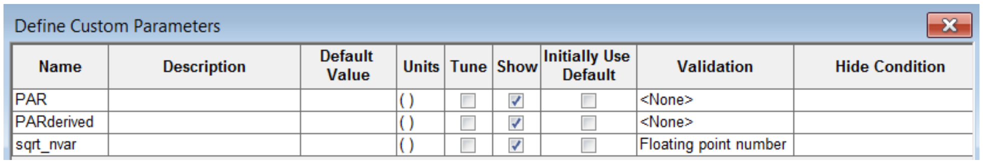
Once the equation of the Tx model is completed, go to the "I/O" tab of the Tx model and add the following output ports: *x*, *y_pilots*, *s_orig*, and *iEb*.



I/O Ports:

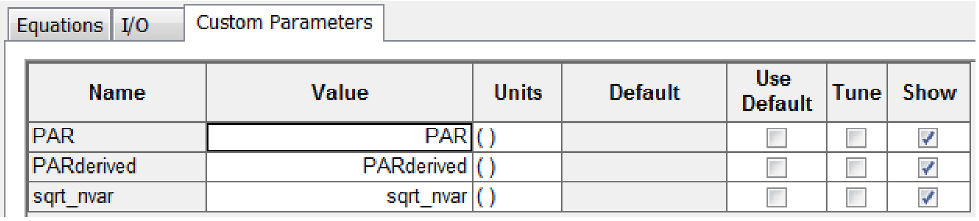
| Symbol Port Name | Name in Equations | Direction | MultiPort | Port Rate |
|---|---|---|---|---|
| x | x | Output | □ | 1 |
| y_pilots | y_pilots | Output | □ | 1 |
| s_orig | s_orig | Output | □ | 1 |
| iEb | iEb | Output | □ | 1 |

With the I/O specification, the variables *x*, *y_pilots*, *s_orig*, and *iEb* after the evaluation of the Tx equation will be transferred to the output ports of the Tx model. The Tx equation also needs variables *PAR*, *PARderived*, and *sqrt_nvar* for evaluation. These variables can be passed in through custom parameters. Go to the "Custom Parameters" tab, click on the "Define Custom Parameters..." button, add parameters *PAR*, *PARderived*, and *sqrt_nvar* in the "Define Custom Parameters" dialog. Because parameters *PAR* and *PARderived* are structured, set the "Validation" column of these two parameters to <None>.



| Name | Description | Default Value | Units | Tune | Show | Initially Use Default | Validation | Hide Condition |
|---|---|---|---|---|---|---|---|---|
| PAR | | | ( ) | □ | ☑ | □ | <None> | |
| PARderived | | | ( ) | □ | ☑ | □ | <None> | |
| sqrt_nvar | | | ( ) | □ | ☑ | □ | Floating point number | |

Click "OK" to save the custom parameter definition. In the "Custom Parameters" tab of the Tx model, set the value of parameters *PAR*, *PARderived*, and *sqrt_nvar* to variables *PAR*, *PARderived*, and *sqrt_nvar* obtained from the Workspace Variables respectively.



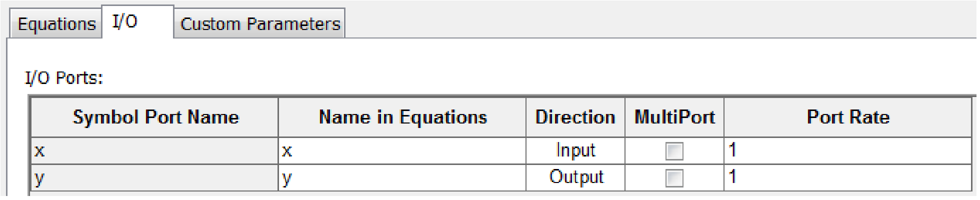
| Name | Value | Units | Default | Use Default | Tune | Show |
|---|---|---|---|---|---|---|
| PAR | PAR | ( ) | | □ | □ | ☑ |
| PARderived | PARderived | ( ) | | □ | □ | ☑ |
| sqrt_nvar | sqrt_nvar | ( ) | | □ | □ | ☑ |

Click "OK" in the Tx model properties dialog. The setup for the Tx model is now completed.

6. To model the noise channel, place an MATLAB_Script model on the "UFMC Design" schematic, name it "Channel". In the "Equations" tab of the Channel model, you can copy and paste lines 334 to 336 directly from the original script.

```
% add noise
n = sqrt_nvar*(1/sqrt(2))*(randn(PARderived.lMCsym,
PAR.NsymbsperTTI)+j*randn(PARderived.lMCsym,PAR.
NsymbsperTTI));
y = x + n; %superimpose layers and add noise
```

Then in the "I/O" tab of the Channel model, specify *x* as Input port and *y* as Output port.



I/O Ports:

| Symbol Port Name | Name in Equations | Direction | MultiPort | Port Rate |
|---|---|---|---|---|
| x | x | Input | ☐ | 1 |
| y | y | Output | ☐ | 1 |

This equation also needs variables *PAR*, *PARderived*, and *sqrt_nvar*. Go to the "Custom Parameters" tab of the Channel model, create and set the parameters *PAR*, *PARderived*, and *sqrt_nvar* in the same way as the customer parameters in the Tx model.

7. By examing Rx part of the script, you can find that the input signals for Rx are *y*, *y_pilots*, and *s_orig* (for error computation). The outputs of Rx are *SE_ZF*, *symErr_ZF*, *SE_MF*, *symErr_MF*, *SE_MMSE*, *symErr_MMSE*, *SE_FFT*, and *symErr_FFT*. The presence of these outputs depends on whether the corresponding settings (*PAR.Rx.ZF*, *PAR.Rx.MF*, *PAR.Rx.MMSE*, *PAR.Rx.FFTbasedRx*) are turned on. If you run the original script in MATLAB or examine the script,

```
SE_FFT(:,(iTTI-1)*PAR.NsymbsperTTI+1:iTTI*PAR.
NsymbsperTTI,isnr) = abs(s_orig-s_est_FFT).^2;
symErr_FFT(:,(iTTI-1)*PAR.NsymbsperTTI+1:iTTI*PAR.
NsymbsperTTI,isnr) = sum(s_HD_FFT~=s_orig);
```

there is a matrix (so-called "subframe") of size (*PAR.nPRB * PAR.blockSize) x PAR.NsymbsperTTI* to be appended to *SE_{ZF, MF, MMSE, FFT}* for each TTI. Similarly, there is a matrix of size 1 x *PAR.NsymbsperTTI* to be appended to *symErr_{ZF, MF, MMSE, FFT}* for each TTI. The matrices generated by looping over the TTIs are appended in the second dimension, and the bigger matrices generated by looping over the SNRs are appended in the third dimension. Therefore, in the original script, the size of *SE_{ZF, MF, MMSE, FFT}* after simulation is *(PAR.nPRB * PAR.blockSize*) x *(PAR.NsymbsperTTI * PAR.NTTIs)* x *PARderived.nSNR* and the size of *symErr_{ZF, MF, MMSE, FFT}* after simulation is 1 x *(PAR.NsymbsperTTI * PAR.NTTIs) x PARderived.nSNR*. In SystemVue, because the SNR loop is moved to Sweep and the TTI loop is moved to data flow simulation control, the Rx model only needs to output the matrices generated for each TTI.

```
SE_FFT = abs(s_orig-s_est_FFT).^2;
symErr_FFT = sum(s_HD_FFT~=s_orig);
```

Based on this analysis, you can start to create a Rx model in SystemVue. First, place an MATLAB_Script model on the "UFMC Design" schematic, name it "Rx". Then copy and paste lines 341 to 441 from the original script to the "Equations" tab of the Rx model with the following modifications:

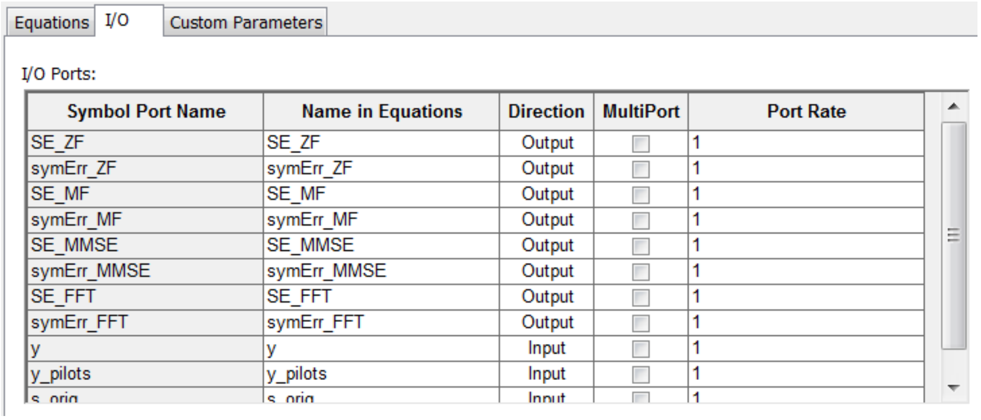
a) comment out lines that create *SE_{ZF, MF, MMSE, FFT}* and *symErr_{ZF, MF, MMSE, FFT}* (corresponding to lines 426 to 441 in the original script), and replace them with the last section of the code below

b) comment out the line that creates *sH_FFT_tmp(:,iTTI)* (corresponding to line 414 in the original script); this is because the loop index *iTTI* is moved to the data flow simulation control and *sH_FFT_tmp* is not used anywhere in the Rx part of the script

```
        % CP removal
        if (PAR.lFIR == 1) % OFDM
            y = PARderived.Rx.CPrem*y;
        end
        %% obtain symbol estimates
        for iNsymbsperTTI=1:PAR.NsymbsperTTI
            % ZF
            if PAR.Rx.ZF %ZF active?
                w_ZF = PARderived.w_ZF;
                s_est_ZF(:,iNsymbsperTTI) = w_ZF*y
(:,iNsymbsperTTI);
                % hard decision
                switch PARderived.Bit_per_Symbol
                    case {1}
                        s_HD_ZF(:,iNsymbsperTTI) =
sign(real(s_est_ZF(:,iNsymbsperTTI)));
                    case {2}
                        s_HD_ZF(:,iNsymbsperTTI) = (
1/sqrt(2))*(sign(real(s_est_ZF(:,iNsymbsperTTI))) +
j*sign(imag(s_est_ZF(:,iNsymbsperTTI))));
                end
            end
            % MF
            if PAR.Rx.MF %MF active?
                w_MF = PARderived.w_MF;
                s_est_MF(:,iNsymbsperTTI) = w_MF*y
(:,iNsymbsperTTI);
                % hard decision
                switch PARderived.Bit_per_Symbol
                    case {1}
                        s_HD_MF(:,iNsymbsperTTI) =
sign(real(s_est_MF(:,iNsymbsperTTI)));
                    case {2}
                        s_HD_MF(:,iNsymbsperTTI) = (
1/sqrt(2))*(sign(real(s_est_MF(:,iNsymbsperTTI))) +
j*sign(imag(s_est_MF(:,iNsymbsperTTI))));
                end
            end
            % MMSE
            if PAR.Rx.MMSE %MMSE active?
                w_MMSE{isnr} = PARderived.w_MMSE.SNR
{isnr};
                s_est_MMSE(:,iNsymbsperTTI) = w_MMSE
{isnr}*y(:,iNsymbsperTTI);
                % hard decision
                switch PARderived.Bit_per_Symbol
                    case {1}
                        s_HD_MMSE(:,iNsymbsperTTI)
= sign(real(s_est_MMSE(:,iNsymbsperTTI)));
                    case {2}
                        s_HD_MMSE(:,iNsymbsperTTI)
= (1/sqrt(2))*(sign(real(s_est_MMSE(:,
iNsymbsperTTI))) + j*sign(imag(s_est_MMSE(:,
iNsymbsperTTI))));
                end
            end
            % UFMC FFT based detection
            if (PAR.lFIR ~= 1 && PAR.Rx.FFTbasedRx)
%UFMC with FFT based detection active
                if PAR.Rx.flag_CFOcomp_on == 1 %
undo CFO
                    y(:,iNsymbsperTTI)  = y(:,
iNsymbsperTTI).*exp(-(1j*2*pi*PAR.rCFO*(1:
PARderived.lMCsym))/PAR.FFTsize).';
                end
                switch PAR.Rx.ChanEst %Frequency
domain estimation. Estimation of the frequency
response of the filters in pass-band.
                    case {'viaKnownSymbs'} %all QAM
symbols used as pilots
                        y_pilots_padded=[y_pilots(:,
iNsymbsperTTI).' zeros(1,2048-length(y_pilots))];
                        H_oversampled = fft
(y_pilots_padded)/sqrt(PAR.FFTsize);
                        H = H_oversampled(1:2:end);
                        H_Alloc = H(PARderived.
allocatedSubcarriers+1)./s_orig(:,iNsymbsperTTI).';
                end
                y_padded=[y(:,iNsymbsperTTI).' zeros
(1,2048-length(y(:,iNsymbsperTTI)))];
                s_oversampled=fft(y_padded)/sqrt
(PAR.FFTsize);
                s_Rx=s_oversampled(1:2:end); %all
subcarriers unequalized
                s_est_FFT(:,iNsymbsperTTI) = s_Rx
(PARderived.allocatedSubcarriers+1)./H_Alloc;
                switch PARderived.Bit_per_Symbol
                    case {1} %BPSK
                        s_HD_FFT(:,iNsymbsperTTI) =
sign(real(s_est_FFT(:,iNsymbsperTTI)));
                    case {2} %QPSK
```

```
                        s_HD_FFT(:,iNsymbsperTTI) =
(1/sqrt(2))*(sign(real(s_est_FFT(:,iNsymbsperTTI)))
+ j*sign(imag(s_est_FFT(:,iNsymbsperTTI))));
                end
                % OFDM FFT based
            elseif (PAR.lFIR == 1 && PAR.Rx.
FFTbasedRx) %OFDM with FFT based detection active
                if PAR.Rx.flag_CFOcomp_on == 1 %
undo CFO
                    y(:,iNsymbsperTTI)  = y(:,
iNsymbsperTTI).*exp(-(1j*2*pi*PAR.rCFO*(PAR.
CPlength+1:PARderived.lMCsym))/PAR.FFTsize).';
                end
                s_Rx=(fft(y(:,iNsymbsperTTI))/sqrt
(PAR.FFTsize)).';
                %generate 8 times oversampled
frequency sig.
%               sH_FFT_tmp(:,iTTI)=(fft([y(:,
iNsymbsperTTI).' zeros(1,7*PAR.FFTsize)])/sqrt(PAR.
FFTsize)).'; %TTI is moved to data flow simulation
control
                %
                s_est_FFT(:,iNsymbsperTTI) = s_Rx
(PARderived.allocatedSubcarriers+1);
                switch PARderived.Bit_per_Symbol
                    case {1} %BPSK
                        s_HD_FFT(:,iNsymbsperTTI) =
sign(real(s_est_FFT(:,iNsymbsperTTI)));
                    case {2} %QPSK
                        s_HD_FFT(:,iNsymbsperTTI) =
(1/sqrt(2))*(sign(real(s_est_FFT(:,iNsymbsperTTI)))
+ j*sign(imag(s_est_FFT(:,iNsymbsperTTI))));
                end
            end
        end
        % track MSE and raw SER
%         if (PAR.Rx.ZF)
%             SE_ZF(:,(iTTI-1)*PAR.NsymbsperTTI+1:
iTTI*PAR.NsymbsperTTI,isnr) = abs(s_orig-s_est_ZF).^
2;
%             symErr_ZF(:,(iTTI-1)*PAR.NsymbsperTTI+1
:iTTI*PAR.NsymbsperTTI,isnr) = sum(s_HD_ZF~=s_orig);
%         end
%         if (PAR.Rx.MF)
%             SE_MF(:,(iTTI-1)*PAR.NsymbsperTTI+1:
iTTI*PAR.NsymbsperTTI,isnr) = abs(s_orig-s_est_MF).^
2;
%             symErr_MF(:,(iTTI-1)*PAR.NsymbsperTTI+1
:iTTI*PAR.NsymbsperTTI,isnr) = sum(s_HD_MF~=s_orig);
%         end
%         if (PAR.Rx.MMSE)
%             SE_MMSE(:,(iTTI-1)*PAR.NsymbsperTTI+1:
iTTI*PAR.NsymbsperTTI,isnr) = abs(s_orig-
s_est_MMSE).^2;
%             symErr_MMSE(:,(iTTI-1)*PAR.
NsymbsperTTI+1:iTTI*PAR.NsymbsperTTI,isnr) = sum
(s_HD_MMSE~=s_orig);
%         end
%         if (PAR.Rx.FFTbasedRx) %UFMC with FFT
based detection active
%             SE_FFT(:,(iTTI-1)*PAR.NsymbsperTTI+1:
iTTI*PAR.NsymbsperTTI,isnr) = abs(s_orig-s_est_FFT).
^2;
%             symErr_FFT(:,(iTTI-1)*PAR.NsymbsperTTI+
1:iTTI*PAR.NsymbsperTTI,isnr) = sum
(s_HD_FFT~=s_orig);
%         end

        if (PAR.Rx.ZF)
            SE_ZF = abs(s_orig-s_est_ZF).^2;
            symErr_ZF = sum(s_HD_ZF~=s_orig);
        else
            SE_ZF = [];
            symErr_ZF = [];
        end
        if (PAR.Rx.MF)
            SE_MF = abs(s_orig-s_est_MF).^2;
            symErr_MF = sum(s_HD_MF~=s_orig);
        else
            SE_MF = [];
            symErr_MF = [];
        end
        if (PAR.Rx.MMSE)
            SE_MMSE = abs(s_orig-s_est_MMSE).^2;
            symErr_MMSE = sum(s_HD_MMSE~=s_orig);
        else
            SE_MMSE = [];
            symErr_MMSE = [];
        end
        if (PAR.Rx.FFTbasedRx) %UFMC with FFT based
detection active
            SE_FFT = abs(s_orig-s_est_FFT).^2;
            symErr_FFT = sum(s_HD_FFT~=s_orig);
        else
            SE_FFT = [];
            symErr_FFT = [];
        end
```

Once the "Equations" tab is ready, go to the "I/O" tab, specify *SE_ZF*, *symErr_ZF*, *SE_MF*, *symErr_MF*, *SE_MMSE*, *symErr_MMSE*, *SE_FFT*, and *symErr_FFT* as Output ports, and specify *y*, *y_pilots*, and *s_orig* as Input ports.

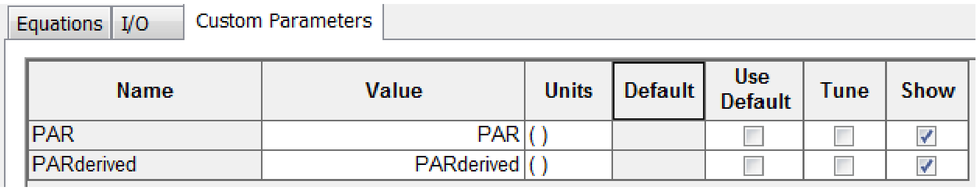


Equations | I/O | Custom Parameters

I/O Ports:

| Symbol Port Name | Name in Equations | Direction | MultiPort | Port Rate |
|---|---|---|---|---|
| SE_ZF | SE_ZF | Output | □ | 1 |
| symErr_ZF | symErr_ZF | Output | □ | 1 |
| SE_MF | SE_MF | Output | □ | 1 |
| symErr_MF | symErr_MF | Output | □ | 1 |
| SE_MMSE | SE_MMSE | Output | □ | 1 |
| symErr_MMSE | symErr_MMSE | Output | □ | 1 |
| SE_FFT | SE_FFT | Output | □ | 1 |
| symErr_FFT | symErr_FFT | Output | □ | 1 |
| y | y | Input | □ | 1 |
| y_pilots | y_pilots | Input | □ | 1 |
| s_orig | s_orig | Input | □ | 1 |

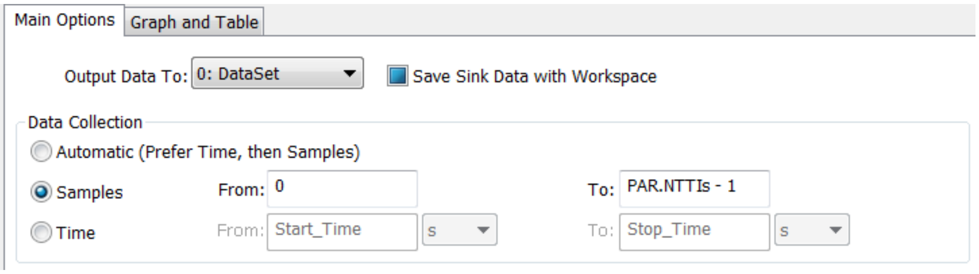
The Rx equation also needs variables *PAR* and *PARderived*, so create custom parameters *PAR* and *PARderived* the same way as the Tx model.



Equations | I/O | Custom Parameters

| Name | Value | Units | Default | Use Default | Tune | Show |
|---|---|---|---|---|---|---|
| PAR | PAR | ( ) | | □ | □ | ☑ |
| PARderived | PARderived | ( ) | | □ | □ | ☑ |

8. For data collection and simulation control, create nine Sink models on the "UFMC Design" schematic, name them "iEb", "SE_ZF", "symErr_ZF", "SE_MF", "symErr_MF", "SE_MMSE", "symErr_MMSE", "SE_FFT", and "symErr_FFT" respectively. Set data collection mode of each sink to use "Samples" and collect samples from 0 to *PAR.NTTIs* - 1. This setup will make the Data Flow simulator keep running the design until each sink collects *PAR. NTTIs* number of matrices.



Main Options | Graph and Table

Output Data To: 0: DataSet ☑ Save Sink Data with Workspace

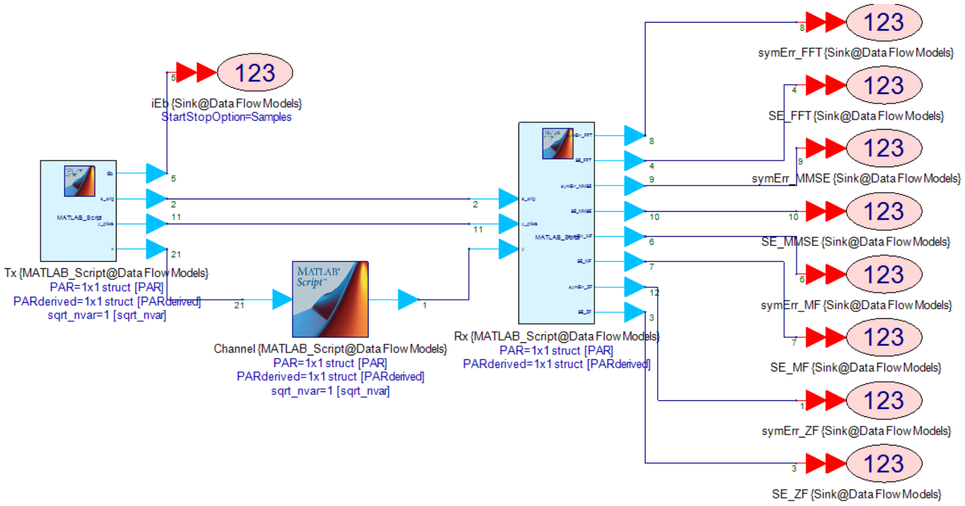
Data Collection

○ Automatic (Prefer Time, then Samples)

● Samples From: 0 To: PAR.NTTIs - 1

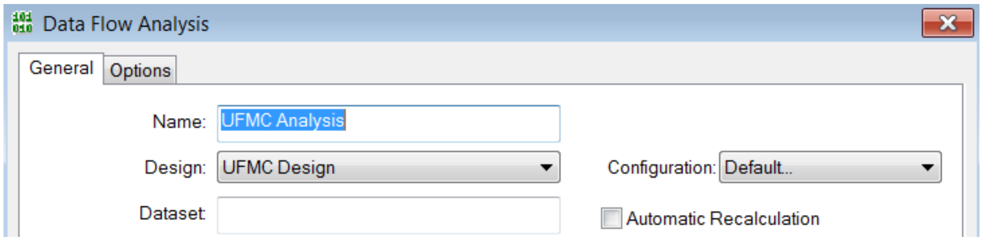
○ Time From: Start_Time s To: Stop_Time s

9. On the "UFMC Design" schematic, connect output ports *s_orig* and *y_pilots* of the Tx model to input ports *s_orig* and *y_pilots* of the Rx model, respectively. Connect output port *x* of the Tx model to input port *x* of the Channel model and connect output port *y* of the Channel model to input port *y* of the Rx model. Connect output port *iEb* of the Tx model to the "iEb" Sink and connect output ports *SE_ZF, symErr_ZF, SE_MF, symErr_MF, SE_MMSE, symErr_MMSE, SE_FFT,* and *symErr_FFT* of the Rx model to the corresponding "SE_ZF", "symErr_ZF", "SE_MF", "symErr_MF", "SE_MMSE", "symErr_MMSE", "SE_FFT", and "symErr_FFT" Sinks.



123
iEb {Sink@Data Flow Models}
StartStopOption=Samples
Tx {MATLAB_Script@Data Flow Models}
PAR=1x1 struct [PAR]
PARderived=1x1 struct [PARderived]
sqrt_nvar=1 [sqrt_nvar]
MATLAB Script
Channel {MATLAB_Script@Data Flow Models}
PAR=1x1 struct [PAR]
PARderived=1x1 struct [PARderived]
sqrt_nvar=1 [sqrt_nvar]
Rx {MATLAB_Script@Data Flow Models}
PAR=1x1 struct [PAR]
PARderived=1x1 struct [PARderived]
symErr_FFT {Sink@Data Flow Models}
SE_FFT {Sink@Data Flow Models}
symErr_MMSE {Sink@Data Flow Models}
SE_MMSE {Sink@Data Flow Models}
symErr_MF {Sink@Data Flow Models}
SE_MF {Sink@Data Flow Models}
symErr_ZF {Sink@Data Flow Models}
SE_ZF {Sink@Data Flow Models}

10. Create a Data Flow Analysis in the workspace tree, name it "UFMC Analysis" and set the Design to "UFMC Design".



Data Flow Analysis

General | Options

Name: UFMC Analysis

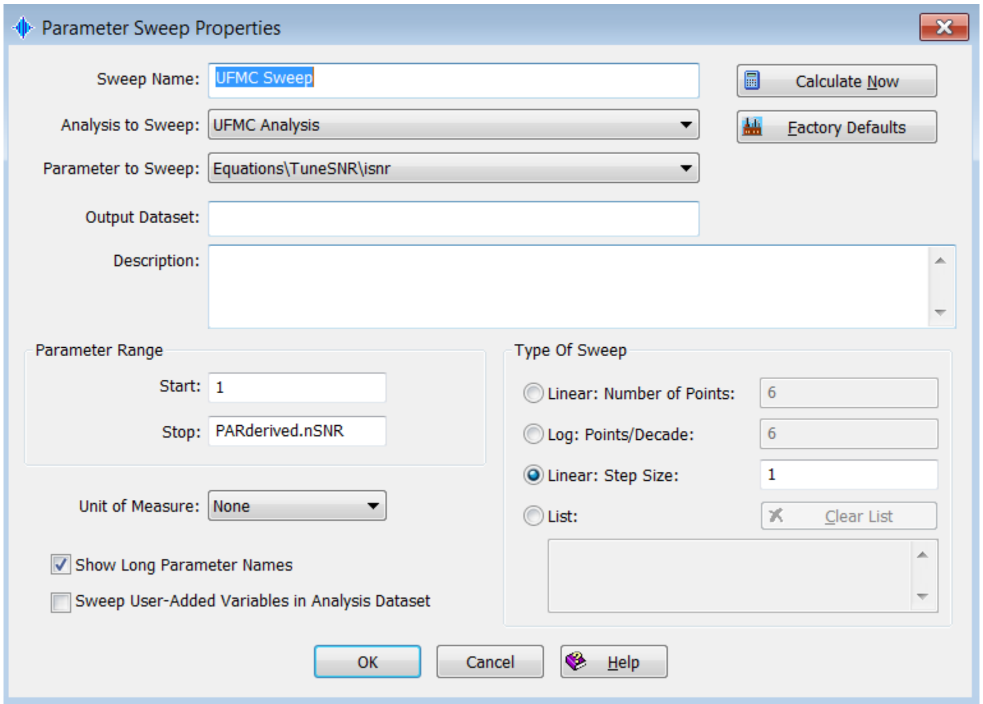
Design: UFMC Design

Configuration: Default...

Dataset:

□ Automatic Recalculation

11. Create a Sweep in the workspace tree, name it "UFMC Sweep". Select "Analysis to Sweep" to "UFMC Analysis" and select "Parameters to Sweep" to "Equations\TuneSNR\isnr". Set "Parameter Range" from 1 to *PARderived. nSNR* and make it a Linear sweep with Step Size equal to 1. The sweep will handle the SNR loop in the original script.



Parameter Sweep Properties

Sweep Name: UFMC Sweep

Analysis to Sweep: UFMC Analysis

Parameter to Sweep: Equations\TuneSNR\isnr

Output Dataset:

Description:

Parameter Range

Start: 1

Stop: PARderived.nSNR

Unit of Measure: None

☑ Show Long Parameter Names

□ Sweep User-Added Variables in Analysis Dataset

Type Of Sweep

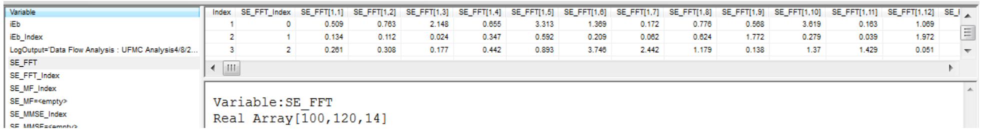
○ Linear: Number of Points: 6

○ Log: Points/Decade: 6

● Linear: Step Size: 1

○ List:

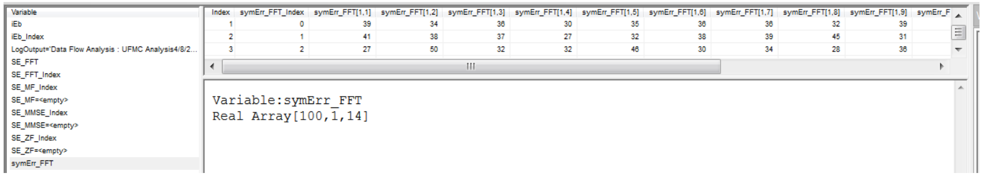
12. If you run "UFMC Analysis", you will get dataset variables *SE_ZF, symErr_ZF, SE_MF, symErr_MF, SE_MMSE, symErr_MMSE, SE_FFT,* and *symErr_FFT* in the "UFMC Analysis_UFMC Design_Data" dataset. Note that with the default settings, only *SE_*FFT and *symErr_FFT* are available. The size of *SE_{ZF, MF, MMSE, FFT}* in the analysis dataset is *PAR.NTTIs* x (*PAR.nPRB * PAR.blockSize*) x *PAR.NsymbsperTTI*, because Data Flow analysis creates a new dimension first and appends the collected data (matrices with size *PAR.nPRB x PAR.blockSize* x *PAR.NsymbsperTTI* ) in the first dimension. Similarly, the size of *symErr_{ZF, MF, MMSE, FFT}* in the analysis dataset is *PAR.NTTIs* x 1 x *PAR.NsymbsperTTI* because each collected data is a matrix with size 1 x *PAR.NsymbsperTTI*.



Variable
iEb
iEb_Index
LogOutput='Data Flow Analysis : UFMC Analysis4/8/2...
SE_FFT
SE_FFT_Index
SE_MF_Index
SE_MF=<empty>
SE_MMSE_Index

| Index | SE_FFT_Index | SE_FFT[1,1] | SE_FFT[1,2] | SE_FFT[1,3] | SE_FFT[1,4] | SE_FFT[1,5] | SE_FFT[1,6] | SE_FFT[1,7] | SE_FFT[1,8] | SE_FFT[1,9] | SE_FFT[1,10] | SE_FFT[1,11] | SE_FFT[1,12] | SE_I |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0.509 | 0.763 | 2.148 | 0.655 | 3.313 | 1.369 | 0.172 | 0.776 | 0.568 | 3.619 | 0.163 | 1.069 | |
| 2 | 1 | 0.134 | 0.112 | 0.024 | 0.347 | 0.592 | 0.209 | 0.062 | 0.624 | 1.772 | 0.279 | 0.039 | 1.972 | |
| 3 | 2 | 0.261 | 0.308 | 0.177 | 0.442 | 0.893 | 3.746 | 2.442 | 1.179 | 0.138 | 1.37 | 1.429 | 0.051 | |

```
Variable:SE_FFT
Real Array[100,120,14]
```



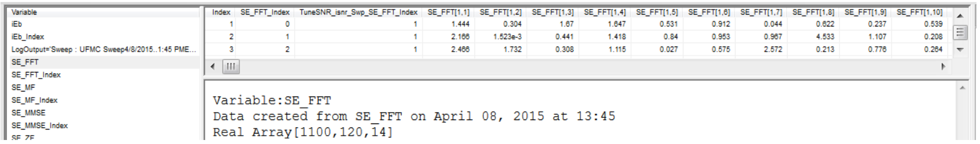
Variable
iEb
iEb_Index
LogOutput='Data Flow Analysis : UFMC Analysis4/8/2...
SE_FFT
SE_FFT_Index
SE_MF_Index
SE_MF=<empty>
SE_MMSE_Index
SE_MMSE=<empty>
SE_ZF_Index
SE_ZF=<empty>
symErr_FFT

| Index | symErr_FFT_Index | symErr_FFT[1,1] | symErr_FFT[1,2] | symErr_FFT[1,3] | symErr_FFT[1,4] | symErr_FFT[1,5] | symErr_FFT[1,6] | symErr_FFT[1,7] | symErr_FFT[1,8] | symErr_FFT[1,9] | symErr_F |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 39 | 34 | 36 | 30 | 35 | 36 | 36 | 32 | 39 | |
| 2 | 1 | 41 | 38 | 37 | 27 | 32 | 38 | 39 | 45 | 31 | |
| 3 | 2 | 27 | 50 | 32 | 32 | 46 | 30 | 34 | 28 | 36 | |

```
Variable:symErr_FFT
Real Array[100,1,14]
```

13. If you run "UFMC Sweep", you can observe that the sweep dataset variable *SE_{ZF, MF, MMSE, FFT}* from the sweep dataset *"*UFMC Sweep_Data" has size (*PARderived.nSNR * PAR.NTTIs)* x *PAR.nPRB x (PAR.blockSize* by *PAR.NsymbsperTTI).* This is because the sweep simulation appends the analysis dataset variable for each sweep point in the same first dimension. Similarly, the size of *symErr_{ZF, MF, MMSE, FFT}* in the sweep dataset is *(PARderived.*

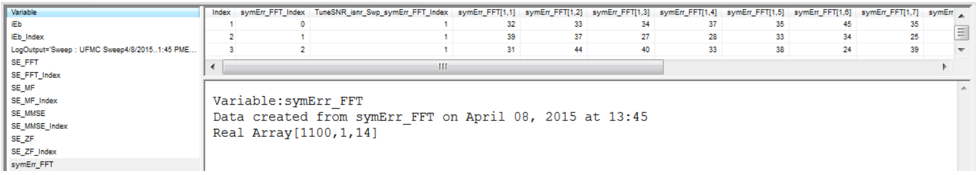
*nSNR * PAR.NTTIs)* x 1 x *PAR.NsymbsperTTI.*



Variable
iEb
iEb_Index
LogOutput='Sweep : UFMC Sweep4/8/2015..1:45 PME...
SE_FFT
SE_FFT_Index
SE_MF
SE_MF_Index
SE_MMSE
SE_MMSE_Index
SE_ZF

| Index | SE_FFT_Index | TuneSNR_isnr_Swp_SE_FFT_Index | SE_FFT[1,1] | SE_FFT[1,2] | SE_FFT[1,3] | SE_FFT[1,4] | SE_FFT[1,5] | SE_FFT[1,6] | SE_FFT[1,7] | SE_FFT[1,8] | SE_FFT[1,9] | SE_FFT[1,10] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1.444 | 0.304 | 1.67 | 1.647 | 0.531 | 0.912 | 0.044 | 0.622 | 0.237 | 0.539 |
| 2 | 1 | 1 | 2.166 | 1.523e-3 | 0.441 | 1.418 | 0.84 | 0.953 | 0.967 | 4.533 | 1.107 | 0.208 |
| 3 | 2 | 1 | 2.466 | 1.732 | 0.308 | 1.115 | 0.027 | 0.575 | 2.572 | 0.213 | 0.776 | 0.264 |

```
Variable:SE_FFT
Data created from SE_FFT on April 08, 2015 at 13:45
Real Array[1100,120,14]
```



Variable
iEb
iEb_Index
LogOutput='Sweep : UFMC Sweep4/8/2015..1:45 PME...
SE_FFT
SE_FFT_Index
SE_MF
SE_MF_Index
SE_MMSE
SE_MMSE_Index
SE_ZF
SE_ZF_Index
symErr_FFT

| Index | symErr_FFT_Index | TuneSNR_isnr_Swp_symErr_FFT_Index | symErr_FFT[1,1] | symErr_FFT[1,2] | symErr_FFT[1,3] | symErr_FFT[1,4] | symErr_FFT[1,5] | symErr_FFT[1,6] | symErr_FFT[1,7] | symErr |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 32 | 33 | 34 | 37 | 35 | 45 | 35 | |
| 2 | 1 | 1 | 39 | 37 | 27 | 28 | 33 | 34 | 25 | |
| 3 | 2 | 1 | 31 | 44 | 40 | 33 | 38 | 24 | 39 | |

```
Variable:symErr_FFT
Data created from symErr_FFT on April 08, 2015 at 13:45
Real Array[1100,1,14]
```

14. With the understanding of how SystemVue stores the swept data and comparing it with how the data is stored in the original script, it is straightforward to modify the post processing part (lines 446 to 475 of the original script) to handle the sweep dataset inside SystemVue. For example, line 458

```
RES.MSE_FFT(isnr) = mean(mean(squeeze(SE_FFT(:,:,
isnr))));
```

of the original script should be changed to

```
RES.MSE_FFT(isnr) = mean(mean(mean(squeeze(SE_FFT
(PAR.NTTIs*(isnr-1)+1:PAR.NTTIs*isnr,:,:)))));
```

in SystemVue. Similarly, line 473

```
RES.SER_FFT(isnr)= sum(sum(squeeze(symErr_FFT(:,:,
isnr)))) / (PAR.NTTIs*PAR.NsymbsperTTI*PARderived.
nUsedCarr);
```

of the original script should be changed to

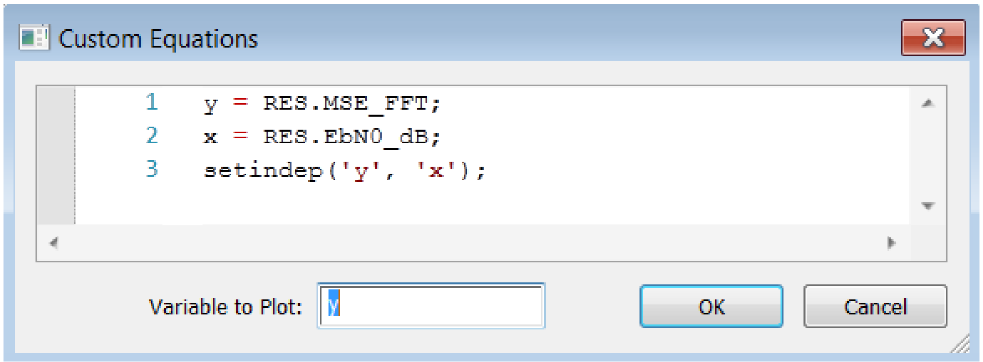
```
RES.SER_FFT(isnr)= sum(sum(squeeze(symErr_FFT(PAR.
NTTIs*(isnr-1)+1:PAR.NTTIs*isnr,:,:)))) / (PAR.
NTTIs*PAR.NsymbsperTTI*PARderived.nUsedCarr);
```

in SystemVue.

Now you can create a workspace tree equation, name it "PostProcessing", copy and paste the following modified code in the equation, and turn off "Auto-calculate".

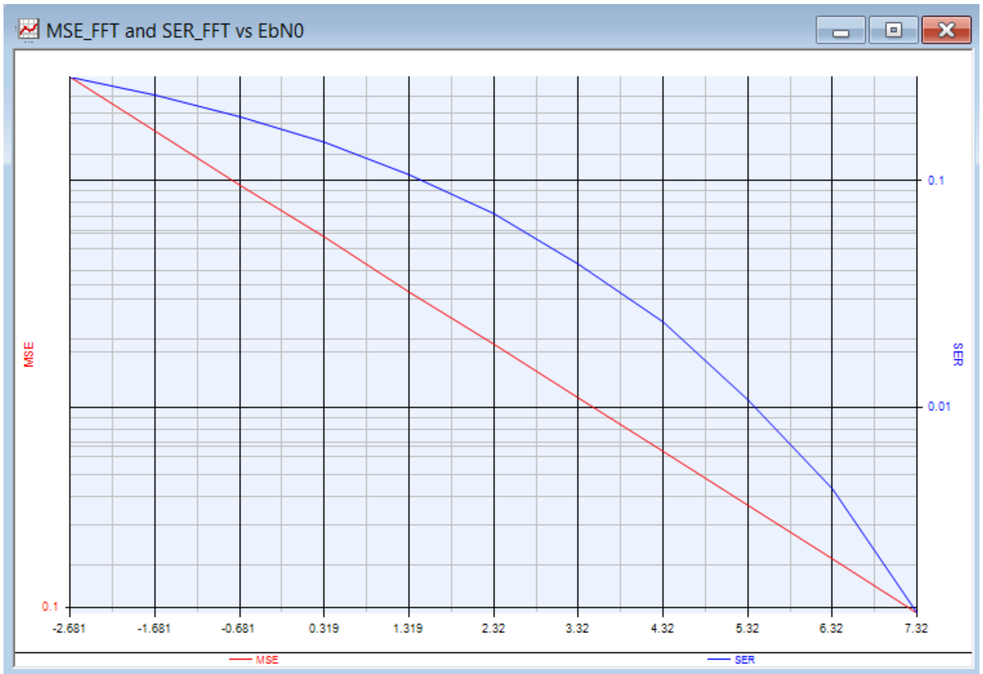
```
using('UFMC Sweep_Data'); %using sweep dataset

%% compute simulation results (SER, EbN0 and MSE)
for isnr = 1:PARderived.nSNR
 % MSE
    if  (PAR.Rx.ZF)
        RES.MSE_ZF(isnr) = mean(mean(mean(squeeze
(SE_ZF(PAR.NTTIs*(isnr-1)+1:PAR.NTTIs*isnr,:,:)))));
    end
    if  (PAR.Rx.MF)
        RES.MSE_MF(isnr) = mean(mean(mean(squeeze
(SE_MF(PAR.NTTIs*(isnr-1)+1:PAR.NTTIs*isnr,:,:)))));
    end
    if  (PAR.Rx.MMSE)
        RES.MSE_MMSE(isnr) = mean(mean(mean(squeeze
(SE_MMSE(PAR.NTTIs*(isnr-1)+1:PAR.
NTTIs*isnr,:,:)))));
    end
    if (PAR.Rx.FFTbasedRx)
       RES.MSE_FFT(isnr) = mean(mean(mean(squeeze
(SE_FFT(PAR.NTTIs*(isnr-1)+1:PAR.
NTTIs*isnr,:,:)))));
    end

  % Eb / N0
    RES.EbN0_dB(isnr) = 10*log10(mean(mean(iEb(PAR.
NTTIs*(isnr-1)+1:PAR.NTTIs*isnr,:,:))))+PAR.SNR_dB
(isnr); % SNR is in fact noise level

 % symbol error rate
    if  (PAR.Rx.ZF)
        RES.SER_ZF(isnr)= sum(sum(squeeze(symErr_ZF
(PAR.NTTIs*(isnr-1)+1:PAR.NTTIs*isnr,:,:)))) / (PAR.
NTTIs*PAR.NsymbsperTTI*PARderived.nUsedCarr);
    end
    if  (PAR.Rx.MF)
        RES.SER_MF(isnr)= sum(sum(squeeze(symErr_MF
(PAR.NTTIs*(isnr-1)+1:PAR.NTTIs*isnr,:,:)))) / (PAR.
NTTIs*PAR.NsymbsperTTI*PARderived.nUsedCarr);
    end
    if  (PAR.Rx.MMSE)
        RES.SER_MMSE(isnr)= sum(sum(squeeze
(symErr_MMSE(PAR.NTTIs*(isnr-1)+1:PAR.
NTTIs*isnr,:,:)))) / (PAR.NTTIs*PAR.
NsymbsperTTI*PARderived.nUsedCarr);
    end
    if (PAR.Rx.FFTbasedRx)
        RES.SER_FFT(isnr)= sum(sum(squeeze
(symErr_FFT(PAR.NTTIs*(isnr-1)+1:PAR.
NTTIs*isnr,:,:)))) / (PAR.NTTIs*PAR.
NsymbsperTTI*PARderived.nUsedCarr);
    end
end
```

15. Run "PostProcessing" equation. You should get the final result in a structure variable *RES* in the Workspace Variables. To visualize it, you can create Y versus X plot, and use the following code in the custom equation of the plot.



```
y = RES.MSE_FFT;
x = RES.EbN0_dB;
setindep('y', 'x');
```

There is a preconfigured graph "MSE_FFT and SER_FFT vs EbN0" in the tutorial workspace showing MST_FFT and SER_FFT versus EbN0 as shown in the screenshot below.



# Understanding Data Flow Simulation

## Understanding Data Flow Simulation

### Introduction

SystemVue is an electronic system-level (ESL) design tool that enables Baseband and RF system architects and algorithm developers to design signal processing, wireless communication, aerospace/defense and high-speed digital systems.

SystemVue uses model-based design methodology. Similar to drawing a block diagram, in SystemVue, you construct a system by placing built-in or custom models on Schematic and connecting input and output ports of the models to specify the data dependency information. To be more specific, in SystemVue, you actually place a part on the schematic which consists of a set of polymorphic models and a symbol.

To run a data flow simulation, you need to add a Data Flow Analysis in the workspace tree and specify the schematic you want to simulate in the data flow analysis. The following links guide you how to create and run a data flow simulation and how to set up the data flow analysis.

- Getting Started with Data Flow
- Setting up the Data Flow Analysis

Simulation in SystemVue is based on data flow models of computation. Before running a simulation, SystemVue data flow simulator will convert the schematic into synchronous data flow (SDF) graph representation, analyze SDF multirate consistency, identify deadlock, resolve sampling rates and compute a static schedule for efficient simulation. Refer to Introduction to Data Flow Simulation for details.

## Single Rate and Multi-Rate Systems

The following examples will guide you through various aspects in data flow simulation, including single rate system (all blocks execute at the same rate), the multi-rate system (blocks run at different rates or there is a change of sampling rate), data flow information table and the concept of synchronous data flow scheduling. The tutorial examples can be found in <SystemVue installation directory>\Examples\Tutorials\DataFlow\SynchronousDataFlow.wsv.

1. Single Rate System Tutorial
2. Single Rate Scheduling Tutorial
3. Multi-Rate System Tutorial
4. Multi-Rate Scheduling Tutorial
5. CD to DAT Sampling Rate Conversion Tutorial

## Envelope Signal

In SystemVue, an envelope signal represents EITHER a real signal x(t) OR an analytic signal $x_a(t) = ( x_i(t) + j\, x_q(t) ) \exp(j\, 2\, \pi\, f_c\, t)$ with positive characterization frequency $f_c > 0$.

SystemVue associates envelope data type with black color. A black-colored port operates on envelope signal. SystemVue restricts an envelope-typed ports to have constant characterization frequency throughout a simulation. In other words, an envelope-typed port operates on EITHER a real signal throughout a simulation, OR an analytic signal with constant characterization frequency throughout a simulation.

The following examples introduce how to use Envelope Signal to model RF signal processing at a carrier frequency. The tutorial examples can be found in <SystemVue installation directory>\Examples\Tutorials\DataFlow\EnvelopeSignal.wsv.

1. Baseband and RF
2. Envelope Bandpass Filter
3. Baseband IF and RF

### Timed System

The following examples introduce sampling rate resolution with respect to the *SampleRateOption* parameter in source models and illustrate its relation to the *System Sample Rate* parameter in data flow analysis.

1. Timed from Schematic
2. Timed from SampleRate

### Filter and Sample Rate

In SystemVue, the envelope data type filter models (filter models with black-colored ports) re-design the filter based on the incoming sampling rate during simulation setup. In contrast, for numeric filter models (e.g., FIR, FIR_Cx, FIR_Fxp, IIR, IIR_Cx, BiquadCascade), once they have been designed, the filter coefficients remain unchanged regardless of simulation sampling rates.

The following examples compare these two types of filters side by side with respect to the change of simulation sampling rate.

1. Frequency Response vs Sample Rate
2. Filtering vs Sample Rate

### Debugging

The following examples provide useful practices for debugging common data flow simulation errors, including deadlock and sample rate inconsistency.

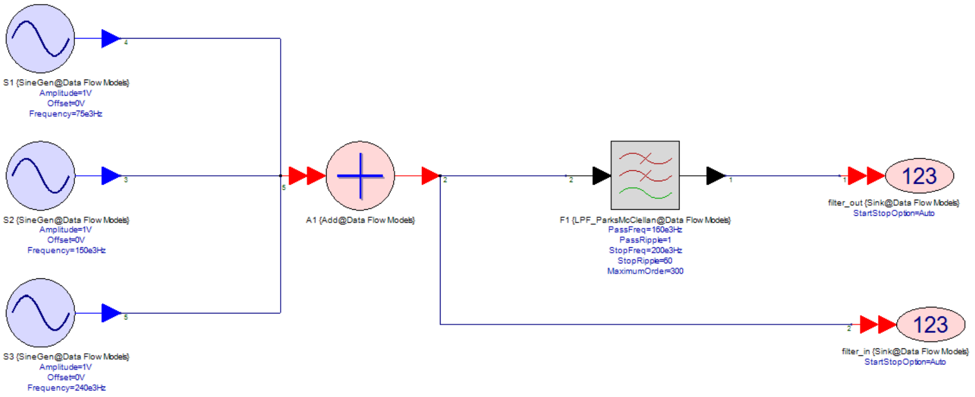
1. Deadlock
2. Sample Rate Inconsistency

## Single Rate System Tutorial

This example will guide you to create a single rate system and to understand synchronous data flow production and consumption rates, sampling rates in timed synchronous data flow, and data flow information table.

1. Open <SystemVue installation directory>\Examples\Tutorials\Algorithm_Design\DataFlow\SynchronousDataI wsv.

2. Create a design as the following screenshot in *Design1* under folder *1. Single Rate System*.



S1 {SineGen@Data Flow Models}
Amplitude=1V
Offset=0V
Frequency=75e3Hz

S2 {SineGen@Data Flow Models}
Amplitude=1V
Offset=0V
Frequency=150e3Hz

S3 {SineGen@Data Flow Models}
Amplitude=1V
Offset=0V
Frequency=240e3Hz

A1 {Add@Data Flow Models}

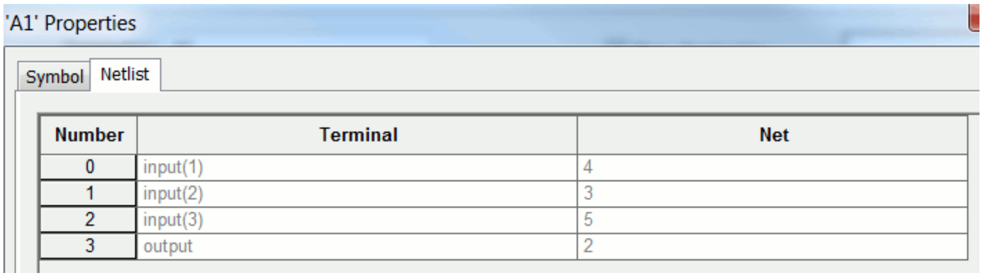
F1 {LPF_ParksMcClellan@Data Flow Models}
PassFreq=160e3Hz
PassRipple=1
StopFreq=200e3Hz
StopRipple=60
MaximumOrder=300

filter_out {Sink@Data Flow Models}
StartStopOption=Auto

filter_in {Sink@Data Flow Models}
StartStopOption=Auto

3. For the three SineGen models, set the Frequency_ parameters to 75 KHz, 150 KHz, and 240 Khz respectively, and leave the rest parameters as default.

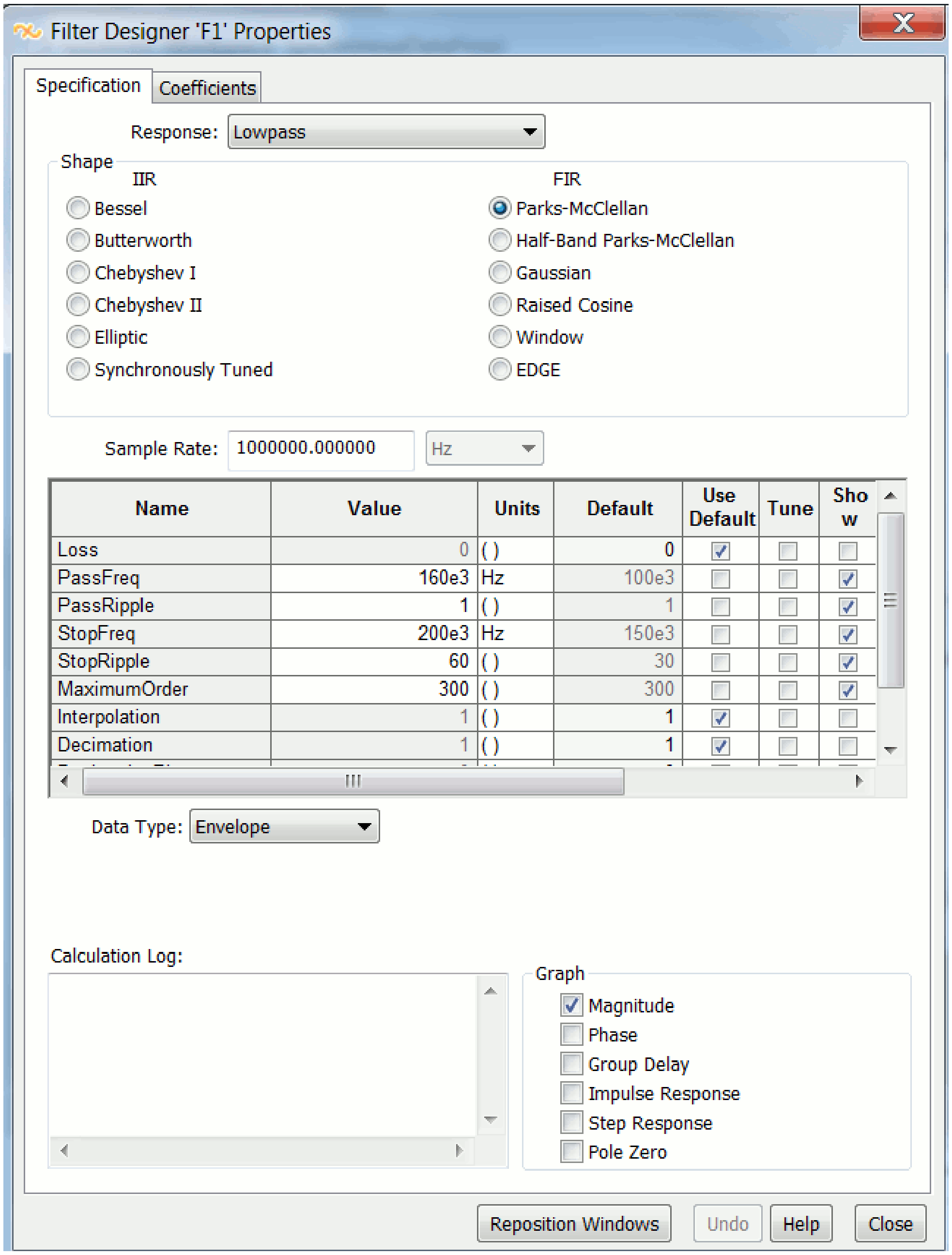
4. Make sure the Add model has three input ports. You can check it by double-click the part, select "Advanced Options...", then select "Netlist" tab.



'A1' Properties

| Number | Terminal | Net |
|---|---|---|
| 0 | input(1) | 4 |
| 1 | input(2) | 3 |
| 2 | input(3) | 5 |
| 3 | output | 2 |

5. To create a LPF ParksMcClellan model, you can first place a filter part on schematic, double-click it to open filter design dialog, select *Lowpass* response and *Parks-McClellan* FIR, set *PassFreq* to 160 Khz, *StopFreq* to 200 Khz, *StopRipple* to 60 (which means -60dB), and leave the rest parameters as default.



Filter Designer 'F1' Properties

Specification | Coefficients

Response: Lowpass

Shape

| IIR | FIR |
|---|---|
| ○ Bessel | ● Parks-McClellan |
| ○ Butterworth | ○ Half-Band Parks-McClellan |
| ○ Chebyshev I | ○ Gaussian |
| ○ Chebyshev II | ○ Raised Cosine |
| ○ Elliptic | ○ Window |
| ○ Synchronously Tuned | ○ EDGE |

Sample Rate: 1000000.000000 Hz

| Name | Value | Units | Default | Use Default | Tune | Show |
|---|---|---|---|---|---|---|
| Loss | 0 | ( ) | 0 | ☑ | □ | □ |
| PassFreq | 160e3 | Hz | 100e3 | □ | □ | ☑ |
| PassRipple | 1 | ( ) | 1 | □ | □ | ☑ |
| StopFreq | 200e3 | Hz | 150e3 | □ | □ | ☑ |
| StopRipple | 60 | ( ) | 30 | □ | □ | ☑ |
| MaximumOrder | 300 | ( ) | 300 | □ | □ | ☑ |
| Interpolation | 1 | ( ) | 1 | ☑ | □ | □ |
| Decimation | 1 | ( ) | 1 | ☑ | □ | □ |

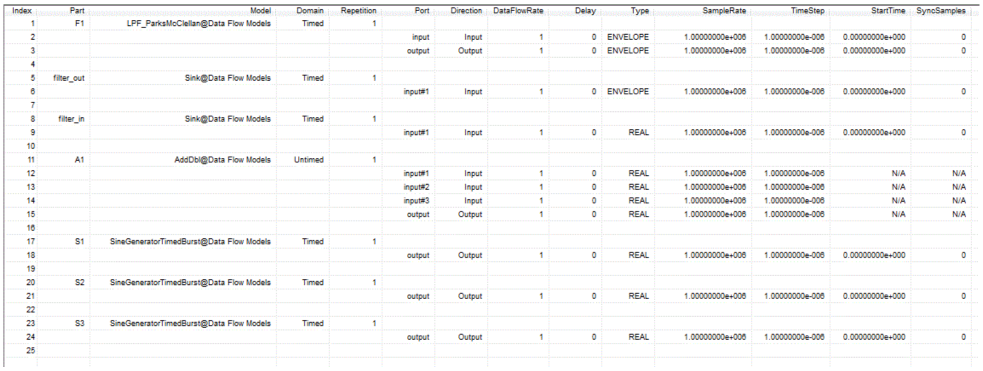
Data Type: Envelope

Calculation Log:

Graph

☑ Magnitude
□ Phase
□ Group Delay
□ Impulse Response
□ Step Response
□ Pole Zero

6. This design is to filter out the 240 Khz sine signal from the combination of three sine signals.
7. Run *DF1* Data Flow Analysis.
8. You can check the *Spectrum* graph to verify the design. The input spectrum (red color) has three tones and the 240 Khz signal has been filtered out in the output spectrum (blue color).
9. In *DF1* Data Flow Analysis, if the Display Data Flow Information check box is checked, SystemVue will display data flow information of the corresponding schematic in a table after the simulation is completed. For *DF1* Data Flow Analysis, the data flow information is displayed in *DF1_Data_DataFlowInfo* in the workspace tree.

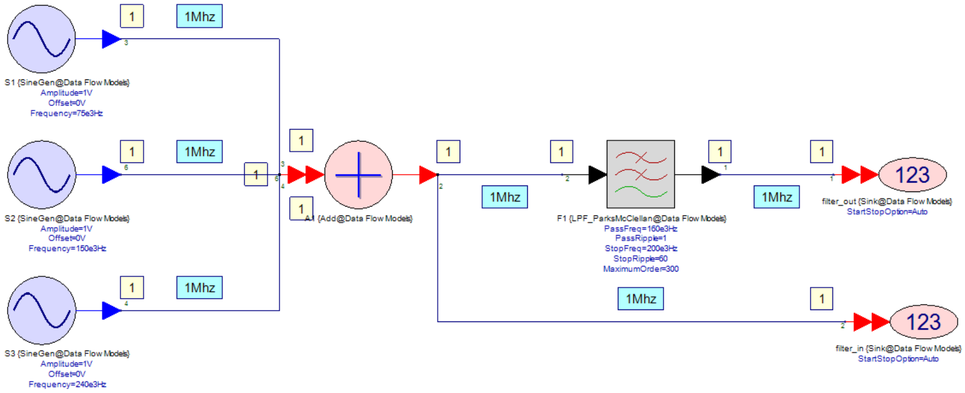


| Index | Part | Model | Domain | Repetition | Port | Direction | DataFlowRate | Delay | Type | SampleRate | TimeStep | StartTime | SyncSamples |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | F1 | LPF_ParksMcClellan@Data Flow Models | Timed | 1 | | | | | | | | | |
| 2 | | | | | input | Input | 1 | 0 | ENVELOPE | 1.00000000e+006 | 1.00000000e-006 | 0.00000000e+000 | 0 |
| 3 | | | | | output | Output | 1 | 0 | ENVELOPE | 1.00000000e+006 | 1.00000000e-006 | 0.00000000e+000 | 0 |
| 4 | | | | | | | | | | | | | |
| 5 | filter_out | Sink@Data Flow Models | Timed | 1 | | | | | | | | | |
| 6 | | | | | input#1 | Input | 1 | 0 | ENVELOPE | 1.00000000e+006 | 1.00000000e-006 | 0.00000000e+000 | 0 |
| 7 | | | | | | | | | | | | | |
| 8 | filter_in | Sink@Data Flow Models | Timed | 1 | | | | | | | | | |
| 9 | | | | | input#1 | Input | 1 | 0 | REAL | 1.00000000e+006 | 1.00000000e-006 | 0.00000000e+000 | 0 |
| 10 | | | | | | | | | | | | | |
| 11 | A1 | AddDbl@Data Flow Models | Untimed | 1 | | | | | | | | | |
| 12 | | | | | input#1 | Input | 1 | 0 | REAL | 1.00000000e+006 | 1.00000000e-006 | N/A | N/A |
| 13 | | | | | input#2 | Input | 1 | 0 | REAL | 1.00000000e+006 | 1.00000000e-006 | N/A | N/A |
| 14 | | | | | input#3 | Input | 1 | 0 | REAL | 1.00000000e+006 | 1.00000000e-006 | N/A | N/A |
| 15 | | | | | output | Output | 1 | 0 | REAL | 1.00000000e+006 | 1.00000000e-006 | N/A | N/A |
| 16 | | | | | | | | | | | | | |
| 17 | S1 | SineGeneratorTimedBurst@Data Flow Models | Timed | 1 | | | | | | | | | |
| 18 | | | | | output | Output | 1 | 0 | REAL | 1.00000000e+006 | 1.00000000e-006 | 0.00000000e+000 | 0 |
| 19 | | | | | | | | | | | | | |
| 20 | S2 | SineGeneratorTimedBurst@Data Flow Models | Timed | 1 | | | | | | | | | |
| 21 | | | | | output | Output | 1 | 0 | REAL | 1.00000000e+006 | 1.00000000e-006 | 0.00000000e+000 | 0 |
| 22 | | | | | | | | | | | | | |
| 23 | S3 | SineGeneratorTimedBurst@Data Flow Models | Timed | 1 | | | | | | | | | |
| 24 | | | | | output | Output | 1 | 0 | REAL | 1.00000000e+006 | 1.00000000e-006 | 0.00000000e+000 | 0 |
| 25 | | | | | | | | | | | | | |

10. Refer to Reading Data Flow Information Table for the meaning of each column. For *Design1*, every model consumes and/or produces one sample for each of its input and/or output port, which is shown in *DF1_Data_DataFlowInfo* as the DataFlowRate for each Input port ( consumption rate) is 1 and the DataFlowRate for each Output port ( production rate) are 1. This type of system does not change sampling rate across functional models, so it is referred to as a Single Rate system.

11. The SampleRate column in *DF1_Data_DataFlowInfo* also shows that the sampling rate remains 1 Mhz for all connections in the system.

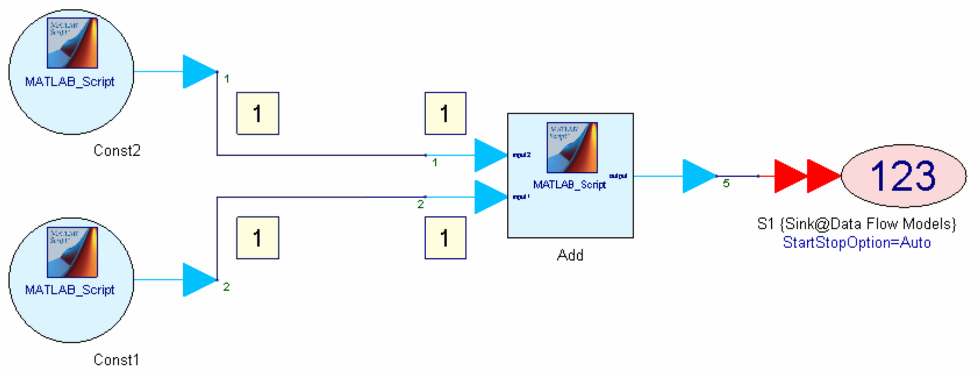
12. It is useful to annotate the production rates, consumption rates, and sampling rates on the schematic to understand the data flow behavior of the systems. Can you do that? As shown in the screenshot below, the yellow annotation next to each input / output port represents the consumption / production rate of the associated model. The blue annotation next to the connection (wire) represents the sampling rate of the signal across such connection.



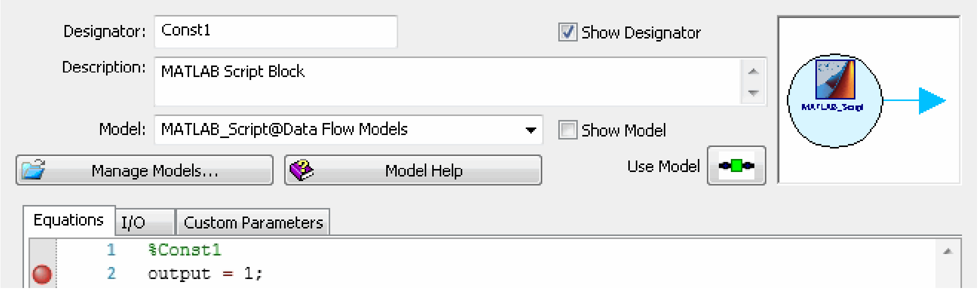
## Single Rate Scheduling Tutorial

This example will let you feel the synchronous data flow scheduling of a single rate system and understand the data driven execution in data flow.

1. Open <SystemVue installation directory>\Examples\Tutorials\Algorithm_Design\DataFlow\SynchronousData wsv.
2. Open *Design2* schematic in folder *2. Single Rate Scheduling*.



3. This is a simple single rate design, where MATLAB_Script model *Const1* outputs one constant value 1 at each firing (execution), MATLAB_Script model *Const2* outputs one constant value 2 at each firing, and MATLAB_Script model *Add* simply adds the two input samples. The production and consumption rates are annotated on schematic.
4. SystemVue computes a static schedule before running a simulation. During simulation, models are executed in the order based on the schedule. Static scheduling results in very efficient simulation because of the minimal runtime overhead. Refer to Scheduling SDF Graphs to learn general concept about scheduling.
5. A data flow model can be executed (fired) only when it has enough data samples on all of its input ports. For *Design2*, *Add* can be executed only after *Const1* and *Const2* are executed to generate two input samples.
6. A breakpoint is inserted for each MATLAB_Script model in *Design1*.

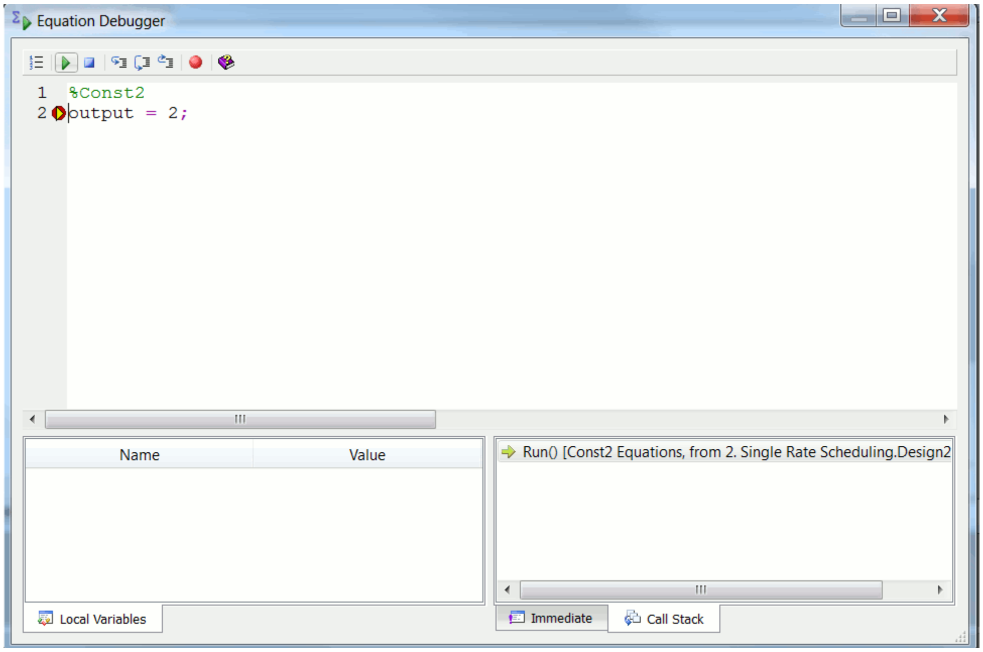

Designator: Const1 ☑ Show Designator

Description: MATLAB Script Block

Model: MATLAB_Script@Data Flow Models □ Show Model

```
%Const1
output = 1;
```

7. Now run *DF2* Data Flow Analysis.
8. When running the simulation, the execution will be stopped whenever it hits a breakpoint. You can observe the firing (execution) sequence by clicking the Go button in the equation debugger. You can stop the simulation by clicking the Stop button in the equation debugger. Refer to Debugging Equations about how to use equation debugger.



```
%Const2
output = 2;
```

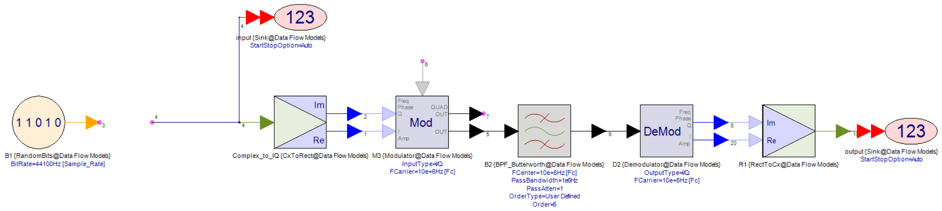
Run() [Const2 Equations, from 2. Single Rate Scheduling.Design2

9. By observing the firing sequence, you can feel the static schedule. *Const1* and *Const2* will be executed before firing *Add*, and the firing sequence is repeated throughout the simulation. Note that there is no data dependency between *Const1* and *Const2*, so firing *Const1* before *Const2* or firing *Const2* before *Const1* are both valid schedules.

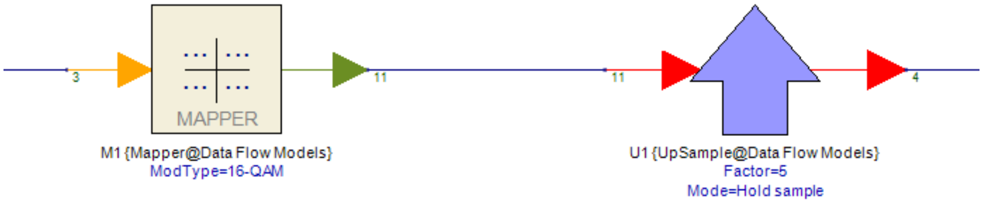
## Multi-Rate System Tutorial

This tutorial guides you to create a multi-rate system, where SDF production and consumption rates are greater than 1. By exercising this example, you will learn how multi-rate properties change sampling rates in timed synchronous data flow.

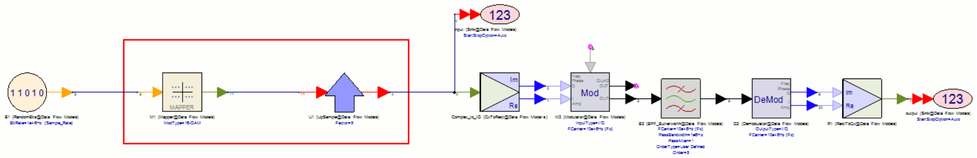
1. Open <SystemVue installation directory>\Examples\Tutorials\Algorithm_Design\DataFlow\SynchronousDataI wsv.
2. Open schematic *Design3* under folder *3. Multi-Rate System*.


123
input {Sink@Data Flow Models}
StartStopOption=Auto
1 1 0 1 0
B1 {RandomBits@Data Flow Models}
BitRate=44100Hz [Sample_Rate]
Im
Re
Complex_to_IQ {CxToRect@Data Flow Models}
Freq
Phase
Q
I
Amp
QUAD
OUT
OUT
Mod
M3 {Modulator@Data Flow Models}
InputType=IQ
FCarrier=10e+6Hz [Fc]
B2 {BPF_Butterworth@Data Flow Models}
FCenter=10e+6Hz [Fc]
PassBandwidth=1e6Hz
PassAtten=1
OrderType=User Defined
Order=5
DeMod
D2 {Demodulator@Data Flow Models}
OutputType=IQ
FCarrier=10e+6Hz [Fc]
R1 {RectToCx@Data Flow Models}
123
output {Sink@Data Flow Models}
StartStopOption=Auto

3. Between RandomBits *B1* and CxToRect *Complex_to_IQ*, add a Mapper model with *ModType* set to *16-QAM*, add a UpSample model with *Factor* = 5 and *Mode* = *Hold sample*. Connect the two models with the rest of the system.

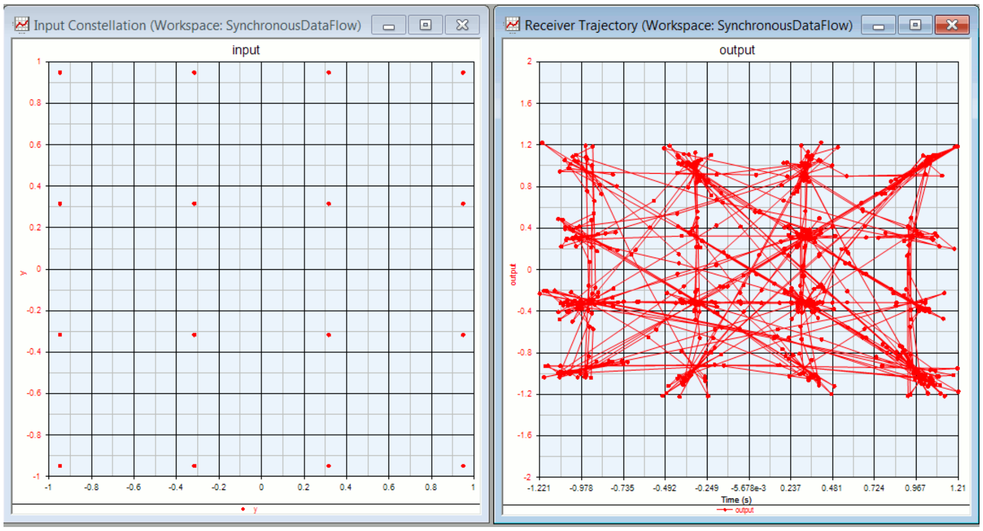


4. Schematic_Design3 now looks like:


123
input {Sink@Data Flow Models}
StartStopOption=Auto
1 1 0 1 0
B1 {RandomBits@Data Flow Models}
BitRate=1e+6Hz [Sample_Rate]
MAPPER
M1 {Mapper@Data Flow Models}
ModType=16-QAM
U1 {UpSample@Data Flow Models}
Factor=5
Im
Re
Complex_to_I/Q {CxToRect@Data Flow Models}
Mod
M3 {Modulator@Data Flow Models}
InputType=I/Q
FCarrier=10e+6Hz [Fc]
B2 {BPF_Butterworth@Data Flow Models}
FCarrier=10e+6Hz [Fc]
PassBandwidth=1e6Hz
PassAtten=1
OrderType=User Defined
Order=5
DeMod
D2 {Demodulator@Data Flow Models}
OutputType=I/Q
FCarrier=10e+6Hz [Fc]
Im
Re
R1 {RectToCx@Data Flow Models}
123
output {Sink@Data Flow Models}
StartStopOption=Auto

5. Run *DF3* Data Flow Analysis.

6. You can see *Input Constellation* graph of 16-QAM and *Receiver Trajectory* of the received signal after modulation, filtering, and demodulation.


Input Constellation (Workspace: SynchronousDataFlow)

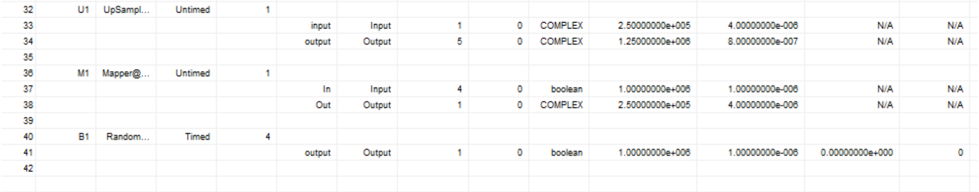
Receiver Trajectory (Workspace: SynchronousDataFlow)

7. In schematic *Design3*, the bit rate and the sampling rate of the RandomBits source are both set at 1 Mhz. In other words, there is one sample per bit outputted from the source.

8. Mapper is a multi-rate model that converts *N* number of bits into a symbol at each firing, where *N* is determined based on modulation scheme *ModType*. For example, *N* = 2 for QPSK modulation, *N* = 4 for 16-QAM modulation, and *N* = 6 for 64-QAM modulation. By setting *ModType* to 16-QAM in *Design3*, the Mapper model converts 4 bits into one symbol. Given that the sampling rate (bit rate) at the input of the Mapper is 1 Mhz, the sampling rate (symbol rate) at the output of the Mapper now becomes 0.25 Mhz because the Mapper conducts 4-to-1 multi-rate conversion.

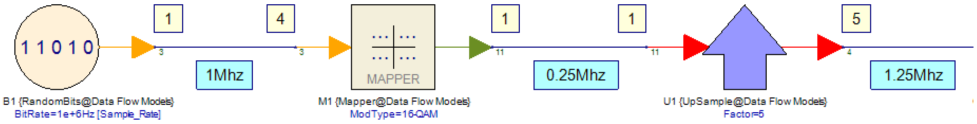
9. UpSample is also a multi-rate model. In *Design3*, the UpSample model consumes one input sample and produces 5 duplicated output samples at each firing (upsample and hold). As a result, the sampling rate after UpSample model now becomes 1.25 Mhz, which 5 times of the input sample rate 0.25 Mhz.

10. The *DF3_Data_DataFlowInfo* table shows the multi-rate data flow behavior and sampling rate change as explained in the above bullet (The Display Data Flow Information check box needs to be checked in *DF3* Data Flow Analysis in order to see the data flow information table). The consumption rate ( DataFlowRate of the input port) of the Mapper model is 4 and the production rate (DataFlowRate of the output port) of the Mapper model is 1. Similarly, the consumption rate of the UpSample model is 1 and the production rate of the UpSample model is 5. The sampling rate is changed from 1 Mhz to 0.25 Mhz to 1.25 Mhz due to the multi-rate behavior of the Mapper and UpSample .



| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 32 | U1 | UpSampl... | Untimed | 1 | | | | | | | | | |
| 33 | | | | | input | Input | 1 | 0 | COMPLEX | 2.50000000e+005 | 4.00000000e-006 | N/A | N/A |
| 34 | | | | | output | Output | 5 | 0 | COMPLEX | 1.25000000e+006 | 8.00000000e-007 | N/A | N/A |
| 35 | | | | | | | | | | | | | |
| 36 | M1 | Mapper@... | Untimed | 1 | | | | | | | | | |
| 37 | | | | | In | Input | 4 | 0 | boolean | 1.00000000e+006 | 1.00000000e-006 | N/A | N/A |
| 38 | | | | | Out | Output | 1 | 0 | COMPLEX | 2.50000000e+005 | 4.00000000e-006 | N/A | N/A |
| 39 | | | | | | | | | | | | | |
| 40 | B1 | Random... | Timed | 4 | | | | | | | | | |
| 41 | | | | | output | Output | 1 | 0 | boolean | 1.00000000e+006 | 1.00000000e-006 | 0.00000000e+000 | 0 |
| 42 | | | | | | | | | | | | | |

11. Can you annotate the production rates, consumption rates, and sampling rates for the first three models on *Design3* schematic? The annotation is shown in the screenshot below, which is very useful to understand how sampling rate is changed with the multi-rate data flow behavior.


1 1 0 1 0
1
4
1Mhz
MAPPER
1
1
0.25Mhz
5
1.25Mhz
B1 {RandomBits@Data Flow Models}
BitRate=1e+6Hz [Sample_Rate]
M1 {Mapper@Data Flow Models}
ModType=16-QAM
U1 {UpSample@Data Flow Models}
Factor=5

## Multi-Rate Scheduling Tutorial

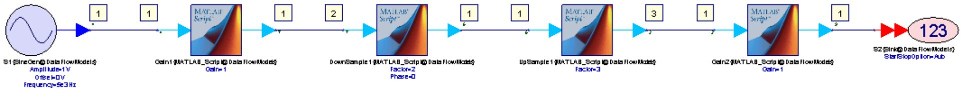
This example lets you feel the schedules of multi-rate systems and understands data driven execution and repetitions vector in SDF scheduling.

1. Open <SystemVue installation directory>\Examples\Tutorials\Algorithm_Design\DataFlow\SynchronousDataF wsv.

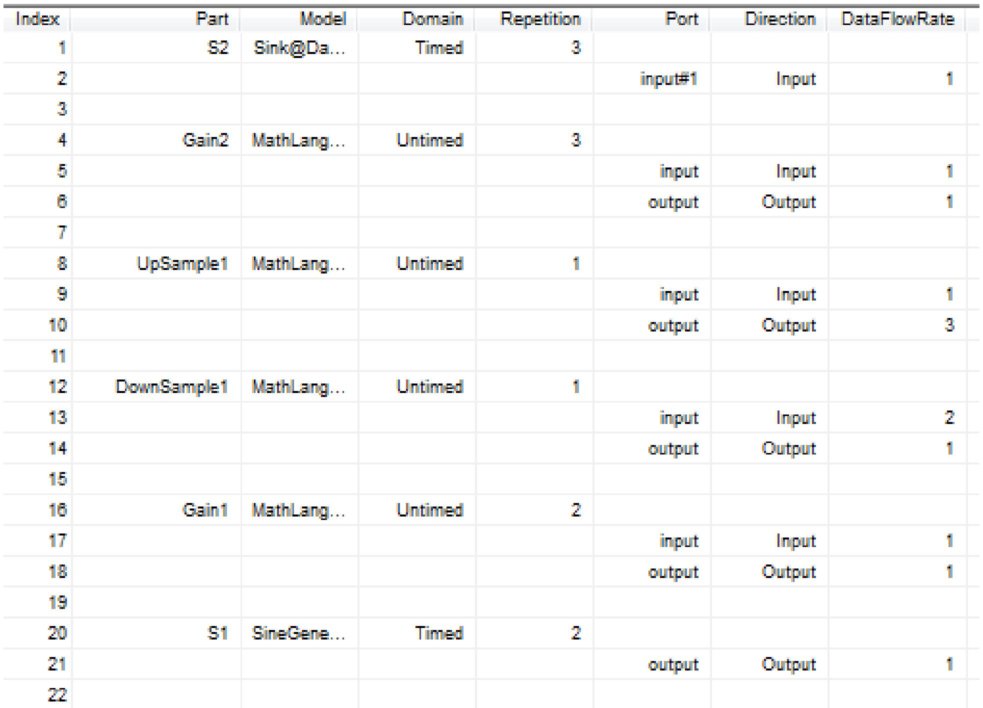
2. Open *Design4* schematic in folder *4. Multi-Rate Scheduling*.

3. In *Design4* schematic, the upper system, as shown in the screenshot below, is enabled by default. The production and consumption rates are annotated on schematic. The MATLAB_Script model *Gain1* as well as *Gain2* consumes one input sample and produces one output sample at each firing (execution). The MATLAB_Script model *DownSample1* with down sampling *Factor* = 2

consumes two input samples and produces one output sample at each firing. The MATLAB_Script model *UpSample1* with upsampling *Factor* = 3 consumes one input sample and produces three output sample at each firing.


1
1
1
2
1
1
3
1
1
123
S1 {SineGen@Data Flow Models}
Amplitude=1 V
Offset=0 V
Frequency=5e3 Hz
Gain1 {MATLAB_Script@Data Flow Models}
Gain=1
DownSample1 {MATLAB_Script@Data Flow Models}
Factor=2
Phase=0
UpSample1 {MATLAB_Script@Data Flow Models}
Factor=3
Gain2 {MATLAB_Script@Data Flow Models}
Gain=1
S2 {Sink@Data Flow Models}
StartStopOption=Aub

4. Using the same approach as Single Rate Scheduling Tutorial, all MATLAB_Script models are inserted with breakpoints.
5. Now run *DF4* Data Flow Analysis.
6. When running the simulation, the execution will be stopped whenever it hits a breakpoint. You can observe the firing (execution) sequence by clicking the Go button in the equation debugger. You can stop the simulation by clicking the Stop button in the equation debugger. Refer to Debugging Equations about how to use equation debugger.
7. By observing the firing sequence, you can figure out the static schedule: *Gain1*, *Gain1*, *DownSample1*, *UpSample1*, *Gain2*, *Gain2*, *Gain2*. *Gain1* must be fired twice before *DownSample1* because it needs two samples to fire. *Gain2* can be fired three times after one fire of *UpSample1* because it generates three samples per firing. This schedule represents a *minimal periodic schedule of the system.
8. *DF4_Data_DataFlowInfo* table shows the repetitions vector of the system. The repetition counts for *Gain1*, *DownSample1*, *UpSample1*, and *Gain2* are 2, 1, 1, 3 as shown in the Repetition column. The repetition counts of these models match the number of firings in the minimal periodic schedule.

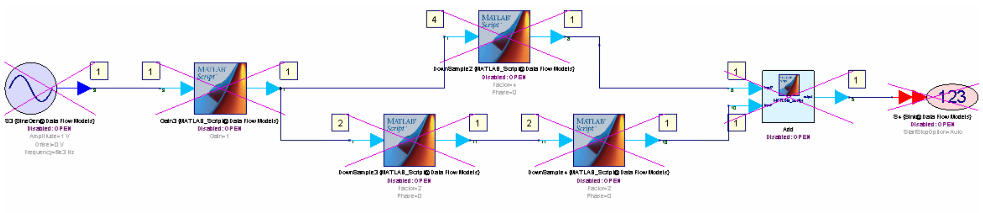


| Index | Part | Model | Domain | Repetition | Port | Direction | DataFlowRate |
|---|---|---|---|---|---|---|---|
| 1 | S2 | Sink@Da... | Timed | 3 | | | |
| 2 | | | | | input#1 | Input | 1 |
| 3 | | | | | | | |
| 4 | Gain2 | MathLang... | Untimed | 3 | | | |
| 5 | | | | | input | Input | 1 |
| 6 | | | | | output | Output | 1 |
| 7 | | | | | | | |
| 8 | UpSample1 | MathLang... | Untimed | 1 | | | |
| 9 | | | | | input | Input | 1 |
| 10 | | | | | output | Output | 3 |
| 11 | | | | | | | |
| 12 | DownSample1 | MathLang... | Untimed | 1 | | | |
| 13 | | | | | input | Input | 2 |
| 14 | | | | | output | Output | 1 |
| 15 | | | | | | | |
| 16 | Gain1 | MathLang... | Untimed | 2 | | | |
| 17 | | | | | input | Input | 1 |
| 18 | | | | | output | Output | 1 |
| 19 | | | | | | | |
| 20 | S1 | SineGene... | Timed | 2 | | | |
| 21 | | | | | output | Output | 1 |
| 22 | | | | | | | |

9. The lower system in *Design4* schematic represents a slightly complex system. Note that there is no data dependency between the path of *DownSample2* and the path of *DownSample3* and *DownSample4*, so it is up to the scheduler to schedule these two paths. In addition, the scheduler could loop the sub sequence *Gain3*, *Gain3*, *DownSample3* twice before running *DownSample4* and *DownSample2* for efficient memory consideration.

   You can disable the upper system and enable the lower system to observe the firing sequence.

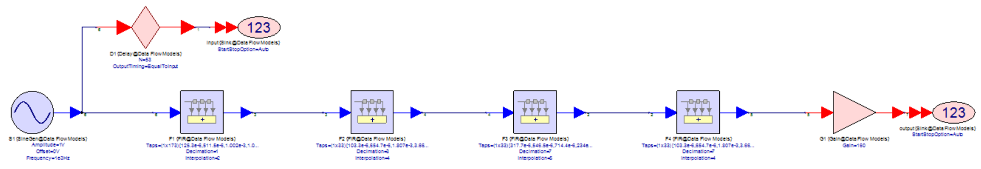
   NOTE Select a group of parts on the schematic (using the mouse to drag a selection box) and then click one of the **Disable to short** or **Disable to open** toolbar buttons.



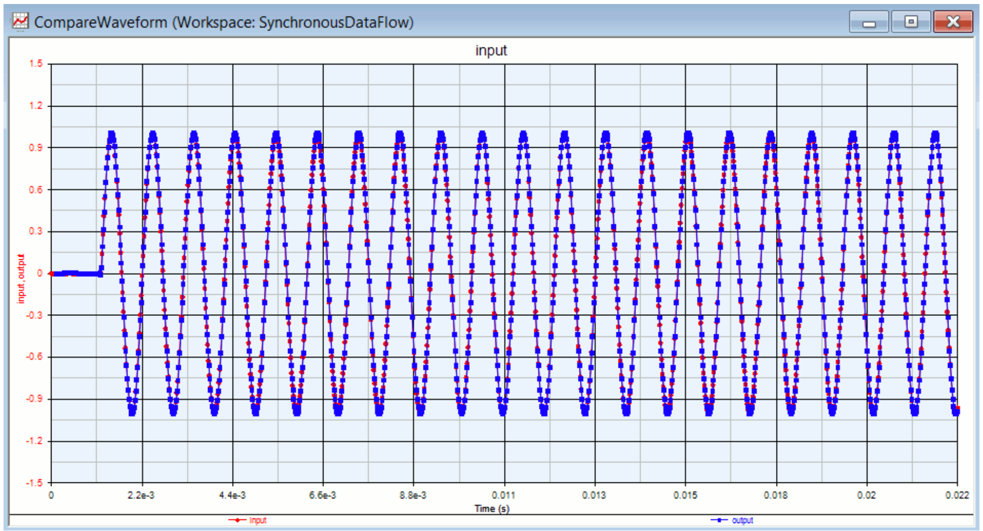
## CD to DAT Sampling Rate Conversion Tutorial

This example illustrates a practical implementation of the CD (compact disk, 44.1 Khz) to DAT (digital audio tape, 48Khz) sampling rate conversion system. Directly implementing a perfect reconstruct polyphase filter with resampling ratio 147 to 160 may require a large number of taps and resources. An alternative approach is to split 160/147 resampling ratio into four stages: 2/1, 4/3, 5/7, and 4/7.

1. Open <SystemVue installation directory>\Examples\Tutorials\Algorithm_Design\DataFlow\SynchronousDataI wsv.
2. Open *Design5* schematic in folder *5. CD to DAT*. It contains four resampling FIR filters *F1*, *F2*, *F3*, and *F4* with interpolation to decimation ratios 2/1, 4/3, 5/7, and 4/7 respectively.

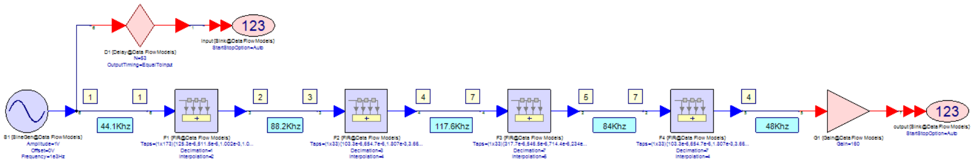

D1 {Delay@Data Flow Models}
N=53
OutputTiming=EqualToInput
123
Input {Sink@Data Flow Models}
StartStopOption=Auto
S1 {SineGen@Data Flow Models}
Amplitude=1V
Offset=0V
Frequency=1e3Hz
F1 {FIR@Data Flow Models}
Taps=(1x173)[125.3e-6,511.5e-6,1.002e-3,1.0...
Decimation=1
Interpolation=2
F2 {FIR@Data Flow Models}
Taps=(1x33)[103.3e-6,654.7e-6,1.807e-3,3.66...
Decimation=3
Interpolation=4
F3 {FIR@Data Flow Models}
Taps=(1x33)[317.7e-6,546.5e-6,714.4e-6,234e...
Decimation=7
Interpolation=5
F4 {FIR@Data Flow Models}
Taps=(1x33)[103.3e-6,654.7e-6,1.807e-3,3.66...
Decimation=7
Interpolation=4
G1 {Gain@Data Flow Models}
Gain=160
123
output {Sink@Data Flow Models}
StartStopOption=Auto

3. Run *DF5* Data Flow Analysis.
4. The _CompareWaveform_graphs compare the input and output waveforms.

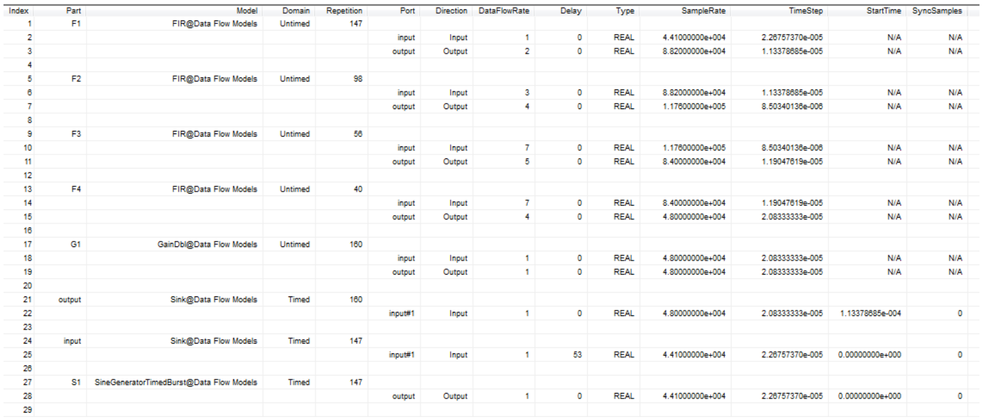


5. The sampling rate of the SineGen source is set to 44.1Khz. Can you annotate the production rates, consumption rates, and sampling rates on schematic? You can compute the results or use *DF5_Data_DataFlowInfo* table.

6. The screenshot below shows the annotated data flow information.


123
D1 {Delay@Data Flow Models}
N=53
OutputTiming=EqualToInput
Input {Sink@Data Flow Models}
StartStopOption=Auto
1
1
44.1Khz
2
3
88.2Khz
4
7
117.6Khz
5
7
84Khz
4
48Khz
123
S1 {SineGen@Data Flow Models}
Amplitude=1V
Offset=0V
Frequency=1e3Hz
F1 {FIR@Data Flow Models}
Taps=(1x173)[125.3e-6,511.5e-6,1.002e-3,1.0...
Decimation=1
Interpolation=2
F2 {FIR@Data Flow Models}
Taps=(1x33)[103.3e-6,654.7e-6,1.807e-3,3.66...
Decimation=3
Interpolation=4
F3 {FIR@Data Flow Models}
Taps=(1x33)[317.7e-6,546.5e-6,714.4e-6,234e...
Decimation=7
Interpolation=5
F4 {FIR@Data Flow Models}
Taps=(1x33)[103.3e-6,654.7e-6,1.807e-3,3.66...
Decimation=7
Interpolation=4
G1 {Gain@Data Flow Models}
Gain=160
output {Sink@Data Flow Models}
StartStopOption=Auto

7. From *DF5_Data_DataFlowInfo* table, the repetition counts for *F1*, *F2*, *F3*, *F4*, and *G1* are 147, 98, 56, 40, and 160 respectively. You can use that to verify the balance equations. Use the connection between *F1* and *F2* as an example, the repetition count of *F1* (147) times the production rate of *F1* (2) must be equal to the repetition count of *F2* (98) times the consumption rate of *F2* (3).

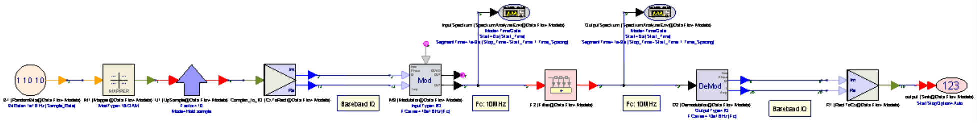


| Index | Part | Model | Domain | Repetition | Port | Direction | DataFlowRate | Delay | Type | SampleRate | TimeStep | StartTime | SyncSamples |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | F1 | FIR@Data Flow Models | Untimed | 147 | | | | | | | | | |
| 2 | | | | | input | Input | 1 | 0 | REAL | 4.41000000e+004 | 2.26757370e-005 | N/A | N/A |
| 3 | | | | | output | Output | 2 | 0 | REAL | 8.82000000e+004 | 1.13378685e-005 | N/A | N/A |
| 4 | | | | | | | | | | | | | |
| 5 | F2 | FIR@Data Flow Models | Untimed | 98 | | | | | | | | | |
| 6 | | | | | input | Input | 3 | 0 | REAL | 8.82000000e+004 | 1.13378685e-005 | N/A | N/A |
| 7 | | | | | output | Output | 4 | 0 | REAL | 1.17600000e+005 | 8.50340136e-006 | N/A | N/A |
| 8 | | | | | | | | | | | | | |
| 9 | F3 | FIR@Data Flow Models | Untimed | 56 | | | | | | | | | |
| 10 | | | | | input | Input | 7 | 0 | REAL | 1.17600000e+005 | 8.50340136e-006 | N/A | N/A |
| 11 | | | | | output | Output | 5 | 0 | REAL | 8.40000000e+004 | 1.19047619e-005 | N/A | N/A |
| 12 | | | | | | | | | | | | | |
| 13 | F4 | FIR@Data Flow Models | Untimed | 40 | | | | | | | | | |
| 14 | | | | | input | Input | 7 | 0 | REAL | 8.40000000e+004 | 1.19047619e-005 | N/A | N/A |
| 15 | | | | | output | Output | 4 | 0 | REAL | 4.80000000e+004 | 2.08333333e-005 | N/A | N/A |
| 16 | | | | | | | | | | | | | |
| 17 | G1 | GainDbl@Data Flow Models | Untimed | 160 | | | | | | | | | |
| 18 | | | | | input | Input | 1 | 0 | REAL | 4.80000000e+004 | 2.08333333e-005 | N/A | N/A |
| 19 | | | | | output | Output | 1 | 0 | REAL | 4.80000000e+004 | 2.08333333e-005 | N/A | N/A |
| 20 | | | | | | | | | | | | | |
| 21 | output | Sink@Data Flow Models | Timed | 160 | | | | | | | | | |
| 22 | | | | | input#1 | Input | 1 | 0 | REAL | 4.80000000e+004 | 2.08333333e-005 | 1.13378685e-004 | 0 |
| 23 | | | | | | | | | | | | | |
| 24 | input | Sink@Data Flow Models | Timed | 147 | | | | | | | | | |
| 25 | | | | | input#1 | Input | 1 | 53 | REAL | 4.41000000e+004 | 2.26757370e-005 | 0.00000000e+000 | 0 |
| 26 | | | | | | | | | | | | | |
| 27 | S1 | SineGeneratorTimedBurst@Data Flow Models | Timed | 147 | | | | | | | | | |
| 28 | | | | | output | Output | 1 | 0 | REAL | 4.41000000e+004 | 2.26757370e-005 | 0.00000000e+000 | 0 |
| 29 | | | | | | | | | | | | | |

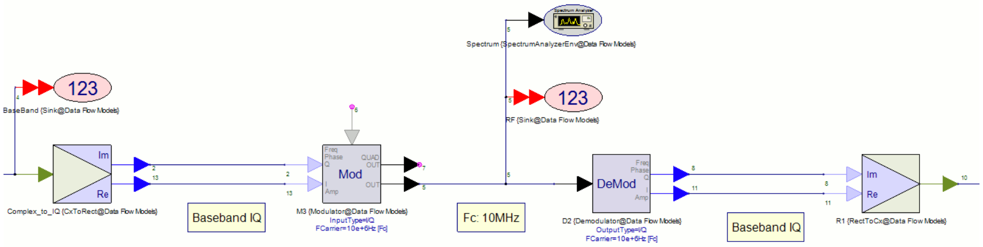
## Baseband and RF

This example will show you how to generate a modulated RF Envelope Signal from baseband I/Q signal, use SpectrumAnalyzerEnv to inspect the modulated spectrum, and use data flow information table to verify the characterization frequency (Fc) of the envelope signal.

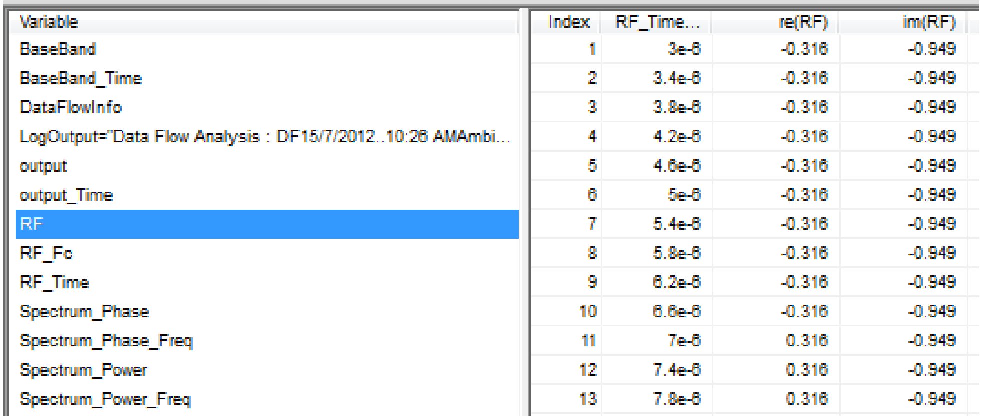
1. Open <SystemVue installation directory>\Examples\Tutorials\Algorithm_Design\DataFlow\EnvelopeSignal.wsv.
2. Open *Design1* under folder *1. BB – RF*.



3. Insert Modulator after CxToRect and insert Demodulator before RectToCx as the following screenshot. Set *FCarrier* parameter of Modulator and Demodulator to *Fc*, which is defined in the equation tab of *Design1* and set to 10Mhz. Set *InputType* parameter of the Modulator and *OutputType* parameter of the Demodulator to *I/Q*. Leave other parameters of Modulator and Demodulator as default. The Modulator takes baseband complex I/Q signal and modulates it to carrier frequency *FCarrier*. The modulated RF signal is collected in Sink *RF* and its spectrum is analyzed using SpectrumAnalyzerEnv *Spectrum*.

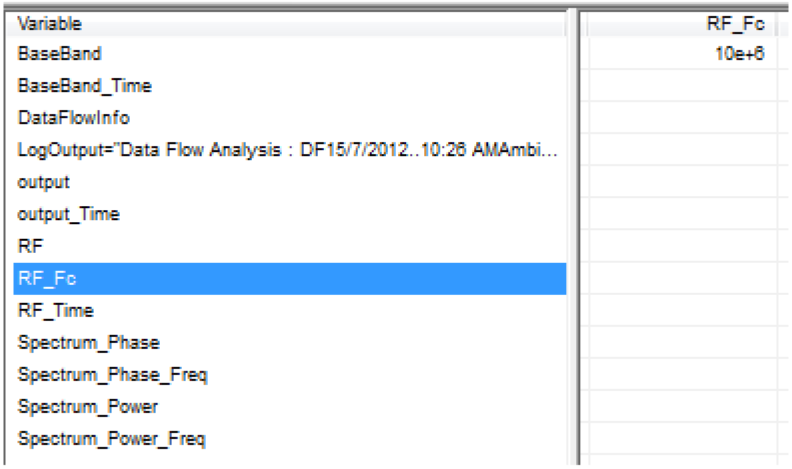

Spectrum {SpectrumAnalyzerEnv@Data Flow Models}
123
BaseBand {Sink@Data Flow Models}
123
RF {Sink@Data Flow Models}
Im
Re
Complex_to_IQ {CxToRect@Data Flow Models}
Baseband IQ
Freq
Phase
Q
QUAD OUT
Mod
I
Amp
OUT
M3 {Modulator@Data Flow Models}
InputType=I/Q
FCarrier=10e+6Hz [Fc]
Fc: 10MHz
DeMod
Freq
Phase
Q
I
Amp
D2 {Demodulator@Data Flow Models}
OutputType=I/Q
FCarrier=10e+6Hz [Fc]
Baseband IQ
Im
Re
R1 {RectToCx@Data Flow Models}

4. Run *DF1* data flow analysis.
5. Inspect *DF1_Data* dataset. For envelope signal with characterization frequency greater than 0, SystemVue records the complex envelope of the envelope signal (i.e., in-phase and quadrature components) as complex numbers in dataset and also records the characterization frequency in dataset. In this example, the complex envelope and the characterization frequency of the modulated RF signal is recorded as dataset variable *RF* and *RF_Fc*.



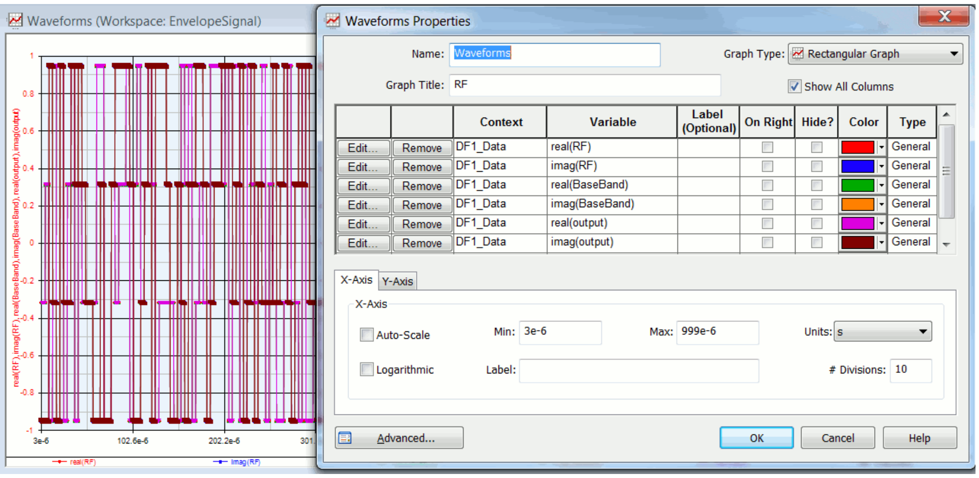
| Variable | Index | RF_Time... | re(RF) | im(RF) |
|---|---|---|---|---|
| BaseBand | 1 | 3e-6 | -0.316 | -0.949 |
| BaseBand_Time | 2 | 3.4e-6 | -0.316 | -0.949 |
| DataFlowInfo | 3 | 3.8e-6 | -0.316 | -0.949 |
| LogOutput="Data Flow Analysis : DF15/7/2012..10:26 AMAmbi... | 4 | 4.2e-6 | -0.316 | -0.949 |
| output | 5 | 4.6e-6 | -0.316 | -0.949 |
| output_Time | 6 | 5e-6 | -0.316 | -0.949 |
| RF | 7 | 5.4e-6 | -0.316 | -0.949 |
| RF_Fc | 8 | 5.8e-6 | -0.316 | -0.949 |
| RF_Time | 9 | 6.2e-6 | -0.316 | -0.949 |
| Spectrum_Phase | 10 | 6.6e-6 | -0.316 | -0.949 |
| Spectrum_Phase_Freq | 11 | 7e-6 | 0.316 | -0.949 |
| Spectrum_Power | 12 | 7.4e-6 | 0.316 | -0.949 |
| Spectrum_Power_Freq | 13 | 7.8e-6 | 0.316 | -0.949 |

6. You can verify that the characterization frequency of the modulated RF signal (*RF_Fc*) is indeed 10Mhz.



| Variable | RF_Fc |
|---|---|
| BaseBand | 10e+6 |
| BaseBand_Time | |
| DataFlowInfo | |
| LogOutput="Data Flow Analysis : DF15/7/2012..10:26 AMAmbi... | |
| output | |
| output_Time | |
| RF | |
| RF_Fc | |
| RF_Time | |
| Spectrum_Phase | |
| Spectrum_Phase_Freq | |
| Spectrum_Power | |
| Spectrum_Power_Freq | |

7. The *Waveforms* graph compares the following three waveforms: 1) the input baseband complex I/Q signal (variable *Baseband*) before Modulator, 2) the complex envelope of the RF signal (variable *RF*) after Modulator, and 3) the output baseband complex I/Q signal (variable *output*) after Demodulator. They should be exactly the same because both Modulator and Demodulator use *I/Q* type. Double click the graph to open the graph properties, you can hide/unhide the signal for comparison.


Waveforms (Workspace: EnvelopeSignal)
real(RF), imag(RF), real(BaseBand), imag(BaseBand), real(output), imag(output)
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
3e-6
102.6e-6
202.2e-6
301.
real(RF)
imag(RF)
X-Axis
Y-Axis
Advanced...
OK
Cancel
Help

Waveforms Properties

Name: Waveforms

Graph Type: Rectangular Graph

Graph Title: RF

☑ Show All Columns

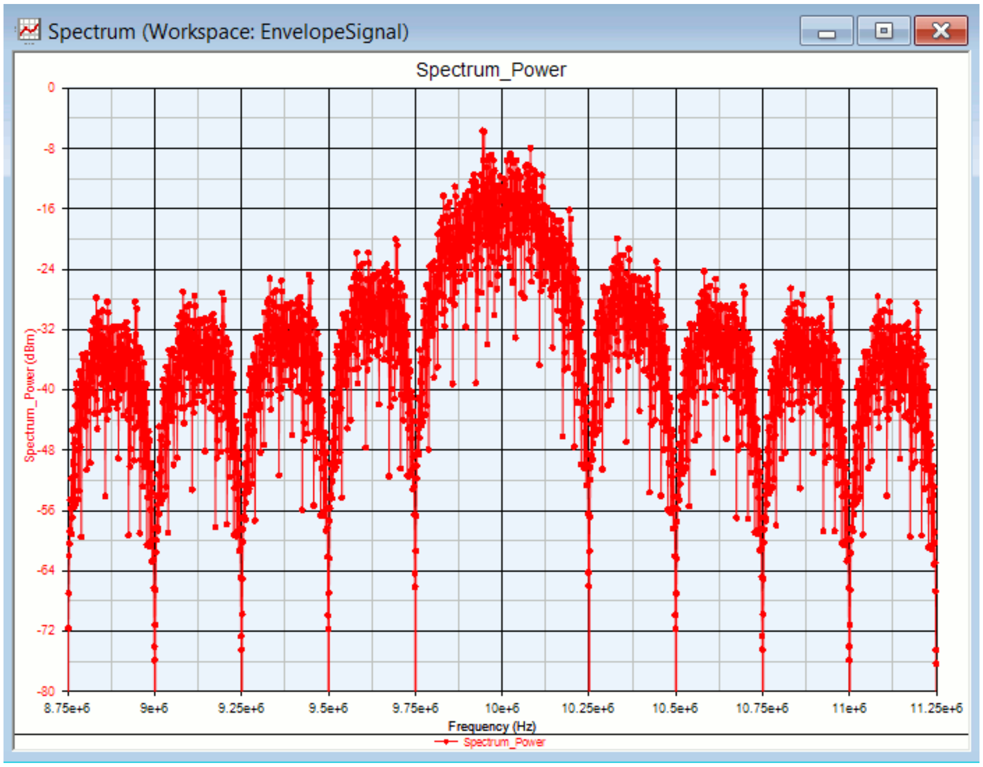
| | | Context | Variable | Label (Optional) | On Right | Hide? | Color | Type |
|---|---|---|---|---|---|---|---|---|
| Edit... | Remove | DF1_Data | real(RF) | | □ | □ | 🟥 | General |
| Edit... | Remove | DF1_Data | imag(RF) | | □ | □ | 🟦 | General |
| Edit... | Remove | DF1_Data | real(BaseBand) | | □ | □ | 🟩 | General |
| Edit... | Remove | DF1_Data | imag(BaseBand) | | □ | □ | 🟧 | General |
| Edit... | Remove | DF1_Data | real(output) | | □ | □ | 🟪 | General |
| Edit... | Remove | DF1_Data | imag(output) | | □ | □ | 🟫 | General |

X-Axis

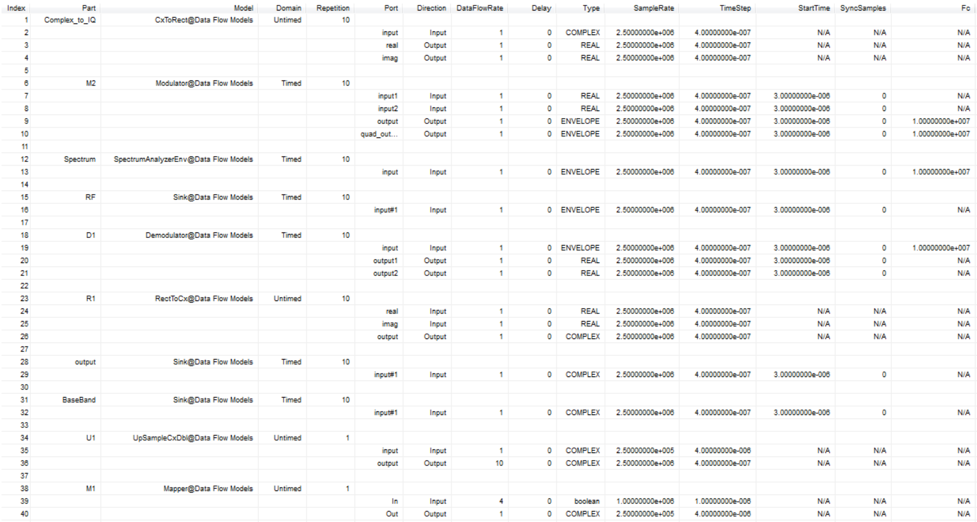
□ Auto-Scale Min: 3e-6 Max: 999e-6 Units: s

□ Logarithmic Label: # Divisions: 10

8. In the *Spectrum* graph, you can inspect the spectrum of the modulated signal. The spectrum of an RF envelope signal is centered at the characterization frequency (*Fc*) with spectrum spanned over the simulation bandwidth of the signal, i.e., from *Fc - Fs/2* to *Fc + Fs/2*, where *Fs* is the sampling rate of the envelope signal at the input of the SpectrumAnalyzerEnv . In this example, the *Spectrum* graph is centered at 10Mhz and spanned from 8.75Mhz (10Mhz - 2.5Mhz/2) to 11.25Mhz (10Mhz + 2.5Mhz/2), where 2.5Mhz is the sampling rate of the modulated RF envelope signal.



9. The *DF1_Data_DataFlowInfo* table shows the sampling rates (column SampleRate) and characterization frequencies (column Fc) of the envelope signals at each connection. For example, the sampling rate at the output of the Modulator, the input of the SpectrumAnalyzerEnv, and the input of the Demodulator is at 2.5Mhz, and the characterization frequency is at 10Mhz.



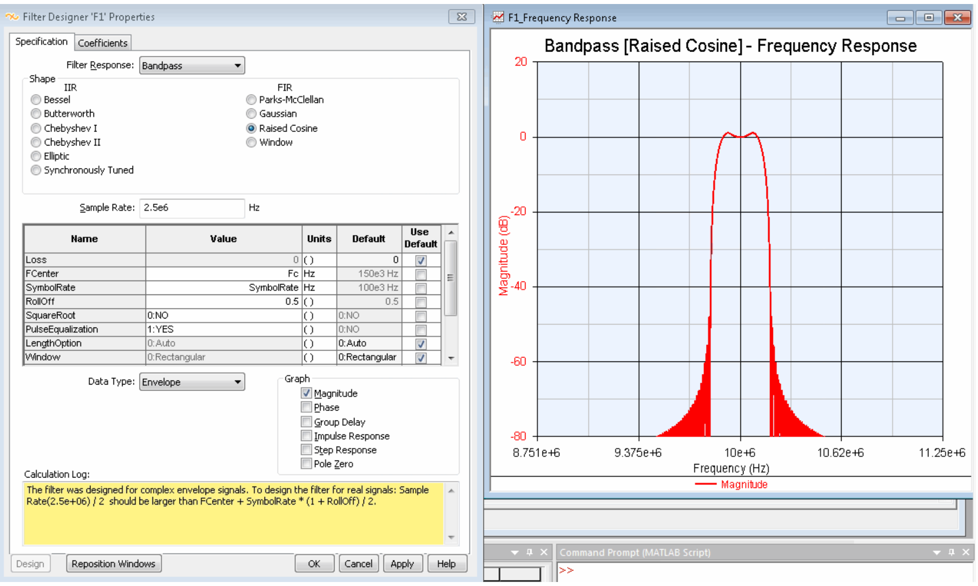
| Index | Part | Model | Domain | Repetition | Port | Direction | DataFlowRate | Delay | Type | SampleRate | TimeStep | StartTime | SyncSamples | Fc |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Complex_to_IQ | CxToRect@Data Flow Models | Untimed | 10 | | | | | | | | | | |
| 2 | | | | | input | Input | 1 | 0 | COMPLEX | 2.50000000e+006 | 4.00000000e-007 | N/A | N/A | N/A |
| 3 | | | | | real | Output | 1 | 0 | REAL | 2.50000000e+006 | 4.00000000e-007 | N/A | N/A | N/A |
| 4 | | | | | imag | Output | 1 | 0 | REAL | 2.50000000e+006 | 4.00000000e-007 | N/A | N/A | N/A |
| 5 | | | | | | | | | | | | | | |
| 6 | M2 | Modulator@Data Flow Models | Timed | 10 | | | | | | | | | | |
| 7 | | | | | input1 | Input | 1 | 0 | REAL | 2.50000000e+006 | 4.00000000e-007 | 3.00000000e-006 | 0 | N/A |
| 8 | | | | | input2 | Input | 1 | 0 | REAL | 2.50000000e+006 | 4.00000000e-007 | 3.00000000e-006 | 0 | N/A |
| 9 | | | | | output | Output | 1 | 0 | ENVELOPE | 2.50000000e+006 | 4.00000000e-007 | 3.00000000e-006 | 0 | 1.00000000e+007 |
| 10 | | | | | quad_out... | Output | 1 | 0 | ENVELOPE | 2.50000000e+006 | 4.00000000e-007 | 3.00000000e-006 | 0 | 1.00000000e+007 |
| 11 | | | | | | | | | | | | | | |
| 12 | Spectrum | SpectrumAnalyzerEnv@Data Flow Models | Timed | 10 | | | | | | | | | | |
| 13 | | | | | input | Input | 1 | 0 | ENVELOPE | 2.50000000e+006 | 4.00000000e-007 | 3.00000000e-006 | 0 | 1.00000000e+007 |
| 14 | | | | | | | | | | | | | | |
| 15 | RF | Sink@Data Flow Models | Timed | 10 | | | | | | | | | | |
| 16 | | | | | input#1 | Input | 1 | 0 | ENVELOPE | 2.50000000e+006 | 4.00000000e-007 | 3.00000000e-006 | 0 | N/A |
| 17 | | | | | | | | | | | | | | |
| 18 | D1 | Demodulator@Data Flow Models | Timed | 10 | | | | | | | | | | |
| 19 | | | | | input | Input | 1 | 0 | ENVELOPE | 2.50000000e+006 | 4.00000000e-007 | 3.00000000e-006 | 0 | 1.00000000e+007 |
| 20 | | | | | output1 | Output | 1 | 0 | REAL | 2.50000000e+006 | 4.00000000e-007 | 3.00000000e-006 | 0 | N/A |
| 21 | | | | | output2 | Output | 1 | 0 | REAL | 2.50000000e+006 | 4.00000000e-007 | 3.00000000e-006 | 0 | N/A |
| 22 | | | | | | | | | | | | | | |
| 23 | R1 | RectToCx@Data Flow Models | Untimed | 10 | | | | | | | | | | |
| 24 | | | | | real | Input | 1 | 0 | REAL | 2.50000000e+006 | 4.00000000e-007 | N/A | N/A | N/A |
| 25 | | | | | imag | Input | 1 | 0 | REAL | 2.50000000e+006 | 4.00000000e-007 | N/A | N/A | N/A |
| 26 | | | | | output | Output | 1 | 0 | COMPLEX | 2.50000000e+006 | 4.00000000e-007 | N/A | N/A | N/A |
| 27 | | | | | | | | | | | | | | |
| 28 | output | Sink@Data Flow Models | Timed | 10 | | | | | | | | | | |
| 29 | | | | | input#1 | Input | 1 | 0 | COMPLEX | 2.50000000e+006 | 4.00000000e-007 | 3.00000000e-006 | 0 | N/A |
| 30 | | | | | | | | | | | | | | |
| 31 | BaseBand | Sink@Data Flow Models | Timed | 10 | | | | | | | | | | |
| 32 | | | | | input#1 | Input | 1 | 0 | COMPLEX | 2.50000000e+006 | 4.00000000e-007 | 3.00000000e-006 | 0 | N/A |
| 33 | | | | | | | | | | | | | | |
| 34 | U1 | UpSampleCxDbl@Data Flow Models | Untimed | 1 | | | | | | | | | | |
| 35 | | | | | input | Input | 1 | 0 | COMPLEX | 2.50000000e+005 | 4.00000000e-006 | N/A | N/A | N/A |
| 36 | | | | | output | Output | 10 | 0 | COMPLEX | 2.50000000e+006 | 4.00000000e-007 | N/A | N/A | N/A |
| 37 | | | | | | | | | | | | | | |
| 38 | M1 | Mapper@Data Flow Models | Untimed | 1 | | | | | | | | | | |
| 39 | | | | | In | Input | 4 | 0 | boolean | 1.00000000e+006 | 1.00000000e-006 | N/A | N/A | N/A |
| 40 | | | | | Out | Output | 1 | 0 | COMPLEX | 2.50000000e+005 | 4.00000000e-006 | N/A | N/A | N/A |

10. The envelope signal simulation technology in SystemVue allows users to use simulation sampling rate in the range of signal bandwidth, which is much smaller than the carrier frequency.

## Envelope Bandpass Filter

This example illustrates how to create a bandpass filter for filtering RF envelope signal.

1. Open <SystemVue installation directory>\Examples\Tutorials\Algorithm_Design\DataFlow\EnvelopeSignal.wsv.
2. Open *Design2* in folder *2. Envelope Bandpass Filter*. This example is continued from the previous example Baseband and RF.
3. Design a bandpass raised cosine filter to filter out the spectrum images (see the spectrum in Baseband and RF) due to the UpSample model that up-samples (up-sample and hold) the incoming symbols by 10 times. To design the filter, place down a filter part, double click the part to bring up the Filter Designer. Choose *Bandpass* Response and select *Raised Cosine* FIR filter. Set *Sample Rate* to 2.5 Mhz (2.5e6 Hz). In the parameter grid, set *FCenter* (center frequency) to "Fc" (*Fc* is defined in the *Design2* equation tab as 10 Mhz, which is the carrier frequency of the Modulator). Set *SymbolRate* (passband bandwidth) to "SymbolRate" (*SymbolRate* is defined in the *Design2* equation tab as 0.25 Mhz, which is the symbol rate after the Mapper ). Set *PulseEqualization* to YES to equalize the symbol pulses due to up-sample and hold. Leave the other parameters as default. Select *Envelope* Data Type. The filter designer will automatically design the filter and show you the bandpass frequency response as in the screenshot below.



Filter Designer 'F1' Properties

Specification | Coefficients

Filter Response: Bandpass

Shape

| IIR | FIR |
|---|---|
| ○ Bessel | ○ Parks-McClellan |
| ○ Butterworth | ○ Gaussian |
| ○ Chebyshev I | ● Raised Cosine |
| ○ Chebyshev II | ○ Window |
| ○ Elliptic | |
| ○ Synchronously Tuned | |

Sample Rate: 2.5e6 Hz

| Name | Value | Units | Default | Use Default |
|---|---|---|---|---|
| Loss | 0 | () | 0 | ☑ |
| FCenter | Fc | Hz | 150e3 Hz | □ |
| SymbolRate | SymbolRate | Hz | 100e3 Hz | □ |
| RollOff | 0.5 | () | 0.5 | □ |
| SquareRoot | 0:NO | () | 0:NO | □ |
| PulseEqualization | 1:YES | () | 0:NO | □ |
| LengthOption | 0:Auto | () | 0:Auto | ☑ |
| Window | 0:Rectangular | () | 0:Rectangular | ☑ |

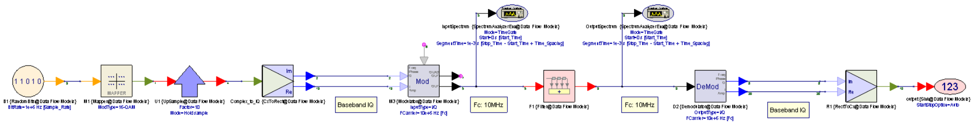
Data Type: Envelope

Graph
☑ Magnitude
□ Phase
□ Group Delay
□ Impulse Response
□ Step Response
□ Pole Zero

Calculation Log:

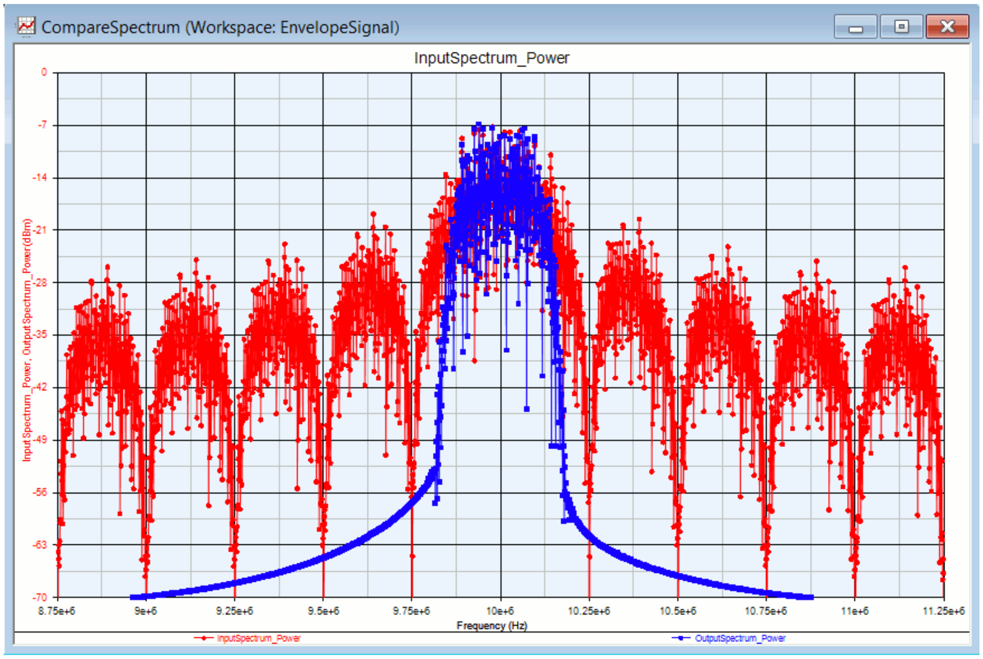
The filter was designed for complex envelope signals. To design the filter for real signals: Sample Rate(2.5e+06) / 2 should be larger than FCenter + SymbolRate * (1 + RollOff) / 2.

4. Close the filter designer. The filter part will be configured automatically to use the BPF RaisedCosine model based on the specification. Place the filter in between Modulator *M3* and Demodulator *D2* as shown in the following screenshots.



5. Run *DF2* data flow analysis.

6. The *CompareSpectrum* graph compares the *InputSpectrum* (red color) and *OutputSpectrum* (blue color) before and after of the bandpass raised cosine filter. As you can see, by setting the center frequency (*FCenter*) of the bandpass filter to the carrier frequency (*Fc = 10 Mhz*) of the RF envelope signal and by setting the passband bandwidth (*SymbolRate*) of the bandpass raised cosine filter to the symbol rate after the Mapper (*SymbolRate* = 0.25 Mhz), we can filter out the images outside the symbol bandwidth (9.875 Mhz to 10.125 Mhz).



7. When filtering RF envelope signal (Fc > 0) with bandpass filters, SystemVue actual designs a low-pass filter with lowpass passband frequency = bandpass passband bandwidth / 2 to filter the complex envelope representation (in-phase and quadrature components) of the RF signal. This approach allows users to use simulation sampling rate (2.5 Mhz in this example) that is much smaller than the carrier frequency (10 Mhz in this example). Filtering an RF signal with its real-value representation requires simulation sampling rate at least twice of the carrier frequency, which is in general order of magnitude larger than the signal bandwidth (see the next tutorial example Baseband IF and RF where RF carrier frequency is at 1 Ghz).
8. Go back to the Filter Designer by double clicking the BPF RaisedCosine part and then clicking the "Filter Designer" button in the part properties dialog. Let's review the message in the Calculation Log:

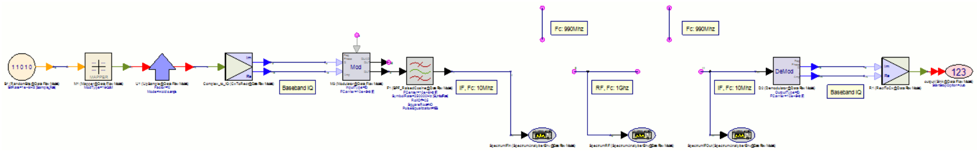
   CAUTION The filter was designed for complex envelope signals. To design the filter for real signals: Sample Rate(2.5e+06) / 2 should be larger than FCenter + SymbolRate * (1 + RollOff) / 2.

   What this message means is that the 2.5 Mhz sample rate is not enough to capture the bandpass filter response for filtering the RF signal in its real-value representation. Instead, the filter is designed with 2.5 Mhz sample rate to filter the complex envelope of the RF signal.
9. For more details, refer to Bandpass and Bandstop Filtering for Analytic Signals.

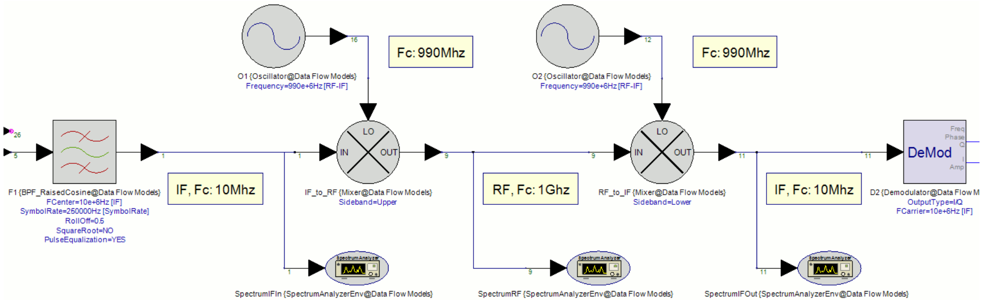
## Baseband, IF and RF

This example illustrates how to use Mixer and Oscillator to convert between intermediate frequency (IF) and radio frequency (RF) and also shows how to use data flow information table to track the change of characterization frequencies (Fc).

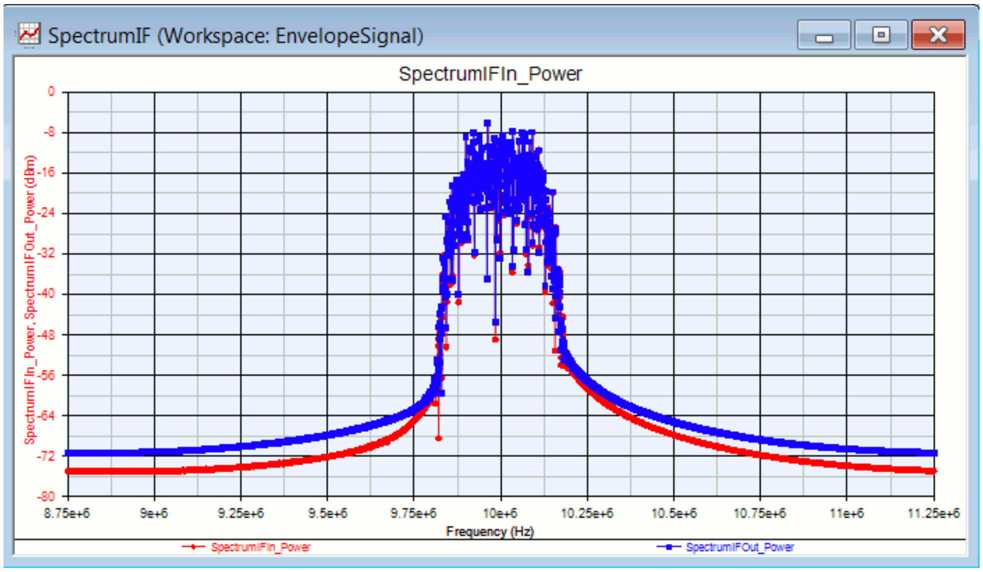
1. Open <SystemVue installation directory>\Examples\Tutorials\Algorithm_Design\DataFlow\EnvelopeSignal.wsv.
2. Open *Design3* in folder *3. BB – IF – RF*. This example is continued from the previous example Envelope Bandpass Filter.



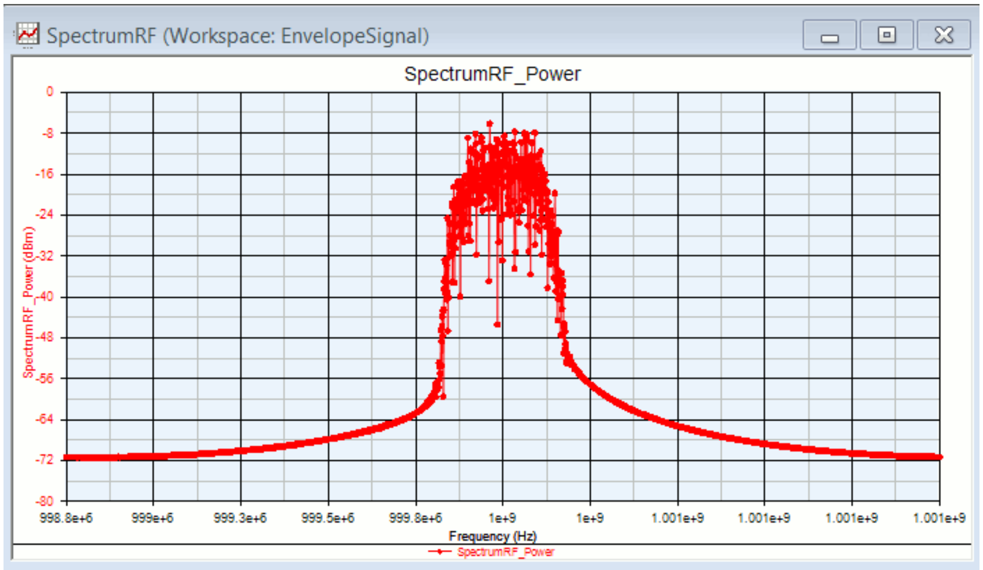
3. Insert a Mixer after BPF RaisedCosine *F1*, name the mixer *IF_to_RF*, and set the *Sideband* parameter to *Upper*. Add an Oscillator and connect its output to the *LO* input of the mixer *IF_to_RF* and set its *Frequency* parameter to "RF-IF" (990 Mhz), where IF = 10 Mhz and RF = 1 GHz are defined in the equation tab of *Design3*. Insert a Mixer between *IF_to_RF* and Demodulator *D2*, name the mixer *RF_to_IF*, and set the *Sideband* parameter to *Lower*. Add an Oscillator and connect its output to the *LO* input of the mixer *RF_to_IF* and set its *Frequency* parameter to "RF-IF". See the following screenshot as reference.


O1 {Oscillator@Data Flow Models}
Frequency=990e+6Hz [RF-IF]
Fc: 990Mhz
O2 {Oscillator@Data Flow Models}
Frequency=990e+6Hz [RF-IF]
Fc: 990Mhz
LO
IN
OUT
LO
IN
OUT
DeMod
Freq
Phase
Q
I
Amp
F1 {BPF_RaisedCosine@Data Flow Models}
FCenter=10e+6Hz [IF]
SymbolRate=250000Hz [SymbolRate]
RollOff=0.5
SquareRoot=NO
PulseEqualization=YES
IF, Fc: 10Mhz
IF_to_RF {Mixer@Data Flow Models}
Sideband=Upper
RF, Fc: 1Ghz
RF_to_IF {Mixer@Data Flow Models}
Sideband=Lower
IF, Fc: 10Mhz
D2 {Demodulator@Data Flow M
OutputType=I/Q
FCarrier=10e+6Hz [IF]
SpectrumIFIn {SpectrumAnalyzerEnv@Data Flow Models}
SpectrumRF {SpectrumAnalyzerEnv@Data Flow Models}
SpectrumIFOut {SpectrumAnalyzerEnv@Data Flow Models}

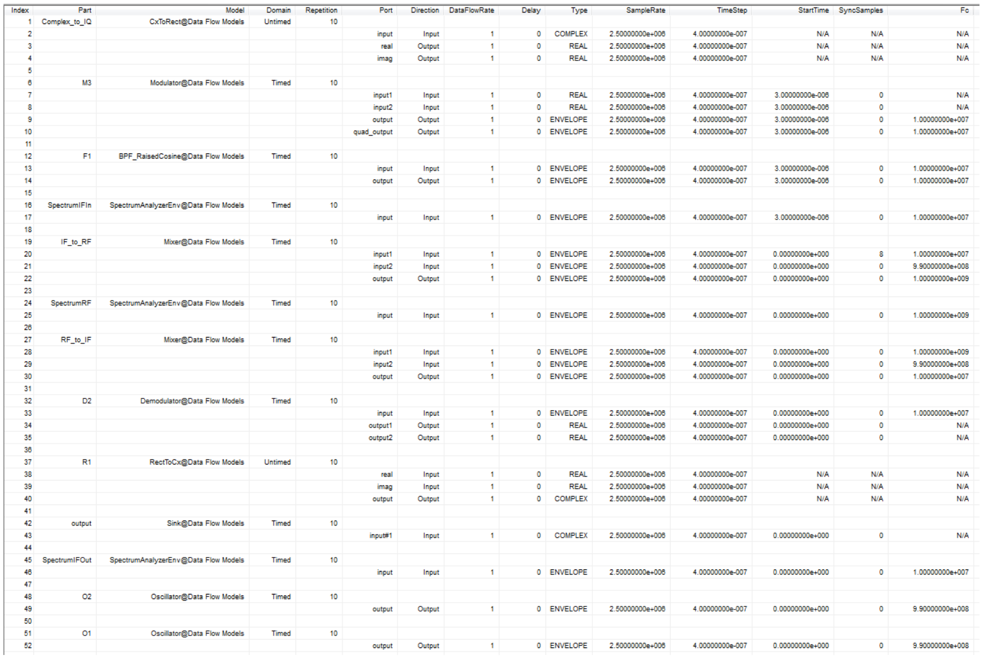
4. The carrier frequency of the Modulator is set to *IF* (10 Mhz) and the center frequency of the BPF RaisedCosine is also set to *IF* (10 Mhz). The mixer *IF_to_RF* will convert the envelope signal from characterization frequency 10 Mhz to characterization frequency 1 Ghz (upper sideband = 990 + 10 = 1000 Mhz), which is the RF carrier frequency. The mixer *RF_to_IF* will convert the envelope signal from characterization frequency 1 Ghz to characterization frequency 10 Mhz (lower sideband = 1000 - 990 = 10 Mhz), which is the IF frequency.
5. Run *DF3* data flow analysis.
6. The *SpectrumIF* graph shows the IF spectrum at the input of the mixer *IF_to_RF* (red color) and the IF spectrum at the output of the mixer *RF_to_IF* (blue color). Both of them are centered at 10 Mhz, i.e., the characterization frequency of the IF envelope signal. The spectrum is spanned from 8.75Mhz (10Mhz - 2.5Mhz/2) to 11.25Mhz (10Mhz + 2.5Mhz/2), where 2.5Mhz is the sampling rate of the modulated IF envelope signal.



7. The *SpectrumRF* graph shows the RF spectrum at the output of the mixer *IF_to_RF* (red color). The spectrum is now centered at 1 Ghz, i.e., the characterization frequency of the RF envelope signal. Note that the spectrum is still spanned across 2.5Mhz bandwidth from (1Ghz - 2.5Mhz/2) to (1Ghz + 2.5Mhz/2) because simulation sampling rate remains the same at 2.5 Mhz.



8. From *DF3_Data_FataFlowInfo* table, you can verify the characterization frequency (column Fc) of the envelope signals at different places. For example, the output of Modulator *M3* is at Fc = 10 Mhz (IF). The LO input of the mixer *IF_to_RF* is at 990 Mhz, which is also the Fc at the output of the Oscillator O1. The output of the mixer *IF_to_RF* is at Fc = 1 Ghz (RF), and the output of the mixer *RF_to_IF* is back again at Fc = 10 Mhz (IF).

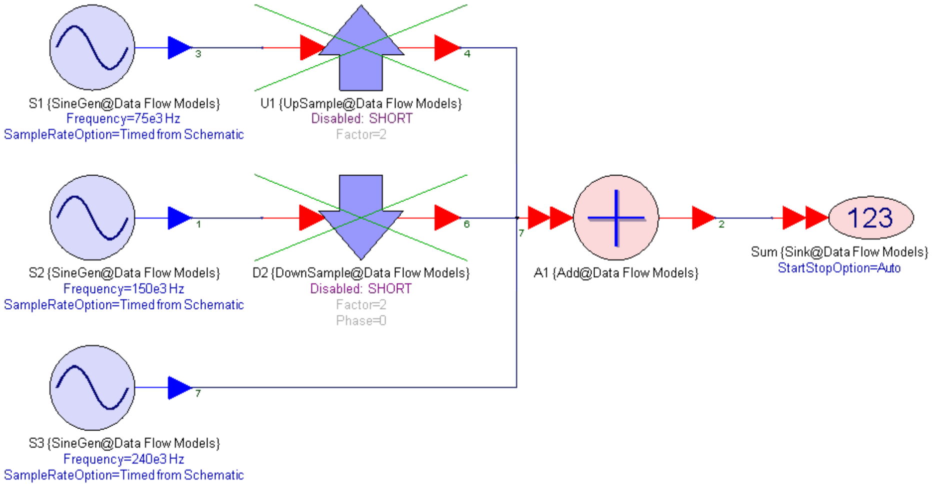


| Index | Part | Model | Domain | Repetition | Port | Direction | DataFlowRate | Delay | Type | SampleRate | TimeStep | StartTime | SyncSamples | Fc |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Complex_to_IQ | CxToRect@Data Flow Models | Untimed | 10 | | | | | | | | | | |
| 2 | | | | | input | Input | 1 | 0 | COMPLEX | 2.50000000e+006 | 4.00000000e-007 | N/A | N/A | N/A |
| 3 | | | | | real | Output | 1 | 0 | REAL | 2.50000000e+006 | 4.00000000e-007 | N/A | N/A | N/A |
| 4 | | | | | imag | Output | 1 | 0 | REAL | 2.50000000e+006 | 4.00000000e-007 | N/A | N/A | N/A |
| 5 | | | | | | | | | | | | | | |
| 6 | M3 | Modulator@Data Flow Models | Timed | 10 | | | | | | | | | | |
| 7 | | | | | input1 | Input | 1 | 0 | REAL | 2.50000000e+006 | 4.00000000e-007 | 3.00000000e-006 | 0 | N/A |
| 8 | | | | | input2 | Input | 1 | 0 | REAL | 2.50000000e+006 | 4.00000000e-007 | 3.00000000e-006 | 0 | N/A |
| 9 | | | | | output | Output | 1 | 0 | ENVELOPE | 2.50000000e+006 | 4.00000000e-007 | 3.00000000e-006 | 0 | 1.00000000e+007 |
| 10 | | | | | quad_output | Output | 1 | 0 | ENVELOPE | 2.50000000e+006 | 4.00000000e-007 | 3.00000000e-006 | 0 | 1.00000000e+007 |
| 11 | | | | | | | | | | | | | | |
| 12 | F1 | BPF_RaisedCosine@Data Flow Models | Timed | 10 | | | | | | | | | | |
| 13 | | | | | input | Input | 1 | 0 | ENVELOPE | 2.50000000e+006 | 4.00000000e-007 | 3.00000000e-006 | 0 | 1.00000000e+007 |
| 14 | | | | | output | Output | 1 | 0 | ENVELOPE | 2.50000000e+006 | 4.00000000e-007 | 3.00000000e-006 | 0 | 1.00000000e+007 |
| 15 | | | | | | | | | | | | | | |
| 16 | SpectrumIFIn | SpectrumAnalyzerEnv@Data Flow Models | Timed | 10 | | | | | | | | | | |
| 17 | | | | | input | Input | 1 | 0 | ENVELOPE | 2.50000000e+006 | 4.00000000e-007 | 3.00000000e-006 | 0 | 1.00000000e+007 |
| 18 | | | | | | | | | | | | | | |
| 19 | IF_to_RF | Mixer@Data Flow Models | Timed | 10 | | | | | | | | | | |
| 20 | | | | | input1 | Input | 1 | 0 | ENVELOPE | 2.50000000e+006 | 4.00000000e-007 | 0.00000000e+000 | 8 | 1.00000000e+007 |
| 21 | | | | | input2 | Input | 1 | 0 | ENVELOPE | 2.50000000e+006 | 4.00000000e-007 | 0.00000000e+000 | 0 | 9.90000000e+008 |
| 22 | | | | | output | Output | 1 | 0 | ENVELOPE | 2.50000000e+006 | 4.00000000e-007 | 0.00000000e+000 | 0 | 1.00000000e+009 |
| 23 | | | | | | | | | | | | | | |
| 24 | SpectrumRF | SpectrumAnalyzerEnv@Data Flow Models | Timed | 10 | | | | | | | | | | |
| 25 | | | | | input | Input | 1 | 0 | ENVELOPE | 2.50000000e+006 | 4.00000000e-007 | 0.00000000e+000 | 0 | 1.00000000e+009 |
| 26 | | | | | | | | | | | | | | |
| 27 | RF_to_IF | Mixer@Data Flow Models | Timed | 10 | | | | | | | | | | |
| 28 | | | | | input1 | Input | 1 | 0 | ENVELOPE | 2.50000000e+006 | 4.00000000e-007 | 0.00000000e+000 | 0 | 1.00000000e+009 |
| 29 | | | | | input2 | Input | 1 | 0 | ENVELOPE | 2.50000000e+006 | 4.00000000e-007 | 0.00000000e+000 | 0 | 9.90000000e+008 |
| 30 | | | | | output | Output | 1 | 0 | ENVELOPE | 2.50000000e+006 | 4.00000000e-007 | 0.00000000e+000 | 0 | 1.00000000e+007 |
| 31 | | | | | | | | | | | | | | |
| 32 | D2 | Demodulator@Data Flow Models | Timed | 10 | | | | | | | | | | |
| 33 | | | | | input | Input | 1 | 0 | ENVELOPE | 2.50000000e+006 | 4.00000000e-007 | 0.00000000e+000 | 0 | 1.00000000e+007 |
| 34 | | | | | output1 | Output | 1 | 0 | REAL | 2.50000000e+006 | 4.00000000e-007 | 0.00000000e+000 | 0 | N/A |
| 35 | | | | | output2 | Output | 1 | 0 | REAL | 2.50000000e+006 | 4.00000000e-007 | 0.00000000e+000 | 0 | N/A |
| 36 | | | | | | | | | | | | | | |
| 37 | R1 | RectToCx@Data Flow Models | Untimed | 10 | | | | | | | | | | |
| 38 | | | | | real | Input | 1 | 0 | REAL | 2.50000000e+006 | 4.00000000e-007 | N/A | N/A | N/A |
| 39 | | | | | imag | Input | 1 | 0 | REAL | 2.50000000e+006 | 4.00000000e-007 | N/A | N/A | N/A |
| 40 | | | | | output | Output | 1 | 0 | COMPLEX | 2.50000000e+006 | 4.00000000e-007 | N/A | N/A | N/A |
| 41 | | | | | | | | | | | | | | |
| 42 | output | Sink@Data Flow Models | Timed | 10 | | | | | | | | | | |
| 43 | | | | | input#1 | Input | 1 | 0 | COMPLEX | 2.50000000e+006 | 4.00000000e-007 | 0.00000000e+000 | 0 | N/A |
| 44 | | | | | | | | | | | | | | |
| 45 | SpectrumIFOut | SpectrumAnalyzerEnv@Data Flow Models | Timed | 10 | | | | | | | | | | |
| 46 | | | | | input | Input | 1 | 0 | ENVELOPE | 2.50000000e+006 | 4.00000000e-007 | 0.00000000e+000 | 0 | 1.00000000e+007 |
| 47 | | | | | | | | | | | | | | |
| 48 | O2 | Oscillator@Data Flow Models | Timed | 10 | | | | | | | | | | |
| 49 | | | | | output | Output | 1 | 0 | ENVELOPE | 2.50000000e+006 | 4.00000000e-007 | 0.00000000e+000 | 0 | 9.90000000e+008 |
| 50 | | | | | | | | | | | | | | |
| 51 | O1 | Oscillator@Data Flow Models | Timed | 10 | | | | | | | | | | |
| 52 | | | | | output | Output | 1 | 0 | ENVELOPE | 2.50000000e+006 | 4.00000000e-007 | 0.00000000e+000 | 0 | 9.90000000e+008 |

## Timed from Schematic

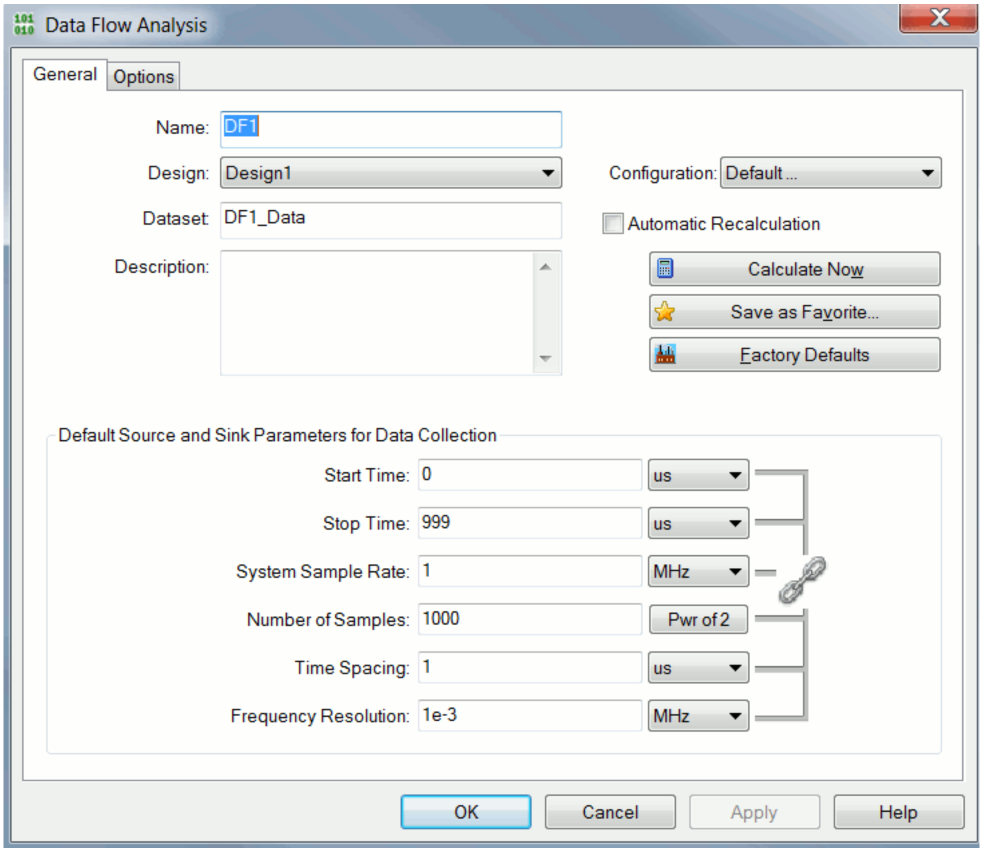
The following example introduces how SystemVue resolves the simulation sampling rate when *SampleRateOption* parameter is set to Timed from Schematic in source models. This example also explains the relation between the *Timed from Schematic* option and the *System Sample Rate* parameter in data flow analysis.

1. Open <SystemVue installation directory>\Examples\Tutorials\DataFlow\TimedSystem.wsv.
2. Open *Design1* under folder *1. Timed from Schematic*.



3
4
S1 {SineGen@Data Flow Models}
Frequency=75e3 Hz
SampleRateOption=Timed from Schematic
U1 {UpSample@Data Flow Models}
Disabled: SHORT
Factor=2
1
6
7
2
123
S2 {SineGen@Data Flow Models}
Frequency=150e3 Hz
SampleRateOption=Timed from Schematic
D2 {DownSample@Data Flow Models}
Disabled: SHORT
Factor=2
Phase=0
A1 {Add@Data Flow Models}
Sum {Sink@Data Flow Models}
StartStopOption=Auto
S3 {SineGen@Data Flow Models}
Frequency=240e3 Hz
SampleRateOption=Timed from Schematic

3. There are three SineGen source models *S1*, *S2*, and *S3* on schematic. Their *Frequency* parameters are set to 75 Khz, 150 Khz, and 240 Khz respectively. The *SampleRateOption* parameters of these three models are all set to *Timed from Schematic*. You can see the *SampleRateOption* parameter by setting the *ShowAdvancedParams* option to YES. By default, the *SampleRateOption* parameter is set to Timed from Schematic.
4. Run *DF1* data flow analysis.
5. When there is no source model that explicitly sets the output sampling rate (i.e., *SampleRateOption = Timed from SampleRate*) and there is no SetSampleRate model on schematic, SystemVue will set the System Sample Rate (defined in the data flow analysis) to the output connections of the sources that require the lowest sampling rate based on the TSDF sampling rate equations. After that, SystemVue can resolve the sampling rates for all other connections. See sampling rate resolution for the exact rules.
6. Since *Design1* is currently a single rate system (the UpSample and DownSample are shorted), the sources *S1*, *S2*, and *S3* operate at the same rate. As a result, SystemVue sets the output sampling rate of these three models to the System Sample Rate defined in *DF1* data flow analysis, which is 1 Mhz in this case.



Name: DF1

Design: Design1

Configuration: Default ...

Dataset: DF1_Data

□ Automatic Recalculation

Description:

Default Source and Sink Parameters for Data Collection

Start Time: 0 us

Stop Time: 999 us

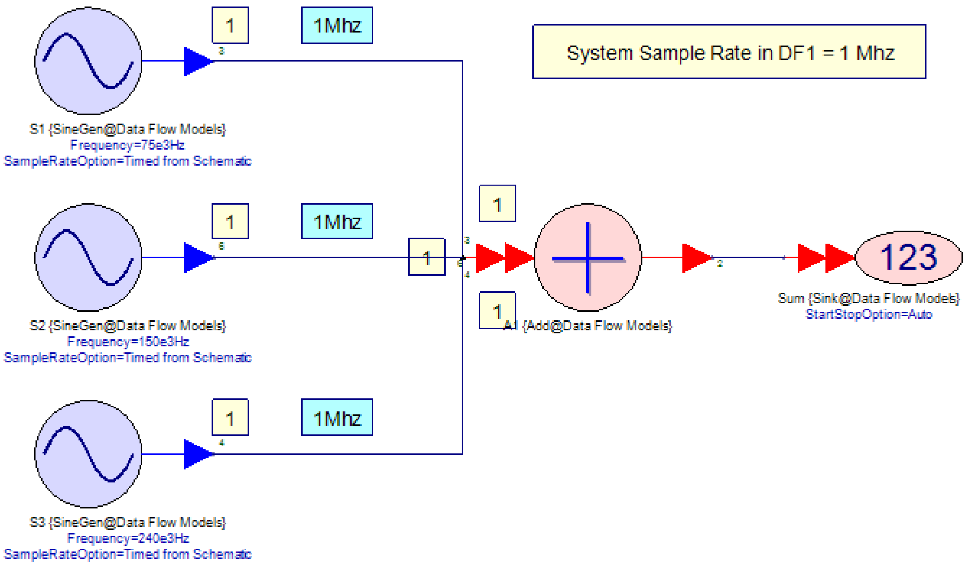
System Sample Rate: 1 MHz

Number of Samples: 1000

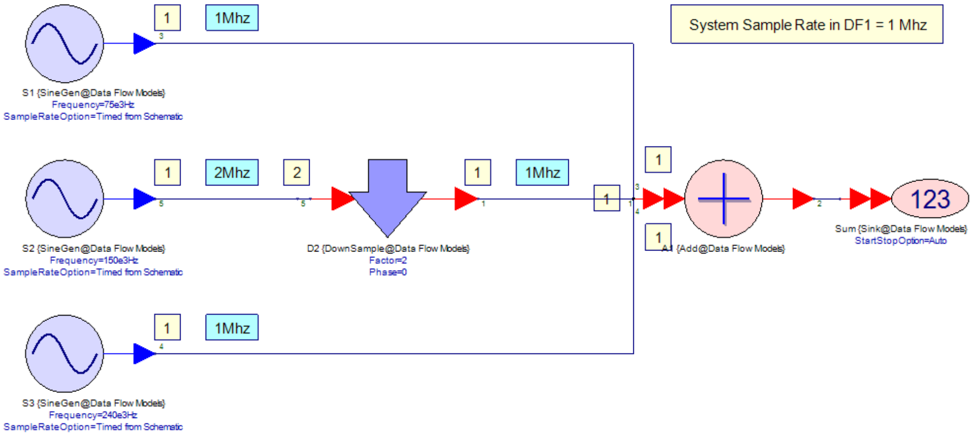
Time Spacing: 1 us

Frequency Resolution: 1e-3 MHz

7. The following screenshot annotates the data flow rates and sampling rates in the system. You can verify them in *DF1_Data_DataFlowInfo* table.

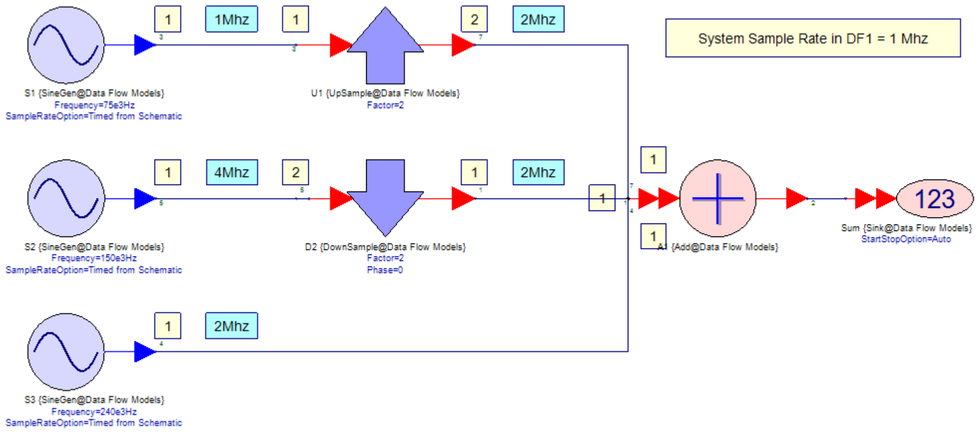

1
1Mhz
System Sample Rate in DF1 = 1 Mhz
S1 {SineGen@Data Flow Models}
Frequency=75e3Hz
SampleRateOption=Timed from Schematic
1
1Mhz
1
1
123
1
A1 {Add@Data Flow Models}
Sum {Sink@Data Flow Models}
StartStopOption=Auto
S2 {SineGen@Data Flow Models}
Frequency=150e3Hz
SampleRateOption=Timed from Schematic
1
1Mhz
S3 {SineGen@Data Flow Models}
Frequency=240e3Hz
SampleRateOption=Timed from Schematic

8. Now enable DownSample *D2* and run *DF1* data flow analysis again.
9. In this case, because there is a 2-to-1 DownSample *D2* between *S2* and the rest of the system, *S2* operates at twice the rate of the other models. You can verify it in *DF1_Data_DataFlowInfo* table. In the table, the repetition count (column repetition) of *S2* is 2, while the repetition counts for all other models are 1. This multirate behavior makes *S1* and *S3* the slowest sources in the system, so SystemVue sets the System Sample Rate (which is defined as 1 Mhz in *DF1* data flow analysis) to the output of *S1* and *S2*. Based on the TSDF sampling rate equations, the sampling rate for *S2* is now 2 Mhz. This can be easily derived as follows: 1) The sampling rate for *S3*'s output is 1 Mhz, which is the sampling rate at the bottom input of the Add model. 2) Since Add is a single rate model, the sampling rate of the middle input of the Add model is also 1 Mhz. 3) Because there is a 2-to-1 down sampling across *D2*, the input sampling rate of *D2* should be 2 Mhz, which makes the output sampling rate of *S2* 2 Mhz. Can you annotate the data flow rates and sampling rates for the current system? The answer is provided in the following screenshot.

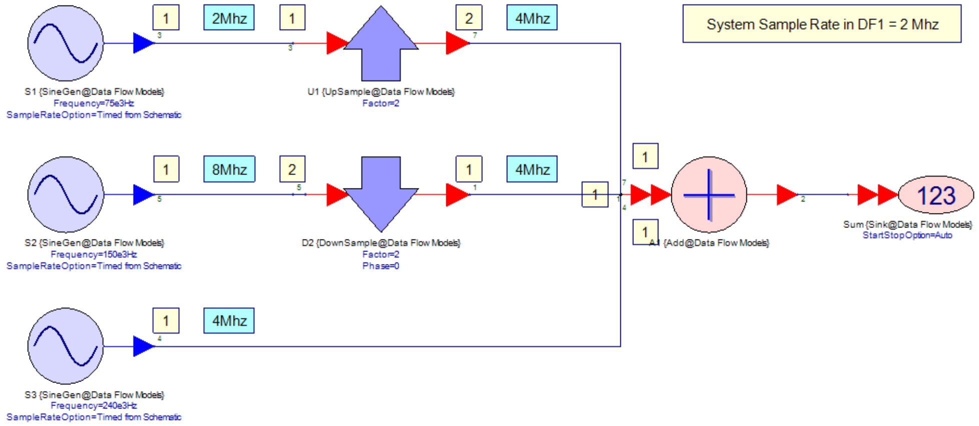

1
1Mhz
System Sample Rate in DF1 = 1 Mhz
S1 {SineGen@Data Flow Models}
Frequency=75e3Hz
SampleRateOption=Timed from Schematic
1
2Mhz
2
1
1Mhz
1
1
123
1
D2 {DownSample@Data Flow Models}
Factor=2
Phase=0
A1 {Add@Data Flow Models}
Sum {Sink@Data Flow Models}
StartStopOption=Auto
S2 {SineGen@Data Flow Models}
Frequency=150e3Hz
SampleRateOption=Timed from Schematic
1
1Mhz
S3 {SineGen@Data Flow Models}
Frequency=240e3Hz
SampleRateOption=Timed from Schematic

10. Now enable UpSample *U1* and run *DF1* data flow analysis again.

11. In this case, because there is a 1-to-2 UpSample *U1* between *S1* and the rest of the system, *S1* now operates at the slowest rates. From *DF1_Data_DataFlowInfo* table, you can verify that the repetition counts of *S1*, *S2*, and *S3* are 1, 4, and 2 respectively. SystemVue now sets the *System Sample Rate* (which is defined as 1 Mhz in *DF1* Data Flow Analysis) to the output of *S1*. Can you annotate the data flow rates and sampling rates for the current system? The answer is provided in the following screenshot. Note that the output sampling rates for *S1*, *S2*, and *S3* are now 1 Mhz, 4 Mhz, and 2 Mhz respectively.


1
1Mhz
1
2
2Mhz
System Sample Rate in DF1 = 1 Mhz
S1 {SineGen@Data Flow Models}
Frequency=75e3Hz
SampleRateOption=Timed from Schematic
U1 {UpSample@Data Flow Models}
Factor=2
1
4Mhz
2
1
2Mhz
1
1
1
123
S2 {SineGen@Data Flow Models}
Frequency=150e3Hz
SampleRateOption=Timed from Schematic
D2 {DownSample@Data Flow Models}
Factor=2
Phase=0
A1 {Add@Data Flow Models}
Sum {Sink@Data Flow Models}
StartStopOption=Auto
1
2Mhz
S3 {SineGen@Data Flow Models}
Frequency=240e3Hz
SampleRateOption=Timed from Schematic

12. Now double click *DF1* data flow analysis and set *System Sample Rate* to 2 Mhz and run *DF1* data flow analysis again. Since the *System Sample Rate* is now set to 2 Mhz, the output sampling rates for *S1*, *S2*, and *S3* are now 2 Mhz, 8 Mhz, and 4 Mhz respectively. Can you annotate the data flow rates and sampling rates for the current system? The answer is provided in the following screenshot.

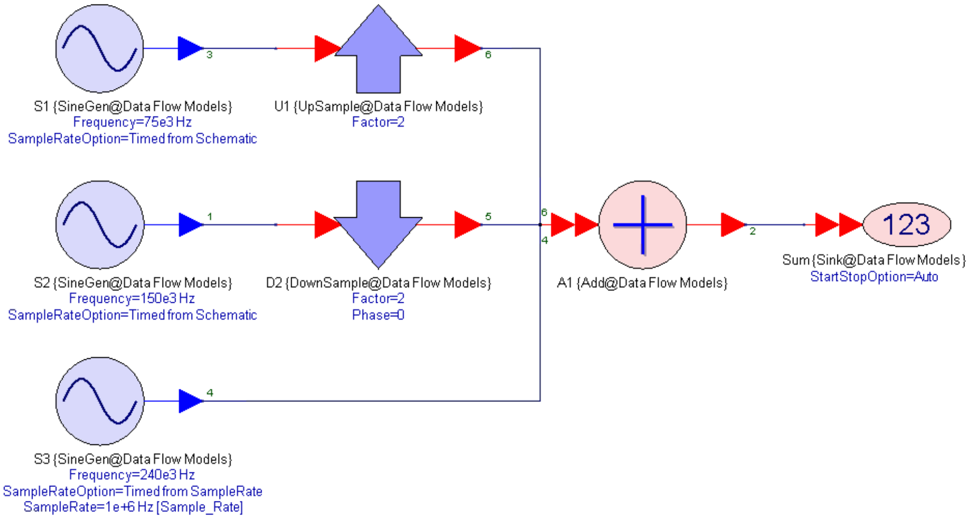

1
2Mhz
1
2
4Mhz
System Sample Rate in DF1 = 2 Mhz
S1 {SineGen@Data Flow Models}
Frequency=75e3Hz
SampleRateOption=Timed from Schematic
U1 {UpSample@Data Flow Models}
Factor=2
1
8Mhz
2
1
4Mhz
1
1
1
123
S2 {SineGen@Data Flow Models}
Frequency=150e3Hz
SampleRateOption=Timed from Schematic
D2 {DownSample@Data Flow Models}
Factor=2
Phase=0
A1 {Add@Data Flow Models}
Sum {Sink@Data Flow Models}
StartStopOption=Auto
1
4Mhz
S3 {SineGen@Data Flow Models}
Frequency=240e3Hz
SampleRateOption=Timed from Schematic

### Timed from SampleRate

The following example introduces how SystemVue resolves the simulation sampling rate when there exists a source model with *SampleRateOption* parameter set to Timed from Schematic.

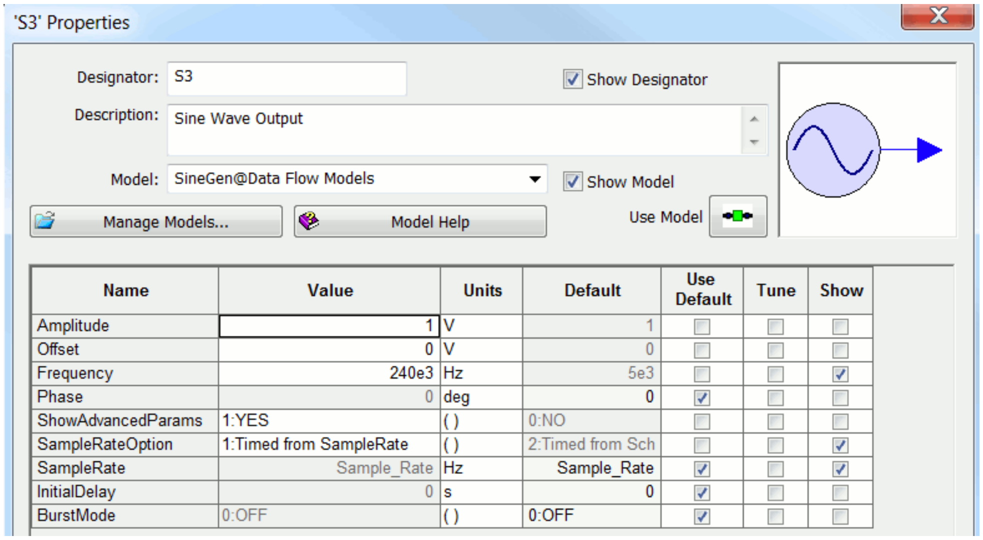
1. Open <SystemVue installation directory>\Examples\Tutorials\DataFlow\TimedSystem.wsv.

2. Open schematic *Design2* under folder *2. Timed from SampleRate*. *Design2* is continued from the tutorial example Timed from Schematic.



3
6
S1 {SineGen@Data Flow Models}
Frequency=75e3 Hz
SampleRateOption=Timed from Schematic
U1 {UpSample@Data Flow Models}
Factor=2
1
5
6
4
2
123
S2 {SineGen@Data Flow Models}
Frequency=150e3 Hz
SampleRateOption=Timed from Schematic
D2 {DownSample@Data Flow Models}
Factor=2
Phase=0
A1 {Add@Data Flow Models}
Sum {Sink@Data Flow Models}
StartStopOption=Auto
4
S3 {SineGen@Data Flow Models}
Frequency=240e3 Hz
SampleRateOption=Timed from SampleRate
SampleRate=1e+6 Hz [Sample_Rate]

3. The *SampleRateOption* parameter of the SineGen model *S3* is set to Timed from SampleRate. What it means is that *S3* now explicitly sets its output sampling rate to the value specified in the *SampleRate* parameter. By default, the *SampleRate* parameter is set to the global equation variable *Sample_Rate*, which is used to access the *System Sample Rate* value in the corresponding data flow analysis. In this case, *Sample_Rate* equals to 1 Mhz, which is the *System Sample Rate* defined in *DF2* data flow analysis. See Accessing Data Flow Analysis Settings from Equations for details.



'S3' Properties

Designator: S3

☑ Show Designator

Description: Sine Wave Output

Model: SineGen@Data Flow Models

☑ Show Model

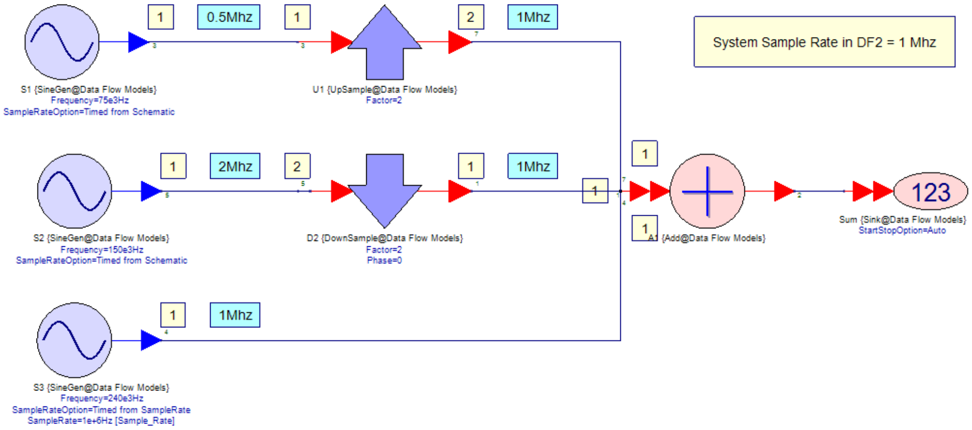
Use Model

| Name | Value | Units | Default | Use Default | Tune | Show |
|---|---|---|---|---|---|---|
| Amplitude | 1 | V | 1 | ☐ | ☐ | ☐ |
| Offset | 0 | V | 0 | ☐ | ☐ | ☐ |
| Frequency | 240e3 | Hz | 5e3 | ☐ | ☐ | ☑ |
| Phase | 0 | deg | 0 | ☑ | ☐ | ☐ |
| ShowAdvancedParams | 1:YES | ( ) | 0:NO | ☐ | ☐ | ☐ |
| SampleRateOption | 1:Timed from SampleRate | ( ) | 2:Timed from Sch | ☐ | ☐ | ☑ |
| SampleRate | Sample_Rate | Hz | Sample_Rate | ☑ | ☐ | ☑ |
| InitialDelay | 0 | s | 0 | ☑ | ☐ | ☐ |
| BurstMode | 0:OFF | ( ) | 0:OFF | ☑ | ☐ | ☐ |

4. If at least one model sets the sampling rate on a particular connection, SystemVue can compute the sampling rates for all other connections based on the TSDF sampling rate equations. See sampling rate resolution for the exact rules.

5. In *Design2*, because *S3* explicitly sets the output sampling rate to 1 Mhz, SystemVue will use that information to resolve the sampling rates for the rest of the system. Can you annotate the data flow rates and sampling rates for

the current system? The answer is provided in the following screenshot. The sampling rates for the rest of the system can be easily derived as follows: 1) *S3*'s output sampling rate is explicitly set to 1 Mhz. 2) The Add model is a single rate model, so the output sampling rates for DownSample *D2* and UpSample *U1* must be 1 Mhz. 3) There is a 2-to-1 down sampling across *D2*, so the output sampling rate for *S2* is 2 Mhz. 4) There is a 1-to-2 up sampling across *U1*, so the sampling rate for *S1* is 0.5 Mhz. You can verify it using the *DF2_Data_DataFlowInfo* table. The repetition count for *S1*, *S2*, and *S3* are 1, 4, and 2 respectively.

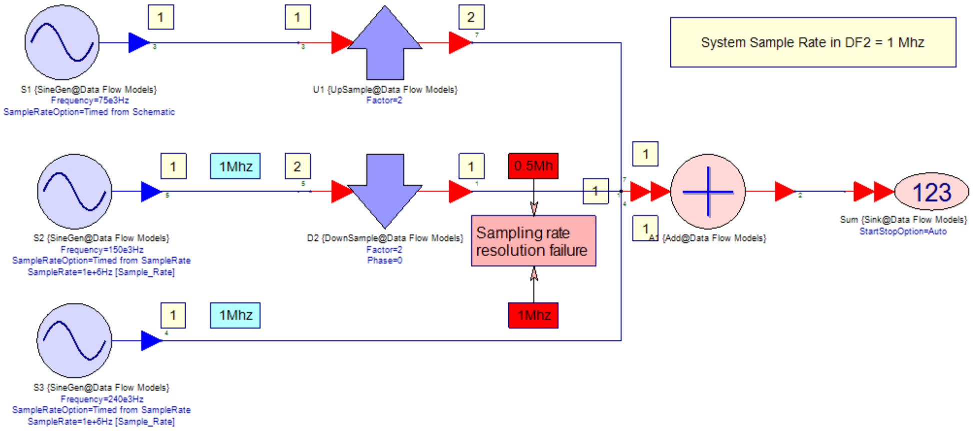

1
0.5Mhz
1
2
1Mhz
System Sample Rate in DF2 = 1 Mhz
S1 {SineGen@Data Flow Models}
Frequency=75e3Hz
SampleRateOption=Timed from Schematic
U1 {UpSample@Data Flow Models}
Factor=2
1
2Mhz
2
1
1Mhz
1
1
123
1
S2 {SineGen@Data Flow Models}
Frequency=150e3Hz
SampleRateOption=Timed from Schematic
D2 {DownSample@Data Flow Models}
Factor=2
Phase=0
A1 {Add@Data Flow Models}
Sum {Sink@Data Flow Models}
StartStopOption=Auto
1
1Mhz
S3 {SineGen@Data Flow Models}
Frequency=240e3Hz
SampleRateOption=Timed from SampleRate
SampleRate=1e+6Hz [Sample_Rate]

6. Now double click on *S2* to bring up the part properties dialog, change *SampleRateOption* to *Timed from SampleRate*. By default, the *SampleRate* parameter for *S2* is set to *Sample_Rate* global variable, which is 1 Mhz because it refers to the *System Sample Rate* parameter in *DF2* data flow analysis. Click OK to enforce the settings.

7. Run *DF2* data flow analysis.

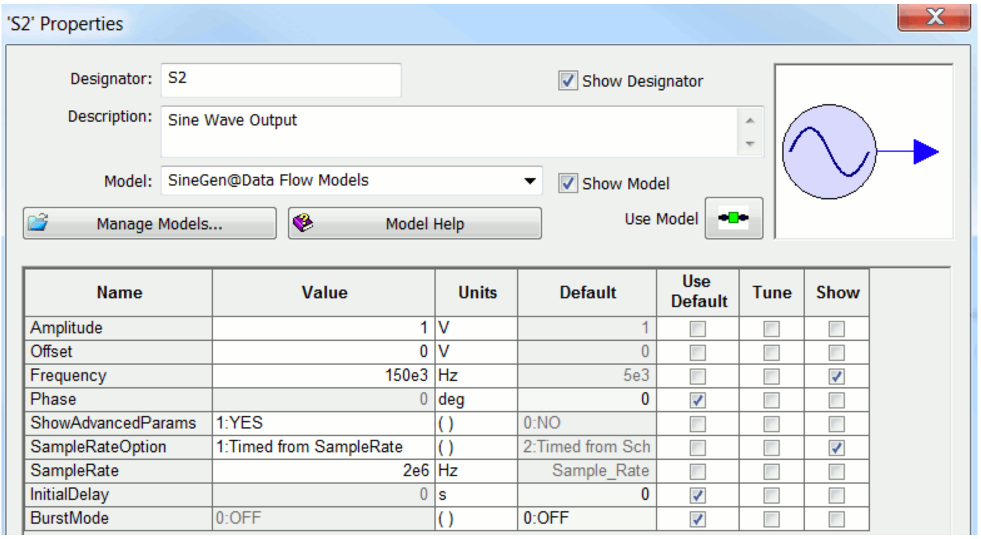
8. You should see an error message "Error: output of SineGeneratorTimedBurst 'Design2__S2': The port Design2__S2__output has a sample rate set to 1.0 e+006 but was resolved as needing to be set to 2.0e+006". The error happens because both *S3* and *S2* want to set the output sampling rate to 1 Mhz but they actually operate at different rates due to DownSample *D2*, SystemVue cannot resolve sampling rates for the system.

**CAUTION** **When there is an error during simulation setup, SystemVue is unable to provide data flow information.**


1
1
2
System Sample Rate in DF2 = 1 Mhz
S1 {SineGen@Data Flow Models}
Frequency=75e3Hz
SampleRateOption=Timed from Schematic
U1 {UpSample@Data Flow Models}
Factor=2
1
1Mhz
2
1
0.5Mh
1
1
123
S2 {SineGen@Data Flow Models}
Frequency=150e3Hz
SampleRateOption=Timed from SampleRate
SampleRate=1e+6Hz [Sample_Rate]
D2 {DownSample@Data Flow Models}
Factor=2
Phase=0
Sampling rate resolution failure
1
A1 {Add@Data Flow Models}
Sum {Sink@Data Flow Models}
StartStopOption=Auto
1
1Mhz
1Mhz
S3 {SineGen@Data Flow Models}
Frequency=240e3Hz
SampleRateOption=Timed from SampleRate
SampleRate=1e+6Hz [Sample_Rate]

9. When there are multiple models that set the sampling rates in a system, it is user's responsibility to make sure the TSDF sampling rate equations have a solution. If not, SystemVue will issue error messages to help users identify the problematic connections and the required sampling rate values. Can you fix this sampling rate resolution issue?

10. You can fix the problem by setting the *SampleRate* parameter of *S2* to 2 Mhz.



'S2' Properties

Designator: S2

☑ Show Designator

Description: Sine Wave Output

Model: SineGen@Data Flow Models

☑ Show Model

| Name | Value | Units | Default | Use Default | Tune | Show |
|---|---|---|---|---|---|---|
| Amplitude | 1 | V | 1 | □ | □ | □ |
| Offset | 0 | V | 0 | □ | □ | □ |
| Frequency | 150e3 | Hz | 5e3 | □ | □ | ☑ |
| Phase | 0 | deg | 0 | ☑ | □ | □ |
| ShowAdvancedParams | 1:YES | ( ) | 0:NO | □ | □ | □ |
| SampleRateOption | 1:Timed from SampleRate | ( ) | 2:Timed from Sch | □ | □ | ☑ |
| SampleRate | 2e6 | Hz | Sample_Rate | □ | □ | □ |
| InitialDelay | 0 | s | 0 | ☑ | □ | □ |
| BurstMode | 0:OFF | ( ) | 0:OFF | ☑ | □ | □ |

## Frequency Response vs Sample Rate

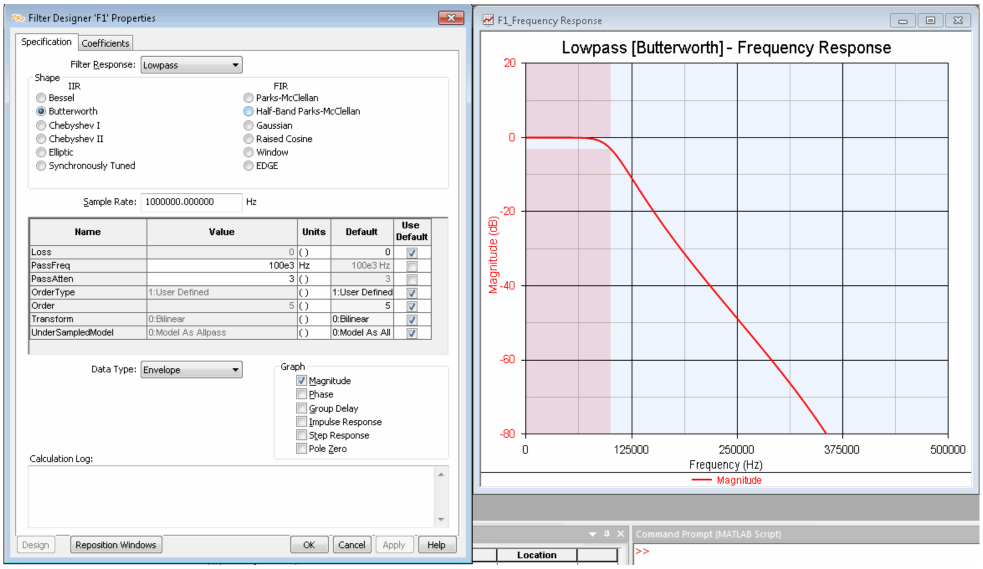
This example will guide you to create an envelope data type lowpass filter and the corresponding numeric filter. By changing the simulation sampling rate other than the sampling rate specified in the filter designer, you will observe different behavior from the two filters.

1. Open <SystemVue installation directory>\Examples\Tutorials\Algorithm_Design\DataFlow\FilterAndSampleR wsv.

2. Open schematic *Design1* in folder *1. Frequency Response vs Sample Rate*.

3. Create and design a lowpass Butterworth IIR filter. First, place a filter part on schematic, double click it to open a filter designer, select *Lowpass* response and *Butterworth* IIR shape. Set *Sample Rate* parameter in the filter designer to 1 MHz (1e6 Hz). By default, the *Sample Rate* parameter is set to the *System Sample Rate* in the data flow analysis. Set *PassFreq* to 150 KHz and leave the rest parameters as default. Make sure the *Data Type* drop-down menu is set to *Envelope*, which is the default. Close the filter designer. The filter designer will automatically design the filter and show you the lowpass frequency response as in the screenshot below.

CAUTION The actual simulation sampling rate at the input of the filter can be different than the *Sample Rate* parameter in the corresponding filter designer.



Filter Designer 'F1' Properties

Specification | Coefficients

Filter Response: Lowpass

Shape

IIR: ○ Bessel, ● Butterworth, ○ Chebyshev I, ○ Chebyshev II, ○ Elliptic, ○ Synchronously Tuned

FIR: ○ Parks-McClellan, ○ Half-Band Parks-McClellan, ○ Gaussian, ○ Raised Cosine, ○ Window, ○ EDGE

Sample Rate: 1000000.000000 Hz

| Name | Value | Units | Default | Use Default |
|---|---|---|---|---|
| Loss | 0 | () | 0 | ☑ |
| PassFreq | 100e3 | Hz | 100e3 Hz | □ |
| PassAtten | 3 | () | 3 | □ |
| OrderType | 1:User Defined | () | 1:User Defined | ☑ |
| Order | 5 | () | 5 | ☑ |
| Transform | 0:Bilinear | () | 0:Bilinear | ☑ |
| UnderSampledModel | 0:Model As Allpass | () | 0:Model As All | ☑ |

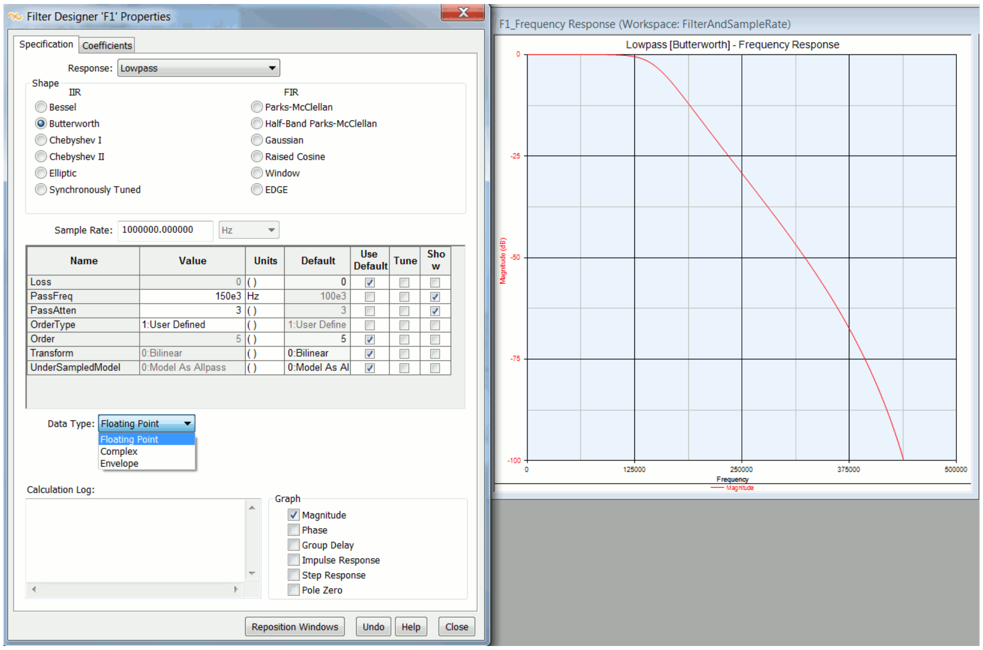
Data Type: Envelope

Graph: ☑ Magnitude, □ Phase, □ Group Delay, □ Impulse Response, □ Step Response, □ Pole Zero

Calculation Log:

4. Close the filter designer. The filter part will be configured automatically to use the LPF Butterworth model based on the specification. Name the filter part *F1* and insert *F1* in between the Impulse source *I1* and the Sink *TimedFilterOutput*.

5. Copy and paste the LPF Butterworth part to another part. Double click the copied part and click "Filter Designer" in the part properties dialog. This will bring up a filter designer containing the same lowpass Butterworth specification. Make sure again the *Sample Rate* in the filter designer is 1 Mhz and *PassFreq* is 150 KHz. Choose *Floating Point* in the *Data Type* drop-down menu. See Data Type section in Filter Specification Window.



Filter Designer 'F1' Properties

Specification | Coefficients

Response: Lowpass

Shape

IIR
- ○ Bessel
- ● Butterworth
- ○ Chebyshev I
- ○ Chebyshev II
- ○ Elliptic
- ○ Synchronously Tuned

FIR
- ○ Parks-McClellan
- ○ Half-Band Parks-McClellan
- ○ Gaussian
- ○ Raised Cosine
- ○ Window
- ○ EDGE

Sample Rate: 1000000.000000 Hz

| Name | Value | Units | Default | Use Default | Tune | Show |
|---|---|---|---|---|---|---|
| Loss | 0 | ( ) | 0 | ☑ | □ | □ |
| PassFreq | 150e3 | Hz | 100e3 | □ | □ | ☑ |
| PassAtten | 3 | ( ) | 3 | □ | □ | ☑ |
| OrderType | 1:User Defined | ( ) | 1:User Define | □ | □ | □ |
| Order | 5 | ( ) | 5 | ☑ | □ | □ |
| Transform | 0:Bilinear | ( ) | 0:Bilinear | ☑ | □ | □ |
| UnderSampledModel | 0:Model As Allpass | ( ) | 0:Model As Al | ☑ | □ | □ |

Data Type: Floating Point

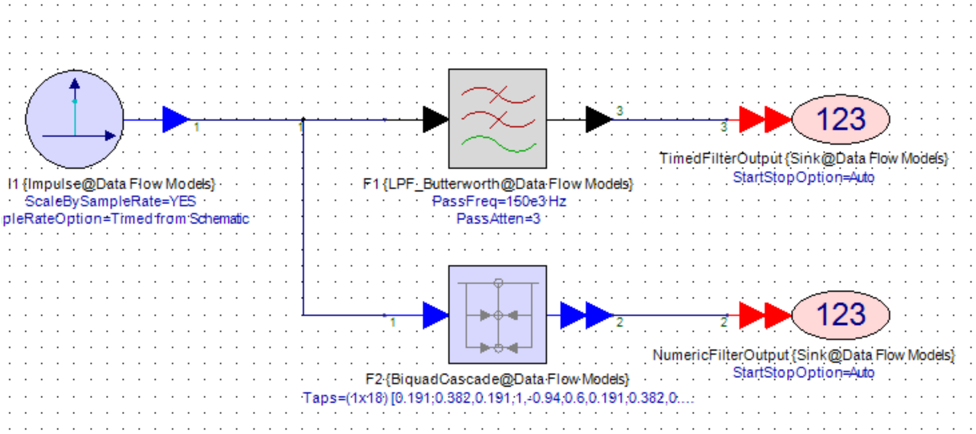
Floating Point
Complex
Envelope

Calculation Log:

Graph
- ☑ Magnitude
- □ Phase
- □ Group Delay
- □ Impulse Response
- □ Step Response
- □ Pole Zero

6. Close the filter designer. Because the filter specification is IIR with *Floating Point* data type, a BiquadCascade model is instantiated when the IIR coefficients can fit into the cascade-biquad structure (otherwise, an IIR model is instantiated). (Cascade-biquad structure has better resistance to numeric errors). Name the resulting part *F2* and insert *F2* in between the Impulse source *I1* and the Sink *NumericFilterOutput*.

7. The resulting schematic should look as follows.

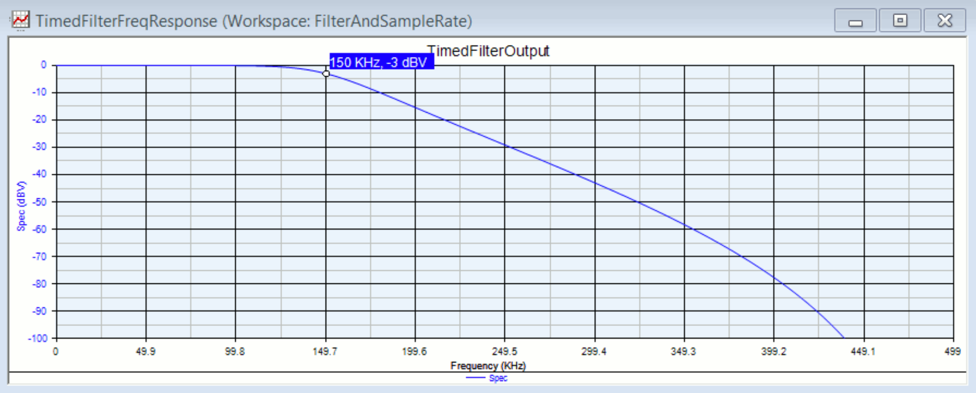


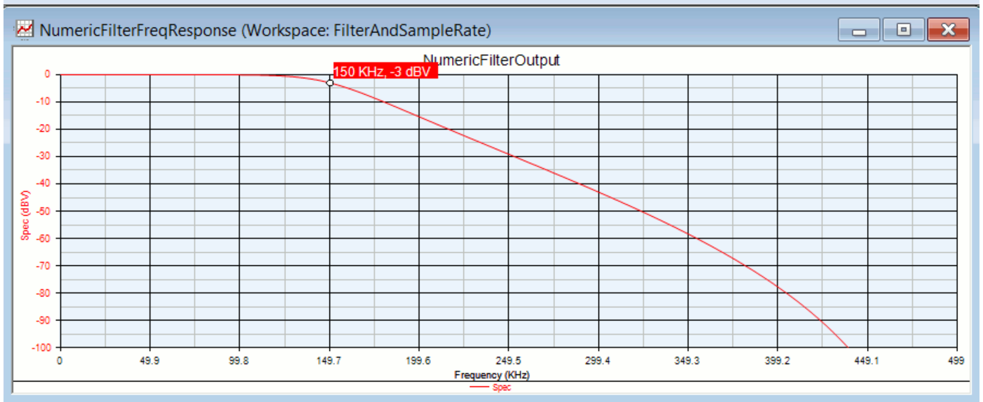
1
1
3
3
123
I1 {Impulse@Data Flow Models}
ScaleBySampleRate=YES
pleRateOption=Timed from Schematic
F1 {LPF_Butterworth@Data Flow Models}
PassFreq=150e3 Hz
PassAtten=3
TimedFilterOutput {Sink@Data Flow Models}
StartStopOption=Auto
1
2
2
123
F2 {BiquadCascade@Data Flow Models}
Taps=(1x18) [0.191;0.382,0.191;1,-0.94;0.6,0.191;0.382,0...
NumericFilterOutput {Sink@Data Flow Models}
StartStopOption=Auto

8. Run *DF1* data flow analysis.

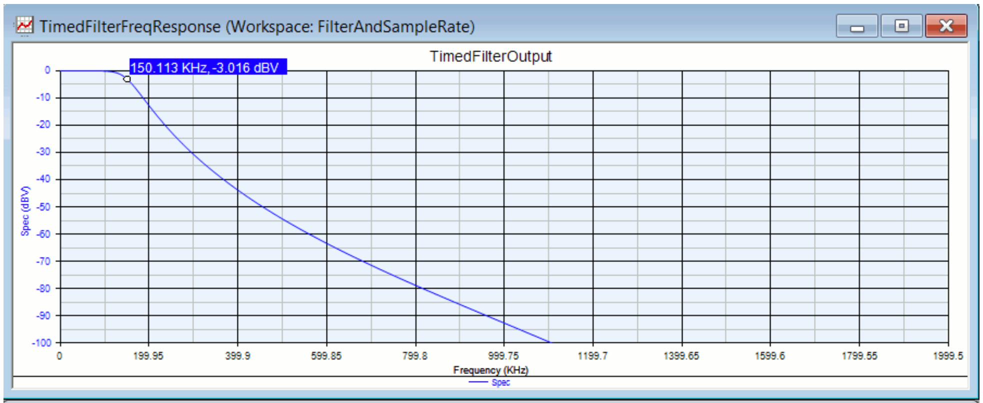
9. Graph *TimedFilterFreqResponse* (blue color) shows the frequency response of the LPF Butterworth model *F1*, i.e., the Fourier transform of the impulse response *TimedFilterOutput*. Graph *NumericFilterFreqResponse* (red color) shows the frequency response of the BiquadCascade model *F2*, i.e., the Fourier transform of the impulse response *NumericFilterOutput*. As you can see, the frequency responses from both filter models are the same. The -3db

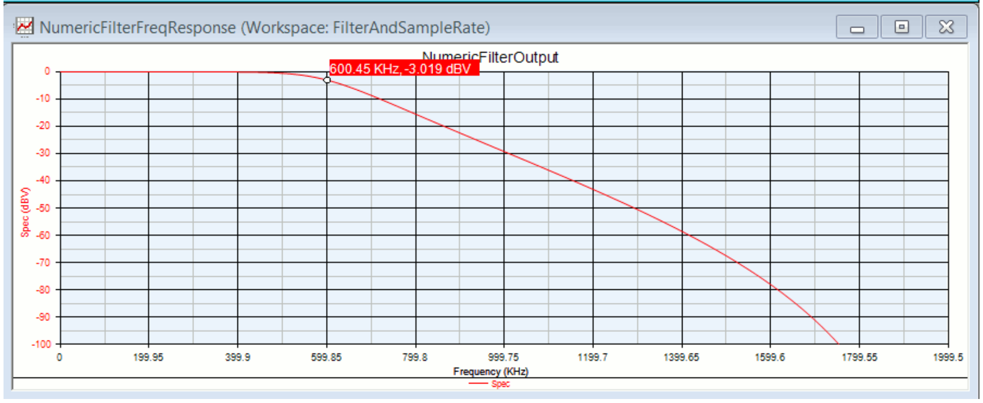
passband edge is at 150 KHz. This is because the numeric filter *F2* is designed to the same specification as the envelope filter *F1* and the *Sample Rate* used to design *F2* is the same as the simulation sampling rate at the input of *F1*.





10. Double click *DF1* data flow analysis and set the *System Sample Rate* parameter to 4 Mhz. As described in the tutorial example Timed from Schematic, the output sampling rate of the Impulse model *I1* will be set to 4 Mhz.
11. Run *DF1* data flow analysis again.
12. Now graph *TimedFilterFreqResponse* is different than graph *NumericFilterFreqResponse*. Since *F1* (LPF Butterworth) is an envelope data type filter model, it will re-design the filter based on the input sampling rate, which is 4 Mhz in this case. As you can see in graph *TimedFilterFreqResponse* , the -3db passband edge is still at 150 KHz and maximum frequency response frequency is at 2 Mhz (1/2 of the sampling rate).

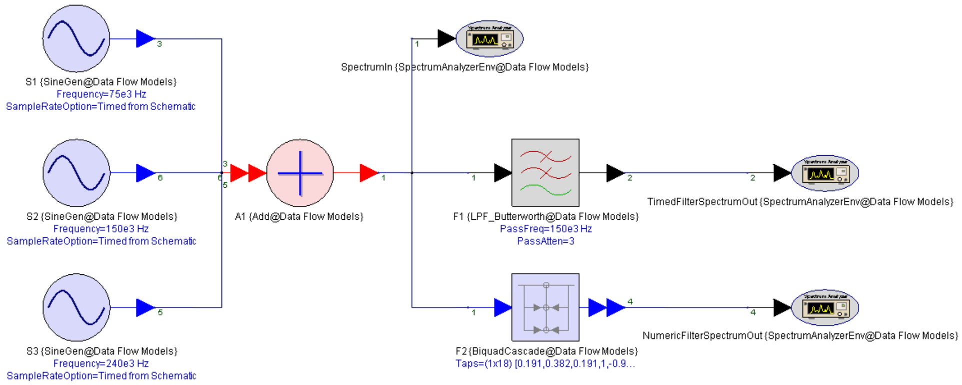




13. The BiquadCascade model *F2* is a numeric digital IIR filter with fixed coefficients. The filter coefficients of *F2* are designed based on 150 KHz *PassFreq* and 1 MHz *Sample Rate*, which actually maps the passband edge to 3π/10 (= 150 Khz / 1 Mhz * 2π) in Z domain. Since 2π in Z domain always maps to the sampling rate of the signal, when the sampling rate changes to 4 Mhz, the passband edge reflects back to 600 KHz (= 3π/10 / 2π * 4 Mhz). This is why in graph *NumericFilterFreqResponse* (frequency response of *F2*), the -3db passband edge is at 600 KHz and the shape of the frequency response remains the same with respect to the simulation sampling rate (2π).

## Filtering vs Sample Rate

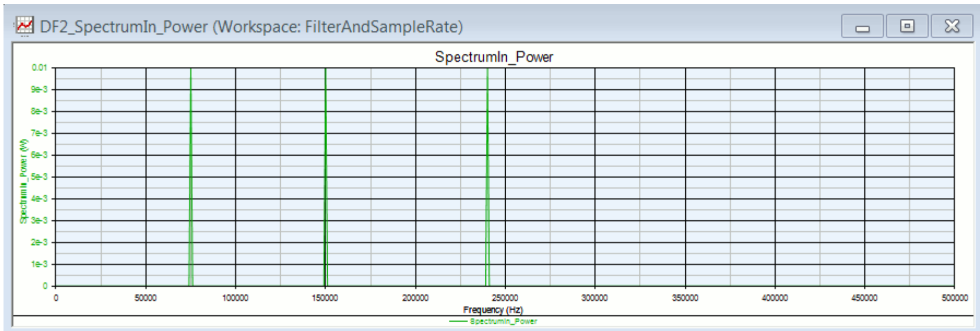
1. Open <SystemVue installation directory>\Examples\Tutorials\Algorithm_Design\DataFlow\FilterAndSampleR wsv.
2. Open schematic *Design2* in folder *2. Filtering vs Sample Rate*. This example is continued from the previous one Frequency Response vs Sample Rate. The LPF Butterworth model *F1* has *PassFreq* = 150 Khz and BiquadCascade model *F2* is designed using lowpass Butterworth specification with *PassFreq* = 150 Khz and *Sample Rate* = 1 Mhz.

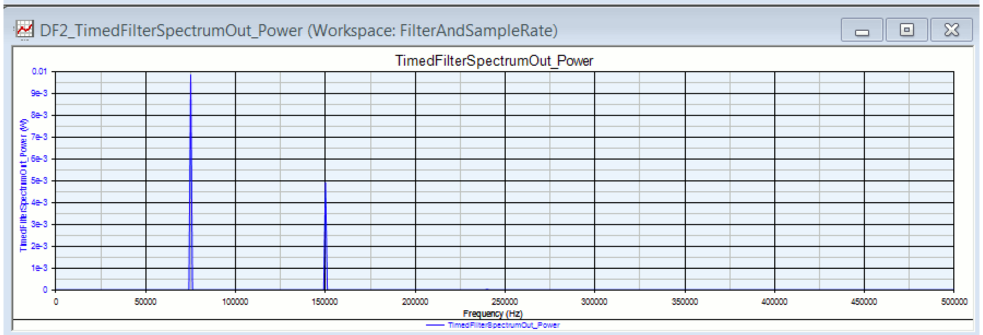

S1 {SineGen@Data Flow Models}
Frequency=75e3 Hz
SampleRateOption=Timed from Schematic
S2 {SineGen@Data Flow Models}
Frequency=150e3 Hz
SampleRateOption=Timed from Schematic
S3 {SineGen@Data Flow Models}
Frequency=240e3 Hz
SampleRateOption=Timed from Schematic
A1 {Add@Data Flow Models}
SpectrumIn {SpectrumAnalyzerEnv@Data Flow Models}
F1 {LPF_Butterworth@Data Flow Models}
PassFreq=150e3 Hz
PassAtten=3
TimedFilterSpectrumOut {SpectrumAnalyzerEnv@Data Flow Models}
F2 {BiquadCascade@Data Flow Models}
Taps=(1x18) [0.191,0.382,0.191,1,-0.9...
NumericFilterSpectrumOut {SpectrumAnalyzerEnv@Data Flow Models}

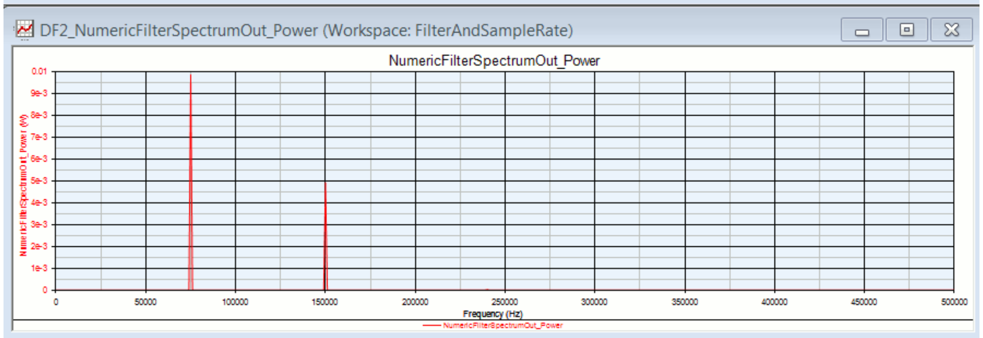
3. The *Frequency* parameters of the three SineGen source models are 75 KHz, 150 KHz, and 240 KHz. The three sine waveforms are combined together and passed to the filters.

4. Run *DF2* data flow analysis with *System Sample Rate* = 1 Mhz.

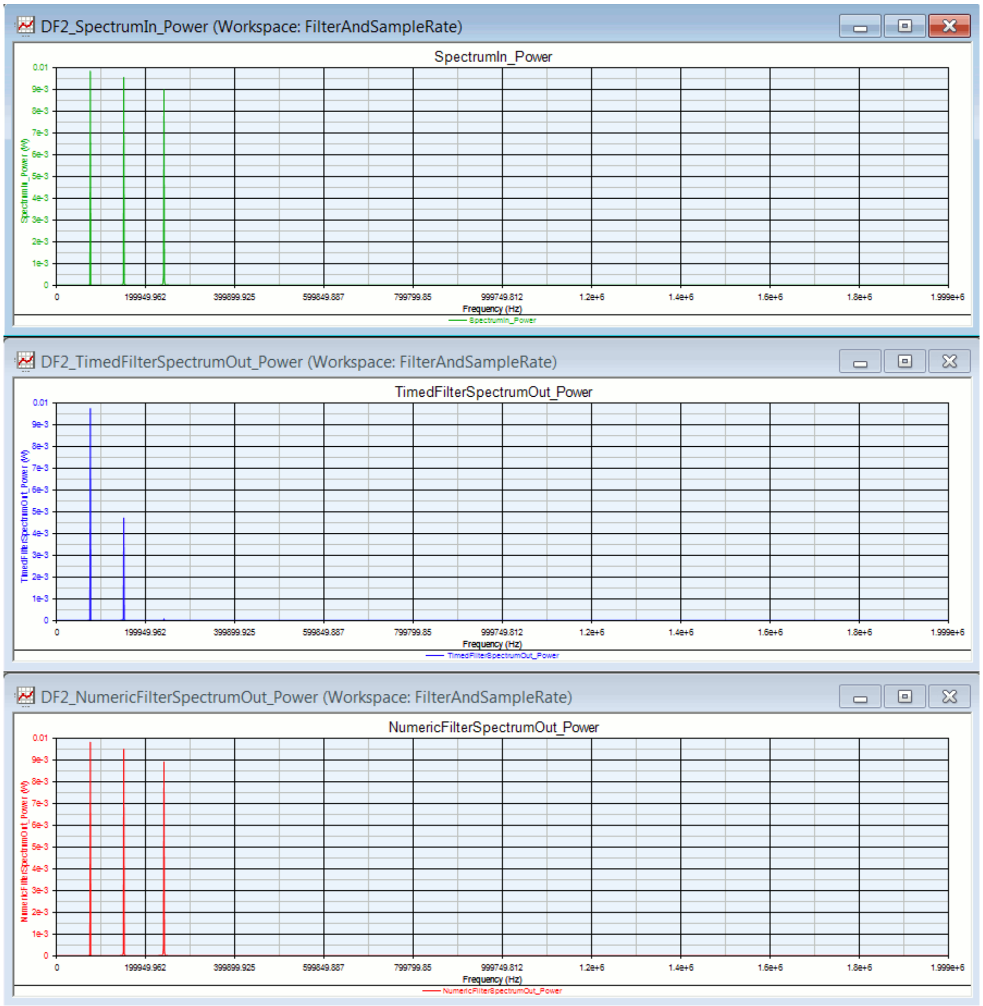
5. Graph *DF2_SpectrumIn_Power* (green color) shows the spectrum of the input signal, graph *DF2_TimedFilterSpectrumOut_Power* (blue color) shows the output spectrum of the envelope LPF Butterworth filter *F1*, and graph *DF2_NumericFilterSpectrumOut_Power* (red color) shows the output spectrum of the numeric BiquadCascade filter *F2*. Since the -3db passband edge is at 150 KHz, the 240 KHz sinusoid is filtered out by *F1* and *F2*. Note that the 1 MHz simulation sampling rate matches the sampling rate used to design the numeric filter *F2*.




DF2_TimedFilterSpectrumOut_Power (Workspace: FilterAndSampleRate)
TimedFilterSpectrumOut_Power
TimedFilterSpectrumOut_Power (W)
0.01
9e-3
8e-3
7e-3
6e-3
5e-3
4e-3
3e-3
2e-3
1e-3
0
0
50000
100000
150000
200000
250000
300000
350000
400000
450000
500000
Frequency (Hz)
TimedFilterSpectrumOut_Power


DF2_NumericFilterSpectrumOut_Power (Workspace: FilterAndSampleRate)
NumericFilterSpectrumOut_Power
NumericFilterSpectrumOut_Power (W)
0.01
9e-3
8e-3
7e-3
6e-3
5e-3
4e-3
3e-3
2e-3
1e-3
0
0
50000
100000
150000
200000
250000
300000
350000
400000
450000
500000
Frequency (Hz)
NumericFilterSpectrumOut_Power

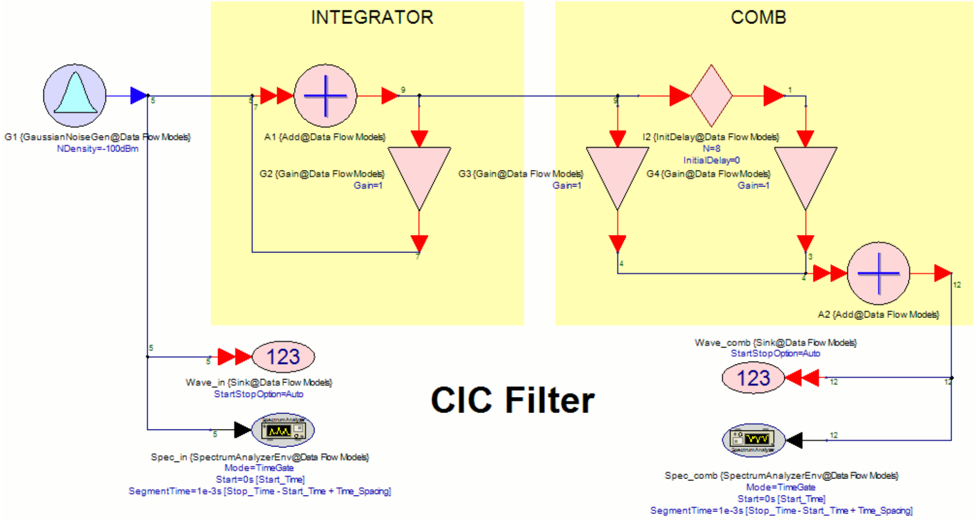
6. Change *System Sample Rate* to 4 MHz in *DF2* and run *DF2* again.
7. The 240 KHz sinusoid is filtered out by the envelope filter LPF Butterworth *F1* because it is re-designed based on the new sampling rate 4 MHz and the -3 db passband edge remains at 150 KHz.
8. As explained in the tutorial example Frequency Response vs Sample Rate, BiquadCascade *F2* is a numeric digital filter, so the passband edge 3π/10 in Z domain now maps to 600 KHz with respect to the new sampling rate 4 MHz. As a result, all three sinusoid waveforms are passed through.


DF2_SpectrumIn_Power (Workspace: FilterAndSampleRate)
SpectrumIn_Power
SpectrumIn_Power (W)
0.01
9e-3
8e-3
7e-3
6e-3
5e-3
4e-3
3e-3
2e-3
1e-3
0
0
199949.962
399899.925
599849.887
799799.85
999749.812
1.2e+6
1.4e+6
1.6e+6
1.8e+6
1.999e+6
Frequency (Hz)
SpectrumIn_Power
DF2_TimedFilterSpectrumOut_Power (Workspace: FilterAndSampleRate)
TimedFilterSpectrumOut_Power
TimedFilterSpectrumOut_Power (W)
Frequency (Hz)
TimedFilterSpectrumOut_Power
DF2_NumericFilterSpectrumOut_Power (Workspace: FilterAndSampleRate)
NumericFilterSpectrumOut_Power
NumericFilterSpectrumOut_Power (W)
Frequency (Hz)
NumericFilterSpectrumOut_Power

## Deadlock

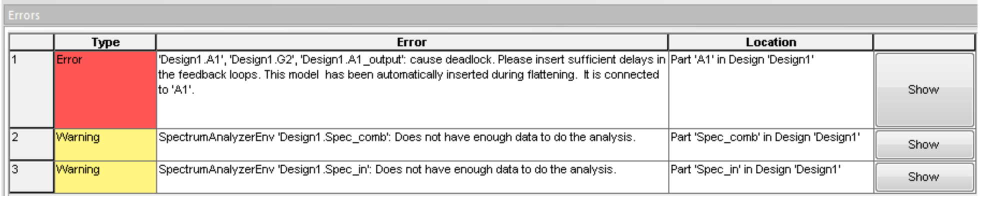
This example illustrates a common scenario that causes deadlocks and shows you how to resolve deadlocks.

1. Open <SystemVue installation directory>\Examples\Tutorials\Algorithm_Design\DataFlow\Debugging.wsv.
2. Open schematic *Design1* in folder *1. Deadlock*. This schematic is taken from the example workspace <SystemVue installation directory>\Examples\Model Building\CIC_filter.wsv. It represents a design of cascaded integrator-comb (CIC) filter.


INTEGRATOR
COMB
G1 {GaussianNoiseGen@Data Flow Models}
NDensity=-100dBm
A1 {Add@Data Flow Models}
G2 {Gain@Data FlowModels}
Gain=1
G3 {Gain@Data Flow Models}
Gain=1
I2 {InitDelay@Data Flow Models}
N=8
InitialDelay=0
G4 {Gain@Data FlowModels}
Gain=-1
A2 {Add@Data Flow Models}
123
Wave_in {Sink@Data Flow Models}
StartStopOption=Auto
Spec_in {SpectrumAnalyzerEnv@Data Flow Models}
Mode=TimeGate
Start=0s [Start_Time]
SegmentTime=1e-3s [Stop_Time - Start_Time + Time_Spacing]
CIC Filter
Wave_comb {Sink@Data Flow Models}
StartStopOption=Auto
123
Spec_comb {SpectrumAnalyzerEnv@Data Flow Models}
Mode=TimeGate
Start=0s [Start_Time]
SegmentTime=1e-3s [Stop_Time - Start_Time + Time_Spacing]
5
7
9
1
3
4
12

3. Run *DF1* data flow analysis.

4. SystemVue reports "Error: 'Design1_*A1', 'Design1*_G2': cause deadlock. Please insert sufficient delays in the feedback loops." When there is a deadlock in a system, SystemVue will report a list of components that cause the deadlock.



Errors

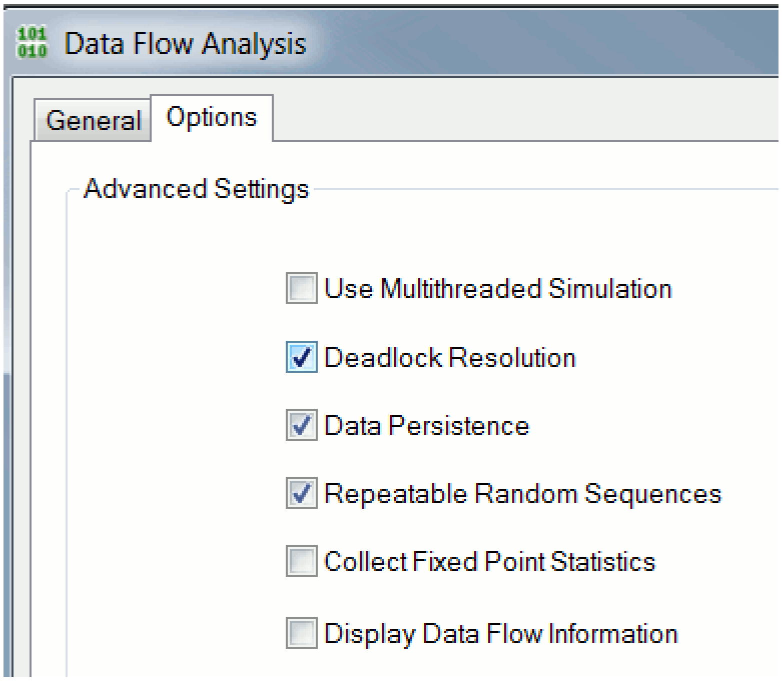
| | Type | Error | Location | |
|---|---|---|---|---|
| 1 | Error | 'Design1.A1', 'Design1.G2', 'Design1.A1_output': cause deadlock. Please insert sufficient delays in the feedback loops. This model has been automatically inserted during flattening. It is connected to 'A1'. | Part 'A1' in Design 'Design1' | Show |
| 2 | Warning | SpectrumAnalyzerEnv 'Design1.Spec_comb': Does not have enough data to do the analysis. | Part 'Spec_comb' in Design 'Design1' | Show |
| 3 | Warning | SpectrumAnalyzerEnv 'Design1.Spec_in': Does not have enough data to do the analysis. | Part 'Spec_in' in Design 'Design1' | Show |

5. Can you solve it?

6. By inspecting schematic *Design1* based on the error message, you can see that there is a feedback loop from the output of Add *A1*, which goes through Gain *G2*, and loops back to the input of *A1*. Based on data driven execution, to execute (fire) *A1*, it needs one sample from each of its input ports. To get the first sample from the lower input connection (from the loop), *G2* must fire once. To fire *G2*, it needs one sample from its input. However, the first input sample to *G2* cannot be generated unless *A1* is fired once. But *A1* cannot be fired due to deadlock.

7. To resolve a deadlock, sufficient number of delays need to be inserted in the loop to provide initial samples. The initial samples are buffered before a system stars execution, and therefore, allow one of the deadlocked blocks to start firing. See Deadlock and Deadlock Resolution for details.

8. Before resolving the deadlock manually, let's try the *Deadlock Resolution* option in the options tab of the data flow analysis. Double click *DF1* data flow analysis and select the *Options* tab. Check *Deadlock Resolution* checkbox.



Advanced Settings

☐ Use Multithreaded Simulation

☑ Deadlock Resolution

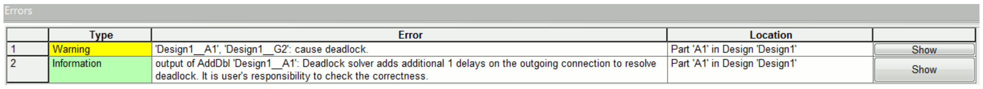
☑ Data Persistence

☑ Repeatable Random Sequences

☐ Collect Fixed Point Statistics

☐ Display Data Flow Information

9. Run *DF1* data flow analysis again.
10. This time, the simulation should run successfully with automatic deadlock resolution. SystemVue shows the following messages to inform users where the delays are inserted and ask users to verify the correctness of the system after deadlock resolution. Based on the message, SystemVue inserts one delay after the Add model *A1*. This initial delay allows the Gain model *G2* to run for the first time without firing *A1*, and therefore, breaks the deadlock.

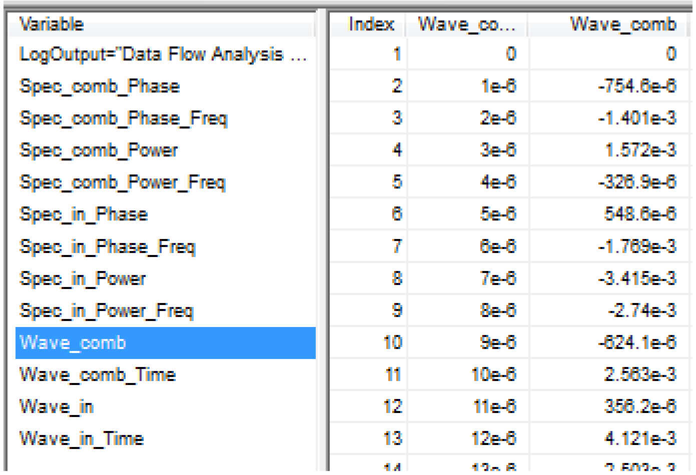


Errors

| | Type | Error | Location | |
|---|---|---|---|---|
| 1 | Warning | 'Design1__A1', 'Design1__G2': cause deadlock. | Part 'A1' in Design 'Design1' | Show |
| 2 | Information | output of AddDbl 'Design1__A1': Deadlock solver adds additional 1 delays on the outgoing connection to resolve deadlock. It is user's responsibility to check the correctness. | Part 'A1' in Design 'Design1' | Show |

**CAUTION** The system may behave differently depending on where SystemVue inserts delays to resolve deadlocks. It is user's responsibility to check the correctness of the deadlock resolution.

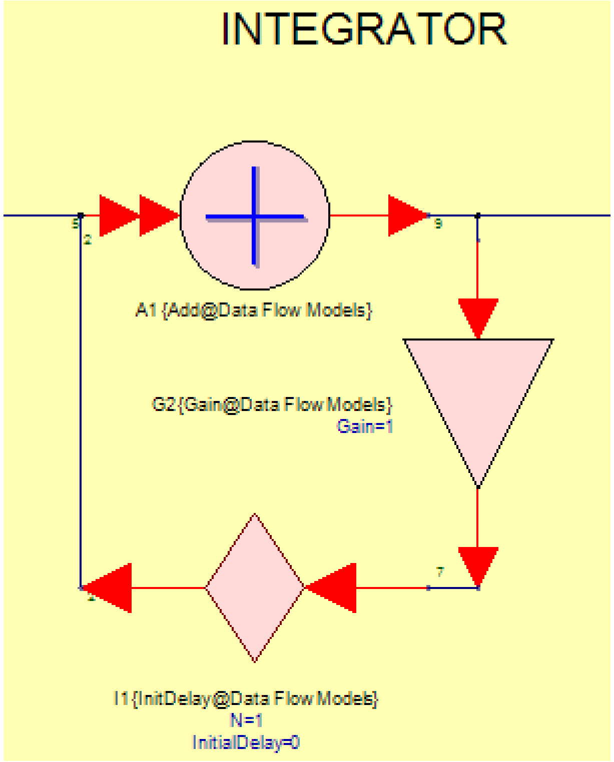
11. Take a look of the variable *Wave_comb* in *DF1_Data* dataset. Notice that the first sample is 0. This is because a one-sample Delay is automatically inserted at the output of the Add model *A1*, which introduces one sample delay at the *Wave_comb* Sink.



| Variable |
|---|
| LogOutput="Data Flow Analysis ... |
| Spec_comb_Phase |
| Spec_comb_Phase_Freq |
| Spec_comb_Power |
| Spec_comb_Power_Freq |
| Spec_in_Phase |
| Spec_in_Phase_Freq |
| Spec_in_Power |
| Spec_in_Power_Freq |
| Wave_comb |
| Wave_comb_Time |
| Wave_in |
| Wave_in_Time |

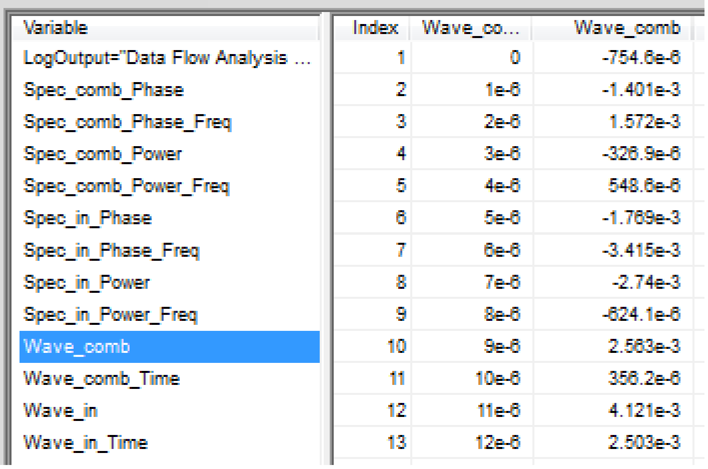
| Index | Wave_co... | Wave_comb |
|---|---|---|
| 1 | 0 | 0 |
| 2 | 1e-6 | -754.6e-6 |
| 3 | 2e-6 | -1.401e-3 |
| 4 | 3e-6 | 1.572e-3 |
| 5 | 4e-6 | -326.9e-6 |
| 6 | 5e-6 | 548.6e-6 |
| 7 | 6e-6 | -1.769e-3 |
| 8 | 7e-6 | -3.415e-3 |
| 9 | 8e-6 | -2.74e-3 |
| 10 | 9e-6 | -624.1e-6 |
| 11 | 10e-6 | 2.563e-3 |
| 12 | 11e-6 | 356.2e-6 |
| 13 | 12e-6 | 4.121e-3 |
| 14 | 13e-6 | 2.503e-3 |

12. Now, let's try to manually resolve the deadlock. Since the feedback loop needs one initial sample (Delay) to make *A1* fireable, we can resolve the deadlock by inserting a Delay model between the output of *G2* and the input of *A1*. Place down a Delay part on schematic. Flip the part direction using the *Mirror* function in the schematic toolbar. Set parameter *N* = 1, which makes the Delay model a one-sample delay. Insert the Delay at the suggested location. See the following screenshot as reference.


INTEGRATOR
A1 {Add@Data Flow Models}
G2{Gain@Data Flow Models}
Gain=1
I1{InitDelay@Data Flow Models}
N=1
InitialDelay=0

13. Uncheck *Deadlock Resolution* in *DF1* data flow analysis options tab. Run *DF1* data flow analysis again.

14. The simulation should run successfully with manual deadlock resolution. Take a look of the variable *Wave_comb* in *DF1_Data* dataset. Notice that the first sample is -754.6e-6 instead of 0. This is because we insert the delay at the input of *A1*, which does not cause extra delay to the rest of the system after *A1*. Again, the location of delay insertion affects system's behavior. It is user's responsibility to make sure the system behaves as design.



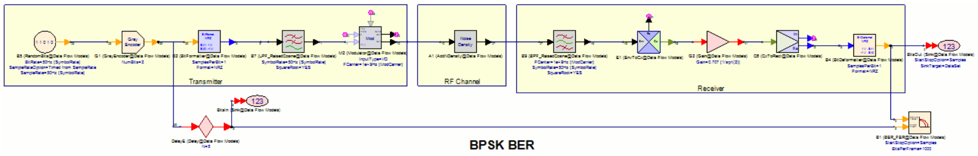
| Variable |
|---|
| LogOutput="Data Flow Analysis ... |
| Spec_comb_Phase |
| Spec_comb_Phase_Freq |
| Spec_comb_Power |
| Spec_comb_Power_Freq |
| Spec_in_Phase |
| Spec_in_Phase_Freq |
| Spec_in_Power |
| Spec_in_Power_Freq |
| Wave_comb |
| Wave_comb_Time |
| Wave_in |
| Wave_in_Time |

| Index | Wave_co... | Wave_comb |
|---|---|---|
| 1 | 0 | -754.6e-6 |
| 2 | 1e-6 | -1.401e-3 |
| 3 | 2e-6 | 1.572e-3 |
| 4 | 3e-6 | -326.9e-6 |
| 5 | 4e-6 | 548.6e-6 |
| 6 | 5e-6 | -1.769e-3 |
| 7 | 6e-6 | -3.415e-3 |
| 8 | 7e-6 | -2.74e-3 |
| 9 | 8e-6 | -624.1e-6 |
| 10 | 9e-6 | 2.563e-3 |
| 11 | 10e-6 | 356.2e-6 |
| 12 | 11e-6 | 4.121e-3 |
| 13 | 12e-6 | 2.503e-3 |

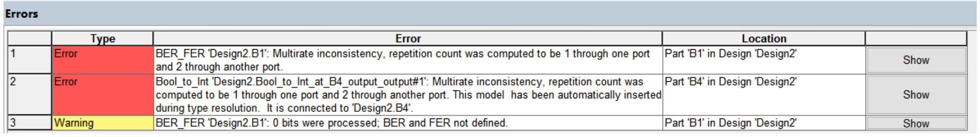
## Sample Rate Inconsistency

This example shows a common scenario that causes sample rate inconsistency and shows you how to resolve it.

1. Open <SystemVue installation directory>\Examples\Tutorials\Algorithm_Design\DataFlow\Debugging.wsv.
2. Open schematic *Design2* in folder *2. Sample Rate Inconsistency*. This schematic is taken from the example workspace <SystemVue installation directory>\Examples\Comms\BER\BPSK_BER.wsv. It contains a design to measure BPSK BER.



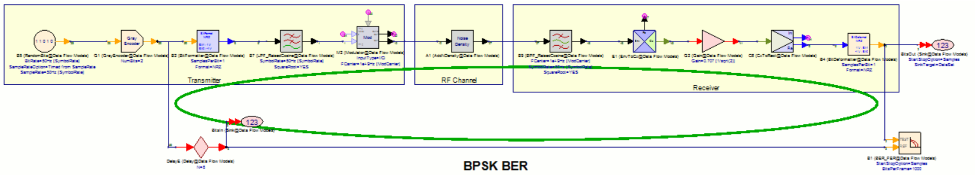
3. Run *DF2* data flow analysis.
4. SystemVue reports an error message "Error: 'Design2_*B1', 'Design2* _Bool_to_Int_at_B4_output_output#1': The scheduler has detected a sample rate inconsistency at the listed components. ...".



Errors

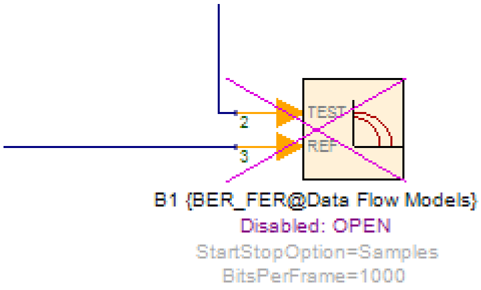
| | Type | Error | Location | |
|---|---|---|---|---|
| 1 | Error | BER_FER 'Design2.B1': Multirate inconsistency, repetition count was computed to be 1 through one port and 2 through another port. | Part 'B1' in Design 'Design2' | Show |
| 2 | Error | Bool_to_Int 'Design2.Bool_to_Int_at_B4_output_output#1': Multirate inconsistency, repetition count was computed to be 1 through one port and 2 through another port. This model has been automatically inserted during type resolution. It is connected to 'Design2.B4'. | Part 'B4' in Design 'Design2' | Show |
| 3 | Warning | BER_FER 'Design2.B1': 0 bits were processed; BER and FER not defined. | Part 'B1' in Design 'Design2' | Show |

5. Before running a simulation, SystemVue will make sure the system is sample rate consistent. If not, SystemVue reports sample rate inconsistency error and identify a component or a list of components that are located in a graph region that causes the sample rate inconsistency. In general, sample rate inconsistency happens due to inconsistent multirate properties in an undirected cycle. The following picture shows the undirected cycle that involves *B1* as reported in the error message.



6. Identify the inconsistent multirate properties is a key step to solving the problem. Data flow information table reports data flow rate information but such information is unavailable if there is an error during simulation setup. A very useful approach to get the data flow rate information is to break the undirected cycle, or in general, break the whole system into a smaller, analyzable subsystem, and then analyze the multirate properties.

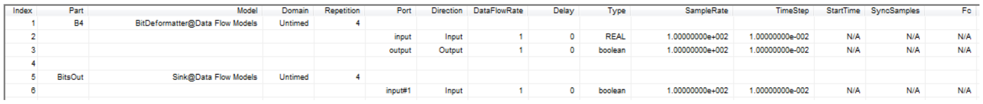
7. Since the BER_FER model *B1* is where the upper and lower paths combine, let's disable it for a moment and see if we can get the multirate information without it. Disable *B1*, as shown in the following screenshot.



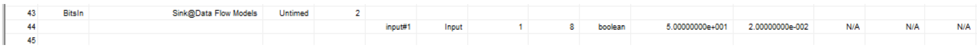
8. Run *DF2* data flow analysis.

9. Now the simulation should run successfully and that allows us to investigate the data flow information table.

10. Inspect the *DF2_Data_DataFlowInfo* table, especially focus on *BitsIn* sink and *BitsOut* sink since they provide data flow information at both input connections of the BER_FER model *B1*. As you can see, the repetition counts (column repetition) for *BitsIn* and *BitsOut* are 2 and 4 respectively, and the sampling rate (column SampleRate) for *BitsIn* (lower path) and *BitsOut* (upper path) are 50 Hz and 100 Hz respectively. However, because the BER_FER model requires the same sampling rate at both of its inputs, the sampling rate conversions across the upper path and across the lower path do not match.

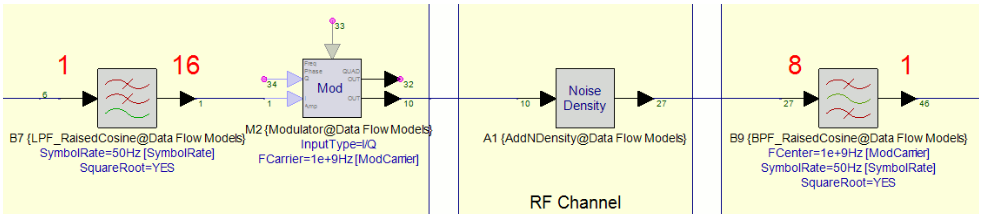


| Index | Part | Model | Domain | Repetition | Port | Direction | DataFlowRate | Delay | Type | SampleRate | TimeStep | StartTime | SyncSamples | Fc |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | B4 | BitDeformatter@Data Flow Models | Untimed | 4 | | | | | | | | | | |
| 2 | | | | | input | Input | 1 | 0 | REAL | 1.00000000e+002 | 1.00000000e-002 | N/A | N/A | N/A |
| 3 | | | | | output | Output | 1 | 0 | boolean | 1.00000000e+002 | 1.00000000e-002 | N/A | N/A | N/A |
| 4 | | | | | | | | | | | | | | |
| 5 | BitsOut | Sink@Data Flow Models | Untimed | 4 | | | | | | | | | | |
| 6 | | | | | input#1 | Input | 1 | 0 | boolean | 1.00000000e+002 | 1.00000000e-002 | N/A | N/A | N/A |



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 43 | BitsIn | Sink@Data Flow Models | Untimed | 2 |  |  |  |  |  |  |  |  |  |  |
| 44 |  |  |  |  | input#1 | Input | 1 | 8 | boolean | 5.00000000e+001 | 2.00000000e-002 | N/A | N/A | N/A |
| 45 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

11. Next, it is useful to identify and annotate the multirate components on both paths. From *DF2_Data_DataFlowInfo* table, the LPF RaisedCosine *B7* is a 1-to-16 interpolation filter and the BPF RaisedCosine *B9* is an 8-to-1 decimation filter. Now we know where is the problem. The sampling rate remains the same in the lower path but the sampling rate becomes twice effectively in the upper path.



B7 {LPF_RaisedCosine@Data Flow Models}
SymbolRate=50Hz [SymbolRate]
SquareRoot=YES

M2 {Modulator@Data Flow Models}
InputType=I/Q
FCarrier=1e+9Hz [ModCarrier]

A1 {AddNDensity@Data Flow Models}

B9 {BPF_RaisedCosine@Data Flow Models}
FCenter=1e+9Hz [ModCarrier]
SymbolRate=50Hz [SymbolRate]
SquareRoot=YES

12. Can you solve the sample rate inconsistency?
13. Changing the *Interpolation* parameter of *B7* to 8 or changing the *Decimation* parameter of *B9* to 16 can solve the problem. Change one of them as suggested.
14. Enable BER_FER *B1* and run *DF2* data flow analysis again. It should work now.

## C++ Model Development

In this tutorial, you will learn how to create and use custom C++ model libraries. Before starting this tutorial, we recommend that you read Introduction to C++ Model Development.

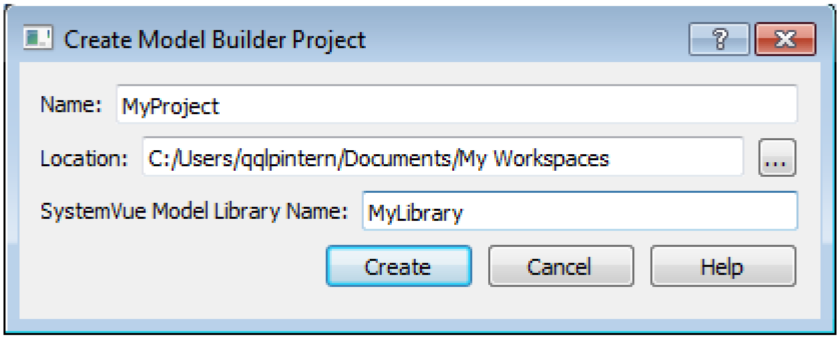
**NOTE** **The Visual Studio version in the screenshot and the tutorial may differ than the version that is supported in your SystemVue release.**

### Example 1: Building Your First Custom C++ Model Library

In this example, you will use SystemVue *Action > Create Model Builder Project...* command to quickly create and build your first custom C++ model library.

#### Create a Model Builder Project using CMake

1. Select Action > Create Model Builder Project
2. Set the Project parameters as shown in the figure below:

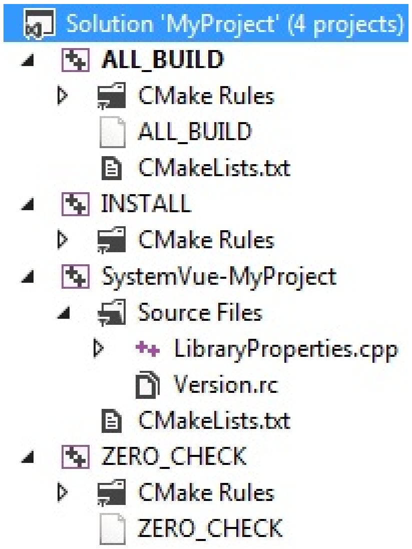


Name: MyProject

Location: C:/Users/qqlpintern/Documents/My Workspaces

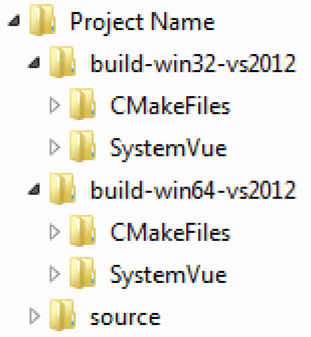
SystemVue Model Library Name: MyLibrary

3. Click Create.
   A new instance of Visual Studio will be invoked with the generated Model Builder Solution. The generated Solution tree is shown in the figure below:



```
Solution 'MyProject' (4 projects)
  ALL_BUILD
    CMake Rules
    ALL_BUILD
    CMakeLists.txt
  INSTALL
    CMake Rules
  SystemVue-MyProject
    Source Files
      LibraryProperties.cpp
      Version.rc
    CMakeLists.txt
  ZERO_CHECK
    CMake Rules
    ZERO_CHECK
```

The folder structure of the generated Model Builder Project is also shown below. Note that the Win32 bit and Win64 bit solutions are created inside the build folders build-win32-vs2012 and build-win64-vs2012 respectively while the source code and the associated CMakeLists.txt files are inside the source folder.



```
Project Name
  build-win32-vs2012
    CMakeFiles
    SystemVue
  build-win64-vs2012
    CMakeFiles
    SystemVue
  source
```

### Create the MyGain C++ Model

1. On the Solution Explorer Window, open CMakeLists.txt file in SystemVue-MyProject project ( This file is located at C:\Users\<user-name>\Documents\My Workspaces\MyProject\source\systemvue\ ). Update this file by uncommenting the following lines and changing the name of the model from MyCustomModel to MyGain:

```
#SVU_CREATE_MODEL ( MyCustomModel )
```

and

```
#    MyCustomModel.h
#    MyCustomModel.cpp
```

This will allow the SVU_CREATE_MODEL macro to add a header and a source file templates for a SystemVue Model. The updated CMakeLists.txt becomes as follows:

```
cmake_minimum_required(VERSION 2.8.8)

# SystemVueInstallDirectory defined in ..
/CMakeLists.txt
include ( ${SystemVueInstallDirectory}/ModelBuilder
/CMake/ModelBuilder.cmake )

# SystemVue code generation maintains this
variable, do not add files to this variable
set ( SVU_GENERATED_FILES
    LibraryProperties.cpp
)

# Change content below as necessary. For example,
add more files, etc.

# Add a new model easily from a template by using
the SVU_CREATE_MODEL macro as commented
# out below.  See the documentation for more
details.
# You will need to add the generated source as
shown in the commented out files below inside
# the SVU_ADD_MODELBUILDER_LIBRARY area.
SVU_CREATE_MODEL ( MyGain )

# Uncomment the lines below to customize the
version information for your DLL.
```

```
# On the PC the info will show either by mousing-
over your DLL or right
# clicking on the DLL and inspecting Properties ->
Details.
#
# The version numbers below, can be up to 4
integers separated by periods.  For
# example, "1.20.22.3".   For more details, refer
to the SystemVue documentation.
SVU_CREATE_VERSION_FILE(
    VERSION_FILE
    # FILE_VERSION 1.0
    # COMPANY_NAME "My Company Name"
    # COPYRIGHT "(c) My Company Name, "
    # PRODUCT_NAME "My Product Name"
    # PRODUCT_VERSION 1.0
    TARGET_TYPE "Model Library"
)

SVU_ADD_MODELBUILDER_LIBRARY(
    SystemVue-MyProject
    ${SVU_GENERATED_FILES}
    ${VERSION_FILE}
    MyGain.h
    MyGain.cpp
)

SVU_INSTALL_MODELBUILDER_LIBRARY( SystemVue-
MyProject )
```

WARNING

Do not add the newly created source files (**MyGain.h** and **MyGain.cpp**) by inserting the files into the solution source tree. Since the generated Solution is created by CMake, adding new source files should be done by updating the appropriate CMakeLists.txt file in the source folder.

NOTE

To add more than one model, add a new SVU_CREATE_MODEL call for each model to be added if there is no existing source code for that model. The macro will generate the header and source file templates for the SystemVue model. In addition, add the corresponding header and source files to the arguments of SVU_ADD_MODELBUILDER_LIBRARY call. Below is an example of adding two SystemVue Models: MyModel1 and MyModel2:

```
SVU_CREATE_MODEL ( MyModel1 )
SVU_CREATE_MODEL ( MyModel2 )

SVU_ADD_MODELBUILDER_LIBRARY(
    SystemVue-MyProject
    ${SVU_GENERATED_FILES}
    MyModel1.h
    MyModel1.cpp
    MyModel2.h
    MyModel2.cpp
)
```

2. Save your changes, right-click the *Install* project in the *Solution Explorer* to rebuild SystemVue-MyProject project. Visual Studio and CMake will generate several warning messages to inform the user that the project content is changed and is going to be reloaded. Click OK to all the warning messages that appear. After successful reloading of SystemVue-MyProject project, new MyGain.h and MyGain.cpp files are added to the project as shown in the figure below. These generated header and source files provide a quick starting point to build a SystemVue Model, for purposes of this example leave these unchanged. By default, this macro uses a gain model as the starting template.

In this example, these files are created at C:\Users\<user-name>\Documents\My Workspaces\MyProject\source\systemvue\ folder, where <user-name> is the user name on the host machine.
The content of MyGain.h is as follows:

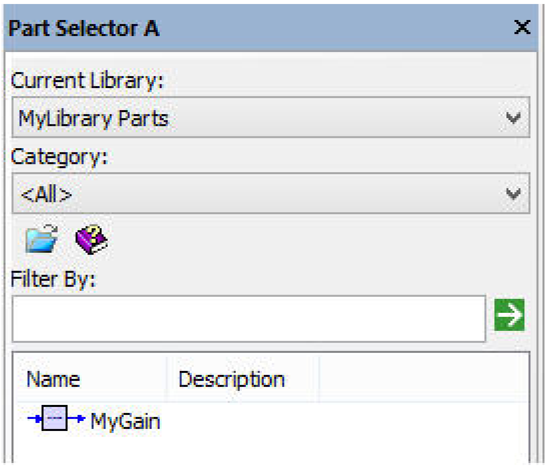
```
#pragma once
#include "SystemVue/ModelBuilder.h"

class MyGain : public SystemVueModelBuilder::DFModel
{
public:
    // This Macro is required for all classes derived
from DFModel
    DECLARE_MODEL_INTERFACE( MyGain );

    // Constructor to initialize parameters
    MyGain();

    //-------- Function Overloads --------
    virtual bool    Run();

    // Ports
    SystemVueModelBuilder::CircularBuffer< double >
input, output;

    // Parameter
    double Gain;

};
```

The content of MyGain.cpp is as follows:

```
#include "MyGain.h"

#ifndef SV_CODE_GEN
DEFINE_MODEL_INTERFACE ( MyGain )
{
    ADD_MODEL_INPUT( input );
    ADD_MODEL_OUTPUT( output );
    ADD_MODEL_PARAMETER( Gain );
    return true;
}
#endif

MyGain::MyGain()
{
    Gain = 0;
}

//------------------------------------------------------
-----------------------------
//    Run
//        Here we do the math
//------------------------------------------------------
-----------------------------
bool MyGain::Run()
{
    output[0] = Gain * input[0];
    return true;
}
```

### Using the MyGain Model

1. Start SystemVue using a Blank template.
2. Click Tools > Library Manager....
3. In the Library Manager dialog box, select Add From File.
4. Browse to your Project location and then output-vs2012\Debug-SystemVue-Win64 (or output-vs2012\Debug-SystemVue-Win32) sub directory, select MyProject.dll.
5. Click Open. Scroll down to see that the library has been added and is shown in the list.
6. Click Close.
7. Under Part Selector, in Current Library choose MyLibrary Parts, this will show the MyGain that you have created.



Part Selector A

Current Library: MyLibrary Parts

Category: <All>

Filter By:

| Name | Description |
|---|---|
| MyGain | |

8. Place an instance of the MyGain and create a simple design using it.

   **CAUTION** SystemVue uses the DLL library name to name the Part, Model and Enum libraries. The model builder DLLs that you load must have unique names. Read Defining the Model Library Properties to override this default behavior.

### Debugging the MyGain Model

1. In Visual Studio, select the Debug > Attach to Process.
2. In the *Attach to Process* dialog:
   a. Set the *Attach to:* field to *Native Code*

      **CAUTION** Any other settings will not allow you to debug your code. If you are using *Express Edition* and do not see *Attach to Process...* menu command, you must set *Tools -> Settings -> Expert Settings*.

   b. In the *Available Process*, select the SystemVue process that is running.
3. In Visual Studio, double-click on the MyGain.cpp file and set a breakpoint in the `MyGain::Run()` method by clicking on the gray bar next the `output[0] = Gain * input[0];` line.
4. Run a simulation using the simple design that you created about - Visual Studio will stop at the line that you set the breakpoint on.
5. Explore the debugger, including the *Autos* and *Locals* tabs.
6. When you are ready to continue, click on the breakpoint (red dot) to remove the breakpoint. Press *F5* to continue runing SystemVue.
7. Detach the debugger by selecting the *Debug > Detach All*.

   **CAUTION** Do not select *Debug > Stop Debugging* as it will exit the SystemVue process.

## Example 2: Developing Your First Custom C++ Model

In this section, you will add an Adder to the model builder library from Example 1. The directions below assume that SystemVue is still running from the previous example.

### Create the MyAdder C+/+ Model

1. On the Solution Explorer Window, open CMakeLists.txt file in SystemVue-MyProject project.
2. In the CMakeLists.txt file:
    a. Update the `SVU_CREATE_MODEL` macro to create a new model named MyAdder.
    b. Add MyAdder.h and MyAdder.cpp to the list of `SVU_ADD_MODELBUILDER_LIBRARY`.
3. Right-click and build the Install project.
4. If all is set up properly, SystemVue will automatically reload MyProject.dll. In the SystemVue *Part Selector*, verify that the DLL has been reloaded and you now see both the MyGain and MyAdder in the list.
5. Modify the generated files MyAdder.has part of developing a SystemVue model for an Adder as shown in the code below:

```
#pragma once
#include "SystemVue/ModelBuilder.h"

class MyAdder : public SystemVueModelBuilder::
DFModel
{
public:
    // This Macro is required for all classes
derived from DFModel
    DECLARE_MODEL_INTERFACE( MyAdder );

    // Constructor to initialize parameters
    MyAdder();

    //-------- Function Overloads --------
    virtual bool    Initialize();
    virtual bool    Run();

    // Ports
    SystemVueModelBuilder::CircularBuffer< double >
In1, In2, Out;

    // Parameter
    double Gain;

};
```

Note the following:

NOTE

- You must include **ModelBuilder.h**.
- You must add macro **DECLARE_MODEL_INTERFACE (<ClassName>);** with public access.
- The data members for parameters and input/outputs must have public access.

6. Modify the MyAdder.cppfile as part of developing a SystemVue model for an Adder as shown in the code below:

```
#include "MyAdder.h"

#ifndef SV_CODE_GEN
DEFINE_MODEL_INTERFACE ( MyAdder )
{
    ADD_MODEL_INPUT( In1 );
    ADD_MODEL_INPUT( In2 );
    ADD_MODEL_OUTPUT( Out );
    ADD_MODEL_PARAMETER( Gain );
    return true;
}
#endif

MyAdder::MyAdder()
{
    Gain = 0;
}


bool MyAdder::Initialize()
{
    if (Gain ==0)
    {
        POST_ERROR("The value of Gain cannot be ==
0");
        return false;
    }
    return true;
}

//----------------------------------------------------
---------------------------------
//    Run
//        Here we do the math
//----------------------------------------------------
---------------------------------
bool MyAdder::Run()
{
    Out[0] = Gain * (In1[0] + In2[0]);
    return true;
}
```

Also, note the following:

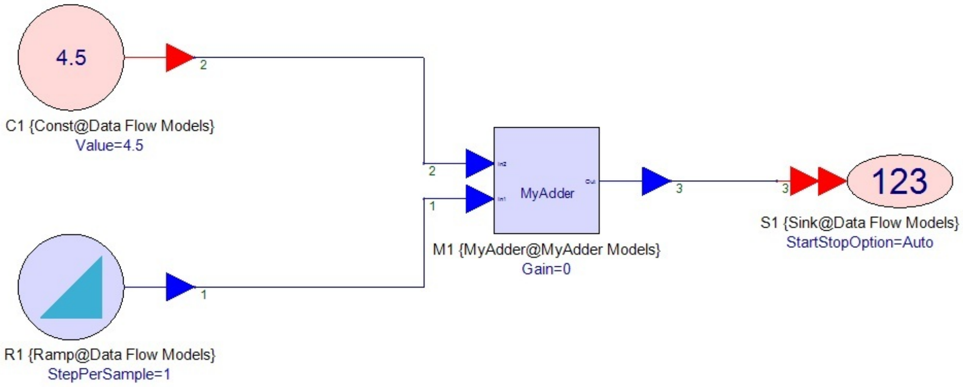
NOTE

- Use the **ADD_MODEL_INPUT(<data member>);** macro to add a data member as input.
- Use the **ADD_MODEL_OUTPUT(<data member>);** macro to add a data member as output.
- Use the **ADD_MODEL_PARAM(<data member>);** macro to add a data member as a parameter.
- Inputs, outputs and parameters can only be added inside **DEFINE_MODEL_INTERFACE(<class name>)** macro.
- Use **POST_ERROR** macro to post an error.

7. Save your updates and build the Install project again.

### Using the MyAdder Model in SystemVue

1. Add a *Design* to the workspace, place an instance of the MyAdder, and create the design:


4.5
2
C1 {Const@Data Flow Models}
Value=4.5
2
1
MyAdder
3
3
123
S1 {Sink@Data Flow Models}
StartStopOption=Auto
M1 {MyAdder@MyAdder Models}
Gain=0
1
R1 {Ramp@Data Flow Models}
StepPerSample=1

2. Simulate the design. The design will give an expected error about the value of Gain == 0. This is the error you have posted in our Initialize() method in Adder.cpp file above.
3. Change the value of Gain to a non-zero value and successfully simulate the design.

## Example 3: Developing a Gardner Timing Recovery C++ Model

### Introduction

In this example, you will develop a C++ model that implements the *Gardner Timing Recovery* algorithm, for details on this algorithm see the IEEE paper referenced below.

In Gardner's paper, the detector algorithm was described by the following equation:

$$u_t(r) = y_I(r-1/2)[y_I(r) - y_I(r-1)] + y_Q(r-1/2)[y_Q(r) - y_Q(r-1)]$$

In the equation above, the detector output

$$u_t(r)$$

is a function of the inphase (I) and quadrature (Q) components of the input signal

$$y$$

. The index of

$$y$$

is incremented in 1/2 sample steps. Instead of using 1/2 sample steps, you will run the signal at twice the rate and decimate it by 2. Doing so, the detector algorithm becomes:

$$u_t(r) = y_I(r-1)[y_I(r) - y_I(r-2)] + y_Q(r-1)[y_Q(r) - y_Q(r-2)]$$

You can further simplify this equation, by observing that the calculation for I and Q above are the same:

$$x_t(r) = y(r-1)[y(r) - y(r-2)]$$

To calculate

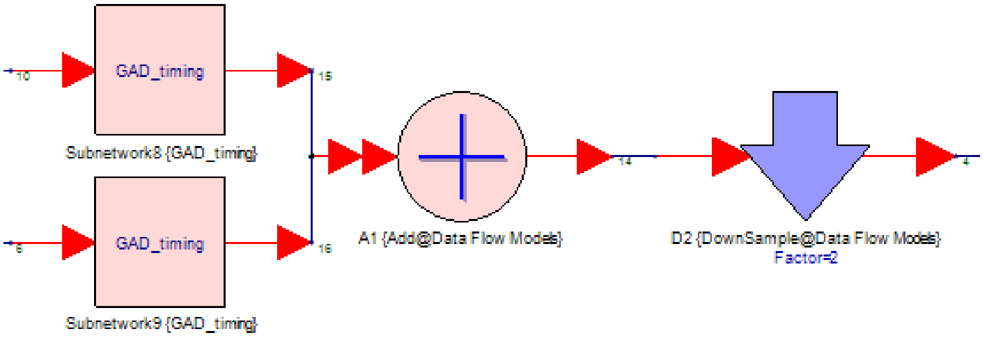
$$u_t(r)$$

, the *Gardner Timing Recovery* algorithm now becomes:

$$u_t(r) = x_I(r) + x_Q(r)$$

## Steps

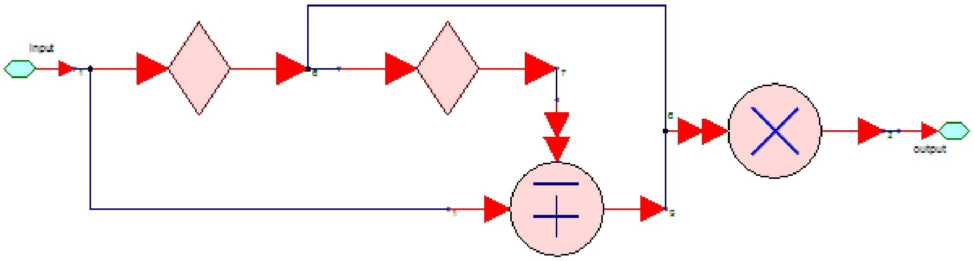
1. This algorithm is used in the QPSK Transceiver Design tutorial example located in: "Tutorials\QPSK_Transceiver_Design\FloatingPoint\QPSK_FloatingPoint.wsv", open this workspace.
2. Open the top-level transceiver design: *QPSK_Fx_SIM*
3. Open the *demod* subnetwork.
4. Open the *qpsk_demod* subnetwork. Here you can see the Gardner algorithm implemented:



5. The *GAD_timing*subnetwork implements the

$$x_t(r)$$

function:



6. Double click on one of the GAD_timing parts and switch the model to *MATLAB_Script@Data Flow Models*. This is the MATLAB Script implementation of the subnetwork:

```
% Define and initialize the tap delay line
persistent y;
```

```
if ( isempty(y) )
    y=zeros(3,1)
end

% Add new input to the tap delay line
y = [input; y(1:end-1) ]

% Detector algorithm
output = y(2) * ( y(1) - y(3) )
```

7. You will be porting this algorithm to C++. Open the *MyProject* Visual Studio solution you created in Example 1.

8. Add a new model named *GAD_timing* using the CMake macro `SVU_CREATE_MODEL`.

9. Reimplement the MATLAB Script implementation above in your new C++ Model, here are a couple of hints:

   - Define a new data member for the GAD_timing C++ class definition in the GAD_timing.h file, the data member will be defined should look like this:

     ```
     double y[3];
     ```

   - Define a new method to initialize the tap delay line elements to zero, this method should have the following signature:

     ```
     bool Initialize();
     ```

10. Right-click and build the Install project.

11. The MyProject.dll should autoreload. Verify that your model is in the part selector.

12. Add the new C++ version of this model to the GAD_timing part:

    a. Double click on one of the GAD_timing parts,

    b. In the *Part Properties* dialog, click on the *Manage Models...* button.

    c. In the *Manage Models* dialog, click on the *Add Model* button.

    d. From the drop-down, select *From Library...*

    e. Set the *Current Library* to *MyProject Models*.

    f. Select the *GAD_timing* model and click *OK*.

    g. In the *Part Properties* dialog, switch the GAD_timing@MyProject Models.

13. Run the simulation and verify that your model functions the same as the other two implementations.

### References

- "A BPSK/QPSK timing-error detector for sampled receivers", FM Gardner, IEEE Transactions on Communications, 1986.

## Example 4: Writing Fixed Point Models

A model having at least one SystemVueModelBuilder::FixedPointCircularBuffer, and /or SystemVueModelBuilder::FixedPointCircularBufferBus input/output is considered as a fixed point model. A fixed point model class must be derived from SystemVueModelBuilder::DFModel as well as an interface class SystemVueModelBuilder::DFFixedPointInterface both with public access. The model class must also override the virtual ERESULT SetOutputFixedPointParameters() method to set output FixedPointParameters based on the model parameters or the FixedPointParameters of inputs. The FixedPointParameters for inputs are set by the simulator before calling SetOutputFixedPointParameters based on the output FixedPointParameters of the previous model in the design.

### FixedPoint Inputs/Outputs

A model must use SystemVueModelBuilder::FixedPointCircularBuffer, and/or SystemVueModelBuilder::FixedPointCircularBufferBus to add a FixedPoint input or output, please see SystemVue CircularBuffer Data Types, and SystemVue FixedPoint Data Type for further details. A data member of type SystemVueModelBuilder::FixedPoint cannot be added as an input or an output.

### Overriding SetOutputFixedPointParameters

The SetParameters method of SystemVueModelBuilder::FixedPointCircularBuffer must be called for each single output and also for each output of a SystemVueModelBuilder::FixedPointCircularBufferBus bus. You may also read the parameters of inputs in this method which are already set by the simulator. The SetOutputFixedPointParameters is called several times during simulation until convergence is achieved. In the case of models whose output precision depends upon input precision, you may query that input has a valid FixedPointParameters or not using AreParametersValid method of the FixedPointCircularBuffer. An input may not have valid FixedPointParameters in first few iterations before convergence only when it is connected to a feedback loop. Even, if any of the input does not have valid FixedPointParameters you must set valid FixedPointParameters for all the outputs.

An example fixed point adder is shown below

```
#pragma once
#include "SystemVue/ModelBuilder.h"
#include "DFFixedPointInterface.h"

class AddFxp :
     public SystemVueModelBuilder::DFModel, public
SystemVueModelBuilder::DFFixedPointInterface
```

```
{
public:
    /// Output Parameters
    int WordLength;
    int IntegerWordLength;
    SystemVueModelBuilder::FixedPointEnums::Sign
IsSigned;
    SystemVueModelBuilder::FixedPointEnums::
OverflowMode Overflow;
    SystemVueModelBuilder::FixedPointEnums::
QuantizationMode Quantization;
    int SaturationBits;

    /// input bus
    SystemVueModelBuilder::FixedPointCircularBufferBus
dataIn;

    ///output
    SystemVueModelBuilder::FixedPointCircularBuffer
dataOut;

private:
    /// Accumulator for the sum
    /// SystemVueModelBuilder::FixedPointValue is
arbitray precision type. An object of
    /// FixedPointValue type may store a fixed-point
value of arbitrary precision
    /// and binary point location without losing
precision or magnitude (no quantization
    /// or overflow handling). This is suitbale for
accumulating the sum. The
    /// overflow/quantization handling will be
performed on dataOut[0] when we
    /// assign this accumulated sum to the dataOut[0]
    SystemVueModelBuilder::FixedPointValue
m_fxpAccumulator ;

public:
    // This Macro is required for all classes derived
from CDFModel
    DECLARE_MODEL_INTERFACE( AddFxp )

    //-------- Function Overloads --------
    bool                Run();                // Do the
math
    bool                Initialize();

    ERESULT SetOutputFixedPointParameters();
};
```

```
#include "AddFxp.h"
```

```
DEFINE_MODEL_INTERFACE ( AddFxp )
{
    SET_MODEL_NAME( "AddFxp" );
    SET_MODEL_CATEGORY( "Math Scalar" );
    SET_MODEL_SYMBOL( "SYM_AddFxp" );

    ADD_MODEL_INPUT( dataIn );
    ADD_MODEL_OUTPUT( dataOut );

    WordLength = 16; // default value
    SystemVueModelBuilder::DFParam cWL =
ADD_MODEL_PARAMETER(WordLength);

    IntegerWordLength = 2; // default value
    SystemVueModelBuilder::DFParam cIWL =
ADD_MODEL_PARAMETER(IntegerWordLength);

    // Adding built in enumerations
    ADD_MODEL_PARAMETER(IsSigned);
    ADD_MODEL_PARAMETER(Quantization);
    ADD_MODEL_PARAMETER(Overflow);

    SaturationBits = 0; // default value
    SystemVueModelBuilder::DFParam cSB =
ADD_MODEL_PARAMETER(SaturationBits);

    return true;
}


ERESULT AddFxp::SetOutputFixedPointParameters()
{
    dataOut.SetParameters(WordLength,IntegerWordLength,
IsSigned,Quantization,Overflow,SaturationBits);
    return NOERROR_;
}

bool AddFxp::Initialize()
{
    if(WordLength <=0)
        POST_ERROR("Word Length must be greater than 0."
);
    return true;
}

//------------------------------------------------------
-----------------------------
//    Go
//        Here we do the math
//------------------------------------------------------
-----------------------------
bool AddFxp::Run()
```

```
{
    m_fxpAccumulator = 0;

    // accumulate the sum for all inputs on the bus
without quantization/overflow
    // handling
    for(size_t szPort=0; szPort < dataIn.GetSize();
szPort++)
    {
        m_fxpAccumulator += dataIn[szPort][0];
    }

    // assign the accumulated sum to output, this will
cause quantization/overflow handling
    dataOut[0] = m_fxpAccumulator;

    return true;
}
```

### Example 5: Writing a Timed Data Flow Model

In this example, you will create a timed sine generator model that sets simulation sample rate based on the SampleRate parameter and generates timed sine wave based on Amplitude, Frequency, and Phase parameters.

Refer to the Writing Data Flow C++ Models section in the User's Guide for detailed documentation on the TimedDFModel class.

1. Open the MyProject Visual Studio solution you created in Example 1.
2. Add a new model named `SineGenerator` using the CMake macro `SVU_CREATE_MODEL`.
3. A timed model must use C++ classes that support time, by default `SVU_CREATE_MODEL` macro creates a template for an untimed model. In Visual Studio, edit `SineGenerator.h` makes the following changes:
    a. Add the necessary include statements for the timed C++ classes:

    ```
    #include "SystemVue/TimedDFModel.h"
    #include "SystemVue/TimedCircularBuffer.h"
    ```

    b. A timed model is derived from the `SystemVueModelBuilder::TimedDFModel` class, instead of `SystemVueModelBuilder::DFModel`. Replace:

    ```
    class SineGenerator : public
    SystemVueModelBuilder::DFModel
    ```

    with

```
class SineGenerator : public
SystemVueModelBuilder::TimedDFModel
```

.

c. A timed model must be derived from the `SystemVueModelBuilder::CircularBuffer` class, instead of `SystemVueModelBuilder::TimedCircularBuffer`. Replace:

```
SystemVueModelBuilder::CircularBuffer< double >
```

with

```
SystemVueModelBuilder::TimedCircularBuffer< dou
ble >
```

.

d. Remove the Gain parameter and input port definitions from the header.

e. Add the new parameters:

```
double Amplitude;
double Frequency;
double Phase;
double SampleRate;
```

f. Finally, you will need to add a new method declaration **`Setup`** which must be used by the model to set the *sample rate* that will be used in the data flow simulation:

```
virtual bool Setup();
```

4. Now, define the implementation in the **`SineGenerator.cpp`**:

a. Update the **`DEFINE_MODEL_INTERFACE`**, delete the unneeded parameter and port and add the definitions for the *SineGenerator* model:

```
ADD_MODEL_PARAM( Amplitude );

{
    SystemVueModelBuilder::DFParam param =
ADD_MODEL_PARAM( Frequency );
    param.SetUnit(SystemVueModelBuilder::
Units::FREQUENCY);
}

{
    SystemVueModelBuilder::DFParam param =
ADD_MODEL_PARAM( Phase );
    param.SetUnit(SystemVueModelBuilder::
Units::ANGLE);
}
{
    SystemVueModelBuilder::DFParam param  =
ADD_MODEL_PARAM( SampleRate );
    param.SetUnit(SystemVueModelBuilder::
Units::FREQUENCY);
}
```

b. For these parameters, define the default values in the constructor ( `SineGenerator::SineGenerator()`method):

```
Amplitude = 1;
Frequency = 5e3;
Phase = 0;
SampleRate = 1e6;
```

c. Add the definition for the `Setup`method to declare the sample rate (if set) to SystemVue. If not set, the model will use the sample rate computed by SystemVue:

```
bool SineGenerator::Setup()
{
    bool bStatus = true;
    if ( SampleRate > 0 )
    {
        // Use TimedCircularBuffer::
SetSampleRate method to set the output sample
rate
        output.SetSampleRate( SampleRate );
    }
    else
    {
        POST_ERROR( "SampleRate must be
greater than 0." );
        bStatus = false;
    }
    return bStatus;
}
```

d. Lastly, update the `Run`method, it will get the current timestamp from the output port:

```
bool SineGenerator::Run()
{
    bool bStatus = true;

    //Use TimedCircularBuffer::GetTime method
to get the time stamp of the output sample
    //In output.GetTime( 0, m_iFiringCount ),
0 means the 0th output sample of each firing
(run), and TimedDFModel::GetCount returns the
current firing count.
    output[0] = Amplitude * sin( 2* 3.1415 *
Frequency * output.GetTime( 0, GetCount() ) +
Phase );

    return bStatus;
}
```

5. Right-click and build the Install project.
6. The MyProject.dll should autoreload. Verify that your model is in the part selector.
7. Build a few test designs to verify that your new timed model is performing as expected.

### Example 6: Writing a Timed Data Flow Model that Overrides the Latency Calculation

The following TimedDownSampler shows an example that overrides. `TimedDFModel::CalculateLatency()` The input samples are downsampled by *Factor*. For each firing (run), only the *Phase* sample among *Factor* input samples is sent to the output. As a result to make the behavior causal, the time stamp of the first output sample should be delayed by *Phase* * input time step for causality.

```
//TimedDownSampler.h
#pragma once
#include "SystemVue/ModelBuilder.h"
#include "SystemVue/TimedDFModel.h"
#include "SystemVue/TImedCircularBuffer.h"

class TimedDownSampler : public SystemVueModelBuilder::
TimedDFModel
{
public:
    DECLARE_MODEL_INTERFACE( TimedDownSampler )
    virtual bool Run();
    virtual bool Setup();

        //Override default latency calculation
    ERESULT CalculateLatency();

    int Factor;
    int Phase;

    SystemVueModelBuilder::TimedCircularBuffer<double>
input;
    SystemVueModelBuilder::TimedCircularBuffer<double>
output;
};
```

```
//TimedDownSampler.cpp
#include "TimedDownSampler.h"

DEFINE_MODEL_INTERFACE( TimedDownSampler )
{
    SystemVueModelBuilder::DFParam paramFactor =
ADD_MODEL_PARAM( Factor );
    paramFactor.SetDefaultValue( "2" );
    SystemVueModelBuilder::DFParam paramPhase =
ADD_MODEL_PARAM( Phase );
    paramPhase.SetDefaultValue( "0" );
    ADD_MODEL_INPUT( input );
    ADD_MODEL_OUTPUT( output );
    return true;
}

bool TimedDownSampler::Setup()
{
        bool bStatus = true;
    if ( Factor < 1 )
        {
        POST_ERROR("Phase should be greater than 1.");
                bStatus = false;
        }
        //Set input data flow rate to Factor
        input.SetRate( (size_t)Factor );

    if( Phase >= Factor || Phase < 0 )
        {
        POST_ERROR("Phase should be greater than or
equal to 0 and less than Factor");
                bStatus = false;
        }

        return bStatus;
}
```

```
ERESULT TimedDownSampler::CalculateLatency()
{
        //For causality, set output start time to be
the input start time + Phase * input time step
    output.SetStartTime( input.GetStartTime ()+ input.
GetTimeStep() * Phase );
    return NOERROR_;
}

bool TimedDownSampler::Run()
{
    output[0] = input[ (size_t)Phase ];
    return true;
}
```

### Example 7: Writing a Timed Data Flow Model uses Envelope Signals

SystemVue provides EnvelopeSignal data type and corresponding SystemVueModelBuilder::EnvelopeCircularBuffer for writing models which require complex envelope signals.

The following EnvelopeToReal example shows how to convert an envelope signal (which can represent either a real signal or a complex envelope signal) to real signal.

```
//EnvelopeToReal.h
#pragma once
#include "SystemVue/ModelBuilder.h"
#include "SystemVue/TimedDFModel.h"
#include "SystemVue/EnvelopeSignal.h"

class EnvelopeToReal : public SystemVueModelBuilder::
TimedDFModel
{
    DECLARE_MODEL_INTERFACE( EnvelopeToReal )
    virtual bool Run();

    //Envelope signal
    SystemVueModelBuilder::EnvelopeCircularBuffer input;

    //Real signal
    SystemVueModelBuilder::CircularBuffer<double>
output;
};
```

```
//EnvelopeToReal.cpp
#include "EnvelopeToReal.h"

DEFINE_MODEL_INTERFACE( EnvelopeToReal )
```

```
{
    ADD_MODEL_INPUT( input );
    ADD_MODEL_OUTPUT( output );

    return true;
}

bool EnvelopeToReal::Run()
{
    //If input represents a real signal (based on
whether the characterization frequency is equal to 0)
    if ( input.GetCharacterizationFrequency() == 0 )
    {
        //Use EnvelopeSignal::real() to get the value
of the real signal
        output[0] = input[0].real();
    }
    //Otherwise, input represents a complex envelope
with associated characterization frequency
    else
    {
        //Use EnvelopeSignal::ConvertToReal to convert
the complex envelope to real signal
        output[0] = input[0].ConvertToReal( input.
GetCharacterizationFrequency(), input.GetTime(0,
GetCount()) );
    }
    return true;
}
```

### Example 8: Writing a Timed Data Flow Model that Overrides the Characterization Frequency Propagation

The following Modulator example up-converts baseband I-Q complex sample to complex envelope signal at *CarrierFrequency*, and use `TimedDFModel::PropagateCharacterizationFrequency()` to set the output characterization frequency.

```
//Modulator.h
#pragma once
#include "SystemVue/ModelBuilder.h"
#include "SystemVue/TimedDFModel.h"
#include "SystemVue/EnvelopeSignal.h"

class Modulator : public SystemVueModelBuilder::
TimedDFModel
{
    DECLARE_MODEL_INTERFACE( Modulator )
    virtual bool Run();
    ERESULT PropagateCharacterizationFrequency();
    double CarrierFrequency;

    //Complex baseband I-Q signal
    SystemVueModelBuilder::DComplexCircularBuffer input;

    //Envelope signal
    SystemVueModelBuilder::EnvelopeCircularBuffer
output;
};
```

```
//Modulator.cpp
#include "Modulator.h"

DEFINE_MODEL_INTERFACE( Modulator )
{
    SystemVueModelBuilder::DFParam
paramCarrierFrequency = ADD_MODEL_PARAM(
CarrierFrequency );
    paramCarrierFrequency.SetDefaultValue( "1e6" );
    ADD_MODEL_INPUT( input );
    ADD_MODEL_OUTPUT( output );
    return true;
}

ERESULT Modulator::PropagateCharacterizationFrequency()
{
    //Set output envelope signal characterization
frequency to be carrier freqnecy
    output.SetCharacterizationFrequency(
CarrierFrequency );
    return NOERROR_;
}

bool Modulator::Run()
{
    //Assign input complex baseband I-Q value to output
envelope signal with associated carrier frequency
    output[0] = input[0];
    return true;
}
```

### Example 9: Writing C++ Models that Control the Simulation

The following `FileWriter` shows an example that uses `SystemVueModelBuilder::DFSinkControl` to control the simulation.

```
#pragma once
```

```
#include "SystemVue/ModelBuilder.h"
#include "SystemVue/SimulationControl.h"
#include <iostream>
#include <fstream>

class FileWriter : public SystemVueModelBuilder::DFModel
{
public:
    FileWriter();
    ~FileWriter();

    DECLARE_MODEL_INTERFACE( FileWriter );

    bool Initialize();
    bool Run();
    bool Finalize();

    SystemVueModelBuilder::DoubleCircularBuffer input;

    int NumToCollect;
    char* FileName;

private:
    size_t m_iBuffer;
    double* m_pdBuffer;
    SystemVueModelBuilder::SinkControl m_control;
    std::ofstream outputFile;
};
```

```
#include "FileWriter.h"

// Buffer size to speed up writing of data
#define FILEWRITER_BUFFER_SIZE 1000000

FileWriter::FileWriter()
{
    NumToCollect = 1;
    FileName = 0;
    m_pdBuffer = 0;
    m_iBuffer = 0;
}

FileWriter::~FileWriter()
{
    // delete the buffer
    delete [] m_pdBuffer;
}

#ifndef SV_CODE_GEN
DEFINE_MODEL_INTERFACE( FileWriter )
```

```
{
    ADD_MODEL_INPUT( input );
    ADD_MODEL_PARAM( NumToCollect );
    SystemVueModelBuilder::DFParam fName =
ADD_MODEL_PARAM( FileName );
    fName.SetParamAsFile();
    return true;
}
#endif

bool FileWriter::Initialize()
{
    bool bStatus = true;

    // Check to make sure parameters are correct
    if ( FileName == 0)
    {
        POST_ERROR("FileName is not specified.");
        bStatus = false;
    }

    if ( NumToCollect < 1)
    {
        POST_ERROR("NumToCollect must be greater than
0.");
        bStatus = false;
    }

    {
        // Windows limits files to be 2 gigabytes in
size 2^31 = 1 << 31
        unsigned long iMaxCollect = ( unsigned(1) << 31)
/sizeof( double);
        if ( NumToCollect > iMaxCollect)
        {
            char str[128];
            sprintf(str,"NumToCollect must less than or
equal to %d", iMaxCollect);
            POST_ERROR( str);
            bStatus = false;
        }
    }

    // If parameters are valid, open the file
    if ( bStatus)
    {
        outputFile.open(FileName, std::ios::binary|std::
ios::out);
        if ( outputFile == false)
        {
            POST_ERROR("Cannot open file.");
            bStatus = false;
        }
        if ( bStatus)
        {
            // Initialize the data collection
controller - it will tract the data collected
            // and declare to SystemVue when the sink
is done collecting data.
            bStatus = m_control.Initialize( this, 0,
NumToCollect-1 );
        }

        if ( bStatus)
        {
            // If all OK - now we initialize the write
buffer
            m_pdBuffer = new double[FILEWRITER_BUFFER_SI
ZE];
            m_iBuffer = 0;
        }
    }

    return bStatus;
}

bool FileWriter::Run()
{
    if ( m_control.CollectData() )     // Check if we
should still collect data
    {
        // Write data into buffer
        m_pdBuffer[m_iBuffer++] = input[0];

        // If buffer is full, write it out to disk
        if ( m_iBuffer == FILEWRITER_BUFFER_SIZE)
        {
            m_iBuffer = 0;
            outputFile.write( reinterpret_cast<char*>
(m_pdBuffer), sizeof(double)*FILEWRITER_BUFFER_SIZE);
        }

    }
    return true;
}

bool FileWriter::Finalize()
{
    // Flush the rest of the buffer to disk
    if (m_iBuffer > 0)
        outputFile.write( reinterpret_cast<char*>
(m_pdBuffer), sizeof(double)*m_iBuffer);

    // Delete the buffer
    delete [] m_pdBuffer;
    m_pdBuffer = 0;

    // Close the file
    outputFile.close();

    return true;
}
```

### Example 10: Using MATLAB Generated C Libraries in C++ Models

This example introduces how to convert an MATLAB function into static link library and use it in SystemVue C++ Model. For more information, refer to the documentation for using third party library in C++ models.

This MATLAB code implements a feed-forward equalizer (FFE) function. Suppose it is written in a file called "MyFFE.m".

```
% MyFFE.m

function [ out ] = MyFFE( Coefficients, SamplesPerBit,
Reset, in )

% Declare persistent in order to preserve internal state

persistent dSamples;
persistent numSamples;
persistent taps;

if ( isempty(dSamples) || Reset )
    numSamples = length(Coefficients) * SamplesPerBit;
    dSamples = zeros(1, numSamples);
    taps = Coefficients';
end

dSamples = [in,dSamples(1:numSamples-1)];
out = dSamples(1:SamplesPerBit:numSamples) * taps;

end
```

NOTE Users can declare **persistent** variables in MATLAB function to preserve internal state.

By using the following MATLAB command, MATLAB coder compiles "MyFFE.m" into "MyFFE.h", "MyFFE.lib", as well as other relevant files. Please refer to MATLAB document for more details.

```
codegen -config:lib -args {coder.typeof(double(0), [1
Inf]), coder.typeof(uint32(16)), coder.typeof(uint32(0))
, coder.typeof(double(0)) } MyFFE.m
```

The following code segment is generated by MATLAB coder as part of "MyFFE.h", which declares MyFFE function in C. MyFFE function is the entry point that performs generated FFE operation in the library ("MyFFE.lib").

```
extern double MyFFE(const emxArray_real_T
*Coefficients, unsigned int SamplesPerBit, unsigned int
Reset, double in);
```

The following code segment is also generated by MATLAB compiler as part of "libmyffe.h", which initializes and terminates the library respectively.

```
extern void MyFFE_free(void);
extern void MyFFE_init(void);
```

The following code shows how to write a SystemVue C++ model, BlindFFE, that uses MATLAB -generated MyFFE function to perform FFE operation.

```
//BlindFFE.h

#pragma once
#include "SystemVue\ModelBuilder.h"

struct emxArray_real_T;

class BlindFFE : public SystemVueModelBuilder::DFModel
{
public:
    BlindFFE();
    ~BlindFFE();

    bool Initialize();
    bool Run();
    bool Finalize();

    // parameters
    double *m_dCoefs;
    int m_iCoefsSize;
    int m_iSamplesPerBit;

    // i/o
    double m_dInput, m_dOutput;

    DECLARE_MODEL_INTERFACE(BlindFFE);
private:
    emxArray_real_T *m_pMatlabCoefs;
    bool m_bReset;
};
```

```
//BlindFFE.cpp

#include "BlindFFE.h"

extern "C" {
#include "../M_Code_Model/codegen/lib/MyFFE/MyFFE.h"
}

#ifndef SV_CODE_GEN
DEFINE_MODEL_INTERFACE(BlindFFE)
{
    SET_MODEL_DESCRIPTION("Feed-Forward Equalizer");
    ADD_MODEL_HEADER_FILE("BlindFFE.h");
    SET_MODEL_CATEGORY("IBIS-AMI Transceivers");

    SystemVueModelBuilder::DFPort port = ADD_MODEL_INPUT
(m_dInput);
    port.SetName("input");

    port = ADD_MODEL_OUTPUT(m_dOutput);
    port.SetName("output");

    SystemVueModelBuilder::DFParam param =
ADD_MODEL_ARRAY_PARAM(m_dCoefs,m_iCoefsSize);
    param.SetName("Coefficients");
    param.SetDefaultValue("[1 0.1 0.2]");
    param.SetDescription("Bit level FFE taps");

    param = ADD_MODEL_PARAM(m_iSamplesPerBit);
    param.SetName("SamplesPerBit");

    return true;
}
#endif

BlindFFE::BlindFFE() : m_pMatlabCoefs(0)
{
    m_iSamplesPerBit = 16; // default value
}

BlindFFE::~BlindFFE()
{
}

bool BlindFFE::Initialize()
{
    bool bStatus = true;

    //create coefficient array for MEX function
    m_pMatlabCoefs = new emxArray_real_T;
    m_pMatlabCoefs->allocatedSize = m_iCoefsSize;
    m_pMatlabCoefs->canFreeData = boolean_T(false);
    m_pMatlabCoefs->data = m_dCoefs;
    m_pMatlabCoefs->numDimensions = 2;
    m_pMatlabCoefs->size = new int[2];
    m_pMatlabCoefs->size[0] = 1;
    m_pMatlabCoefs->size[1] = m_iCoefsSize;

    MyFFE_init();

    if (m_iCoefsSize<1)
        bStatus = false;

    return bStatus;
}

bool BlindFFE::Run()
{
    bool bStatus = true;

    m_dOutput = MyFFE(m_pMatlabCoefs, m_iSamplesPerBit,
0, m_dInput);

    return bStatus;
}

bool BlindFFE::Finalize()
{
    bool bStatus = true;

    MyFFE_free();

    if (m_pMatlabCoefs) {
        delete [] m_pMatlabCoefs->size;
        delete m_pMatlabCoefs;
    }

    return bStatus;
}
```

NOTE

Call MATLAB-generated initialize function, e.g., MyFFE_init(), in the Initialize() method of the SystemVue model to properly initialize the library. Also, call MATLAB-generated free function, e.g., MyFFE_free();, in the Finalize() method of the SystemVue Model to properly close the library.

The following steps guide users to set up a Visual Studio project to build MyFFE into SystemVue model library.

1. Follow these steps to set up a Visual Studio project. Suppose the project name is "SystemVue_Matlab_Coder_Models".
2. Copy the folder containing "MyFFE.h", "MyFFE.lib" and the other header files to the project directory and add it to include directory and link directory.
3. Add "MyFFE.h" into the project Header Files.
4. Add "MyFFE.lib" into the link libraries.
5. Create "BlindFFE.h" and "BlindFFE.cpp" as shown above into the project.
6. Build the solution. The resulting "Matlab_Coder.dll" is a custom SystemVue library that contains BlindFFE model.

**CAUTION** If there is a persistent variable in the MATLAB code, only one instance of such SystemVue model can be placed on the schematic. If there are multiple SystemVue model instances that invoke the same MATLAB function, such persistent variable will be shared by multiple instances, and may cause unexpected simulation results.

## C++ Code Generation

### Introduction

In this tutorial, you will learn how to use custom C++ models that you have written and use them with C++ Code Generation with care, you can develop C++ models that can be reused in C++ code generation.

### Example: Using A Gardner Timing Recovery C++ Model in a Code Generated System

In this example, we will be using the GAD_timing C++ model that was developed in the C++ Model Development Tutorial.

1. Open the tutorial example located in: "Tutorials\QPSK_Transceiver_Design\FloatingPoint\QPSK_FloatingPoint.wsv".
2. Open the C++ model Visual Studio Solution with the GAD_timing model that you developed in the C++ Model Development tutorial. In Visual Studio's *Solution Explorer*, right click on the *INSTALL* project and select *Build*.
3. Load the MyProject.dll into SystemVue and verify that the GAD_timing model is available.
4. Use the *Manage Models* button for the *GAD_timing subnetwork* part instance and add the *GAD_timing* model to the manage models list.
5. Switch to new *GAD_timing* C++ model.
6. Open the *Cpp1*C++ Code Generator that is part of the example workspace. In the C++ Code Generation Options dialog:
    a. Set the *Output Directory* to a new directory named *MyProject2*.

   b. Check the checkbox to *Automatically add generated model to Part model list*.

   c. Finally, click the *Generate Now* button.

   d. This compile will fail as you are missing the include file *GAD_timing.h* that was developed in the *MyProject* Visual Studio solution. You will need to find the full the path to *GAD_timing.h*. The file will be in the source/SystemVue subdirectory of the *MyProject* solution.

7. Edit the *CMakeLists.txt* file in your *SystemVue-MyProject2* project and add the directory to the include path by using the `include_directories()` CMake macro. You must make sure to use forward slashes instead of back-slashes in the path and if your path has spaces, you must surround the path with "". Additionally, the `include_directories` must be defined before the `SVU_ADD_CG_MODELBUILDER_LIBRARY` CMake macro call.

8. You will next have to add the *GAD_timing.cpp*to your list of files that you are compiling into your library. As it is not in the same directory, you will need to have the full file path. When you are done, your edits must look like:

```
include_directories( "C:/Users/myname/Documents/My
Workspaces/MyProject/source/SystemVue" )

SVU_ADD_CG_MODELBUILDER_LIBRARY(
    SystemVue-MyProject2
    ${SVU_GENERATED_FILES}
    ${VERSION_FILE}
    "C:/Users/myname/Documents/My Workspaces
/MyProject/source/SystemVue/GAD_timing.cpp"
)
```

9. To compile and reload, you can execute one of the following two choices:

   - Through SystemVue: Open the *Cpp1* C++ Code Generator and click *Generate Now*.

   - Through Visual Studio: Right-click on the *INSTALL* project in your *Solution Explorer* and select *Build*. (Just click *OK* on all the popup message dialogs, they are triggered by the need to update Visual Studio content due to changes in `CMakeLists.txt` file).

10. Load the MyProject2.dll and add the_qpsk_demod@MyProject2 Models_to the *qpsk_demod* part (using *Manage Models*).

**NOTE** If you make a mistake, CMake might not be able to create a Visual Studio project. If this happens, you will need to run the Model Builder Batch to understand and fix the issues in your `CMakeLists.txt` file.

# Subnetwork Recursion: Automatically Constructing Repetitive Data Flow Schematics

## Introduction

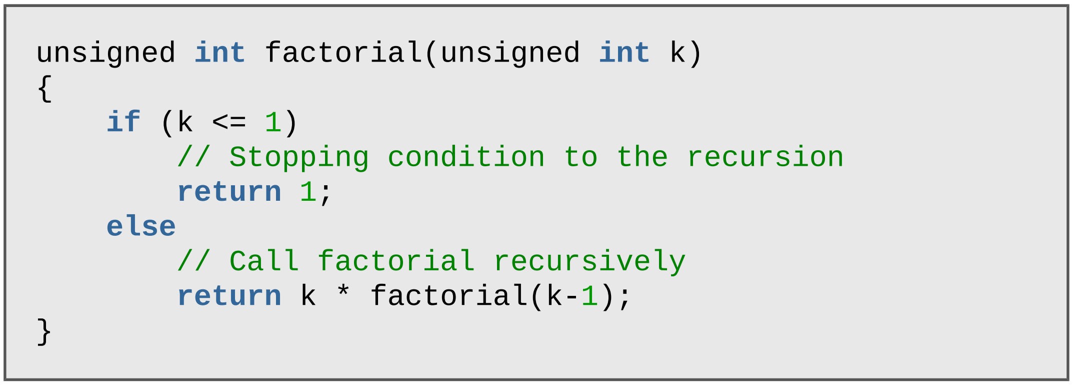
There are many interesting signal processing systems that are constructed using repetitive structures. In this tutorial, you will learn how to use *Subnetwork Recursion* to construct these systems. Examples of uses of these systems are:

- FIR filters
- Filter banks
- Add and Multiplier trees (for hardware design)
- Galois field multiplier (for hardware design, see the "Hardware Design\HDLCodeGeneration\Reed_Solomon\RSEncoder.wsv" example)

In computer programming, *recursion* is defined as a function calling itself. One of the easiest recursive algorithms to understand is factorial:

`k! = k * (k-1) * (k - 2) * ... * 1`

Below is the implementation of factorial in C++:



```
unsigned int factorial(unsigned int k)
{
    if (k <= 1)
        // Stopping condition to the recursion
        return 1;
    else
        // Call factorial recursively
        return k * factorial(k-1);
}
```

To implement a proper recursive function you must have a stopping condition where the recursive function returns a value instead of calling itself. For factorial, the stopping condition is `k == 1`.

In *Subnetwork Recursion*, you must also implement a stopping condition. In *Subnetwork Recursion*, the function is implemented with a *subnetwork*. To implement recursion, the *subnetwork* is instantiated within itself. To implement the stopping condition, a *subnetwork* part is disabled by making it either open or short by controlling the part behavior with an equation.

The *Subnetwork Recursion* is expanded during the initialization of a schematic for simulation or code generation. Thus unlike the C++ factorial example above, the schematic topology is fixed and cannot change during within run making it ideally suited to describe repetitive topologies for use in either context.

### Example 1: Constructing the Fourier Series Approximation of a Square Wave using Subnetwork Recursion

In this example, you will explore how the Fourier series of a square wave can be implemented using Subnetwork recursion.

The Fourier series of a square wave is shown below:

$$
\begin{aligned}
x_{\text{square}}(t) &= \frac{4}{\pi}\sum_{k=1}^{\infty}\frac{\sin(2\pi(2k-1)ft)}{(2k-1)} \\
&= \frac{4}{\pi}\left(\sin(2\pi ft) + \frac{1}{3}\sin(6\pi ft) + \frac{1}{5}\sin(10\pi ft) + \cdots\right)
\end{aligned}
$$

1. Open the example: "Tutorials\Algorithm_Design\Subnetwork_Recursion.wsv".
2. The *SquareWave* design approximates a square wave with a Fourier series of length `k`. Note that the top-level schematic has a *Gain* that corresponds to the

   $$\frac{4}{\pi}$$

   constant shown above.
3. Move the slider to control the number of elements in the Fourier series approximation. As expected when `k = 1` you see a sine wave. As `k` increases, you have a much better approximation of a square wave and can observe Gibbs phenomenon.
4. Open the *FourierSeries* subnetwork, observe that the last term (kth) of the Fourier series is instantiated and added to the *FourierSeries* with `k-1` terms. A good practice to follow when constructing a recursive subnetwork is to set the default parameters of the subnetwork to the stop condition (in this case when `k == 1`). Note with this default, the *FourierSeries* subnetwork is set to *Disabled: OPEN*.
5. Explore the stopping condition by double-clicking on the *FourierSeries* part and then clicking on the part behavior (currently set to *Equation Controlled*). Click on the *Part Behavior* symbol, and select *Control by Equation...*.

### Example 2: Calculating Factorial using Subnetwork Recursion

The solution to this example is supplied in the "Tutorials\Algorithm_Design\Subnetwork_Recursion.wsv" example workspace. Before looking at the solution, implement a recursive subnetwork to calculate the factorial. This subnetwork should:

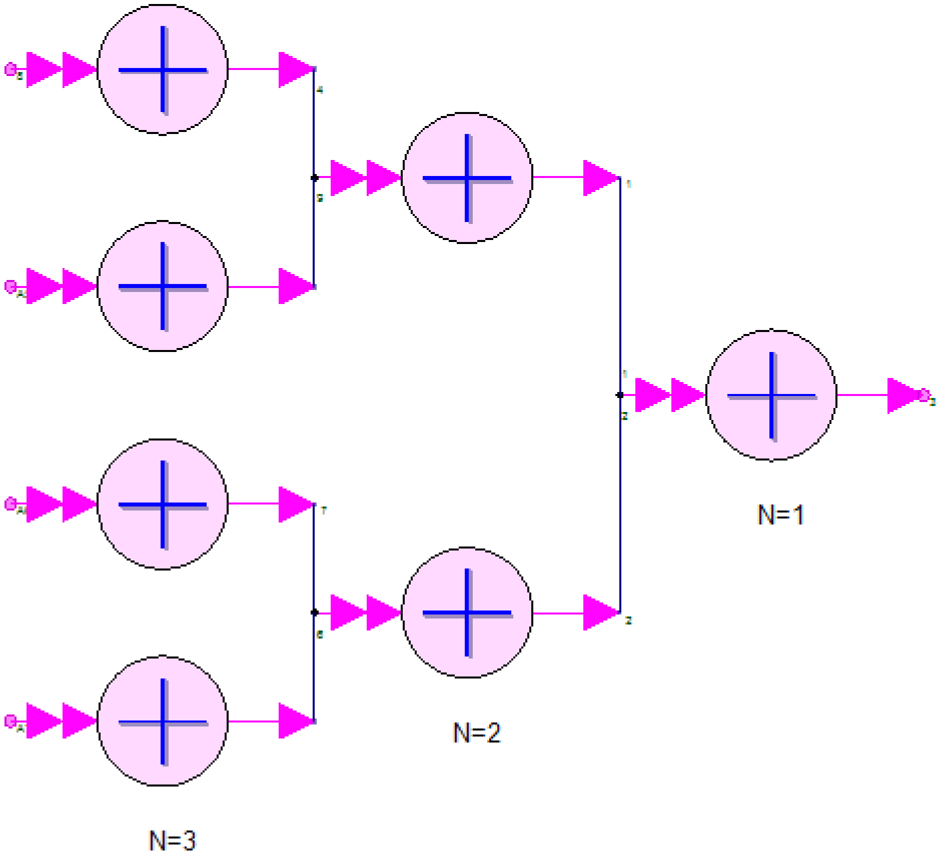
- Define a parameter k.
- Set the default value of k to be the stopping condition of the recursion.
- Output the constant value k!.
- Be implemented using *subnetwork recursion*.

### Example 3: Synthesizable Adder Tree

The last example in the "Tutorials\Algorithm_Design\Subnetwork_Recursion.wsv" workspace is an adder tree implemented with *subnetwork recursion*. Adder trees are used in hardware design when you want to add more than two numbers together. Explore this design, pay attention to how a variable-width bus is used to construct a design that expands to add

$$2^N$$

numbers.



## Cosimulation with SystemC

In this tutorial, you will learn how to use SystemC models inside SystemVue through the demonstration of various use cases using the shipping examples. The examples referenced in this tutorial can be found in the following directory:

```
<SystemVue Installation Directory> \ Examples \ Model Building \
SystemC Modeling
```

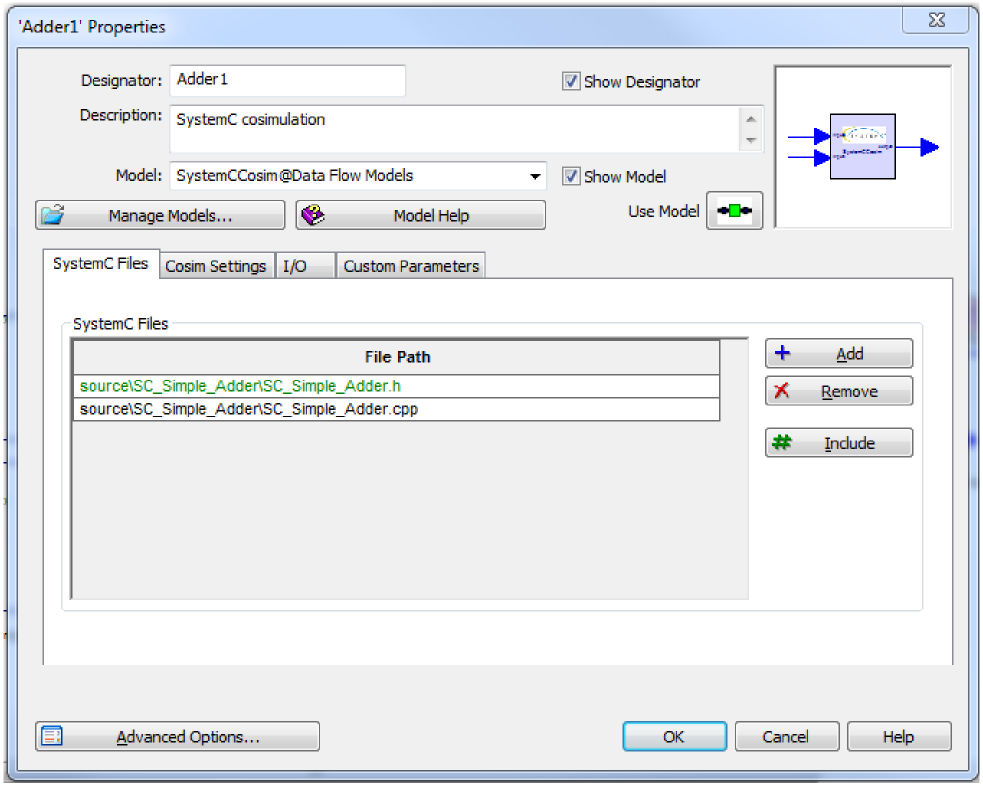
The SystemC source files are located in the same path, inside the "source" folder. To quickly open an example from SystemVue, p ress 'Ctrl + E ' or select Help > Open Example. It is recommended to read SystemC Cosimulation documentation before starting this tutorial .

CAUTION If the **Examples** directory on your computer is **Read Only**, you should copy the "SystemC Modeling" folder to a writable location and run the example workspaces from there.

### Example 1: Setting Up the SystemCCosim Model

To demonstrate how to set up the SystemCCosim model, you will start with a simple SystemC Adder example. The design related to this example is the *Adder design* schematic in the *Adder* folder inside the SystemC_Cosim.wsv workspace. The SystemC source files that implement the adder example are SC_Simple_Adder.h and SC_Simple_Adder.cpp under "source \ SC_Simple_Adder".

1. Create a new blank workspace inside SystemVue. An empty design schematic should open.
2. On the "Part Selector A" window, go to the "Algorithm Design" library, under the category "Cosimulation", and select the SystemCCosim part. Place this part on the schematic.
3. Double-click on the SystemCCosim part on the schematic to open the custom graphical user interface for SystemCCosim inside the part properties dialog. The first page (tab) shown in this dialog is the "SystemC Files" page. Click the Add button and browse the "source" directory to locate the two files mentioned above (under "source\SC_Simple_Adder"). Select both files and click OK. Now, the "SystemC Files" page should look like the picture below. As you can see, the first file is highlighted with green color, meaning that this file contains the declaration of the SystemC model to be simulated. In this case the SC_Simple_Adder.h is the correct one. However, say if the declaration is in another file, you can select the right file in the list and click the Include button on the right:



'Adder1' Properties

Designator: Adder1 ☑ Show Designator

Description: SystemC cosimulation

Model: SystemCCosim@Data Flow Models ☑ Show Model

Manage Models... | Model Help | Use Model

SystemC Files | Cosim Settings | I/O | Custom Parameters

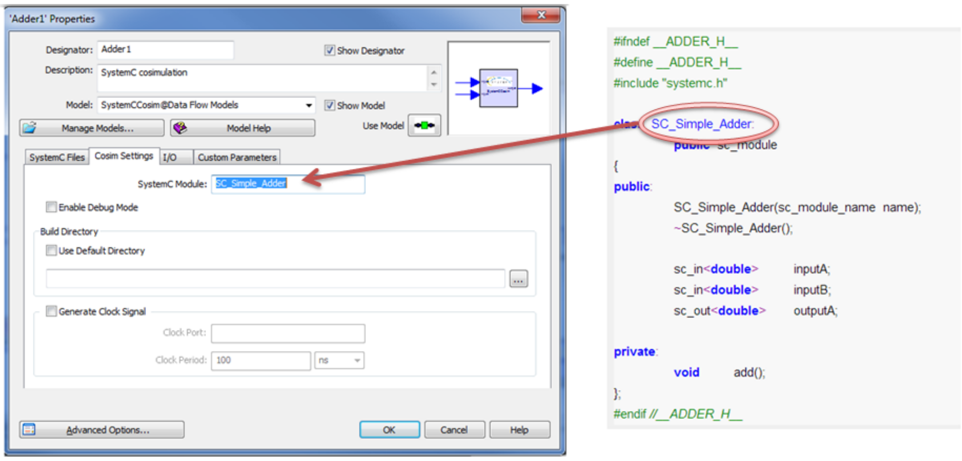
SystemC Files

| File Path |
|---|
| source\SC_Simple_Adder\SC_Simple_Adder.h |
| source\SC_Simple_Adder\SC_Simple_Adder.cpp |

Add | Remove | Include

Advanced Options... | OK | Cancel | Help

4. Next, go to the "Cosim Settings" page. Fill the SystemC Module and Build Directory fields (source code of SC_Simple_Adder.h is provided on the right for reference):



'Adder1' Properties

Designator: Adder1 ☑ Show Designator

Description: SystemC cosimulation

Model: SystemCCosim@Data Flow Models ☑ Show Model

Manage Models... | Model Help | Use Model

SystemC Files | Cosim Settings | I/O | Custom Parameters

SystemC Module: SC_Simple_Adder

☐ Enable Debug Mode

Build Directory

☐ Use Default Directory

☐ Generate Clock Signal

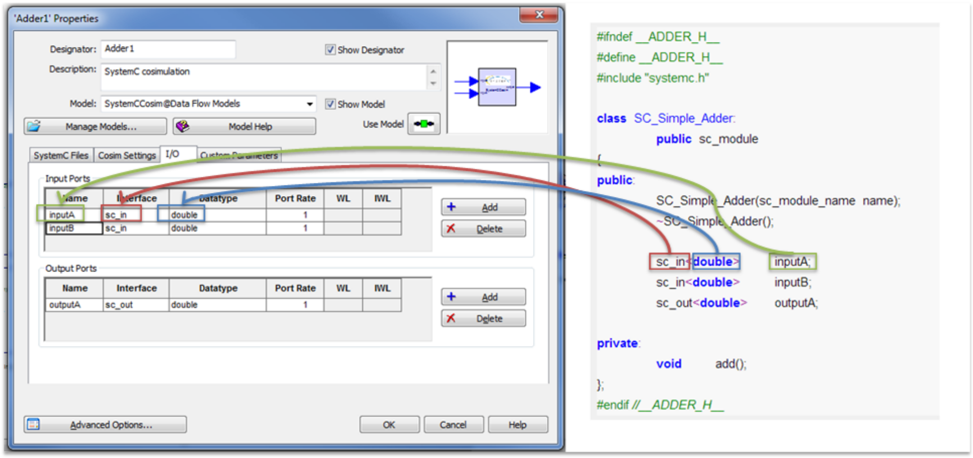
Clock Port:

Clock Period: 100 ns

Advanced Options... | OK | Cancel | Help

```
#ifndef __ADDER_H__
#define __ADDER_H__
#include "systemc.h"

class SC_Simple_Adder:
          public sc_module
{
public:
          SC_Simple_Adder(sc_module_name  name);
          ~SC_Simple_Adder();

          sc_in<double>       inputA;
          sc_in<double>       inputB;
          sc_out<double>      outputA;

private:
          void       add();
};
#endif //__ADDER_H__
```

5. Finally, go to the "I/O" page and define the port characteristics (Name, Interface, Datatype, Port Rate) as these are defined in the source code file (source code of SC_Simple_Adder.h is provided on the right for reference):



'Adder1' Properties

Designator: Adder1 ☑ Show Designator

Description: SystemC cosimulation

Model: SystemCCosim@Data Flow Models ☑ Show Model

Manage Models... | Model Help | Use Model

SystemC Files | Cosim Settings | I/O | Custom Parameters

Input Ports

| Name | Interface | Datatype | Port Rate | WL | IWL |
|---|---|---|---|---|---|
| inputA | sc_in | double | 1 | | |
| inputB | sc_in | double | 1 | | |

Output Ports

| Name | Interface | Datatype | Port Rate | WL | IWL |
|---|---|---|---|---|---|
| outputA | sc_out | double | 1 | | |

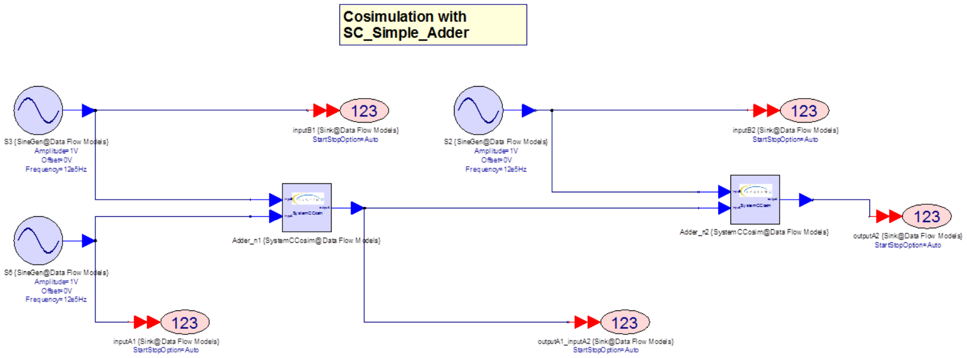
Add | Delete

Advanced Options... | OK | Cancel | Help

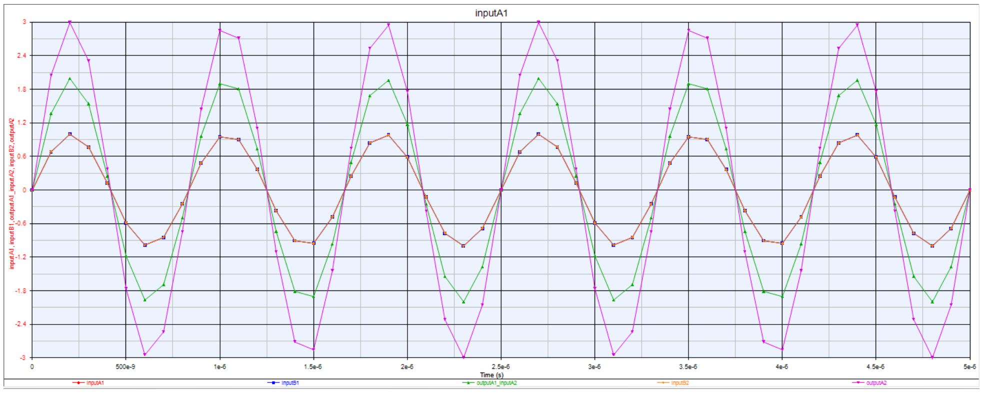
```
#ifndef __ADDER_H__
#define __ADDER_H__
#include "systemc.h"

class  SC_Simple_Adder:
            public  sc_module
{
public:
            SC_Simple_Adder(sc_module_name  name);
            ~SC_Simple_Adder();

            sc_in<double>        inputA;
            sc_in<double>        inputB;
            sc_out<double>       outputA;

private:
            void        add();
};
#endif //__ADDER_H__
```

6. At this point, we can save the changes by clicking OK. There is no need to visit the "Custom Parameters" page for this example, as there are no extra parameters (apart from the default "sc_module_name" one) inside the module's constructor.

7. Now we have already configured the SystemCCosim model, and can use it inside a data-flow design. In this example, we use two instances of the same SystemC model, as you can see in the figure below. The first one adds the values of two sine generators and writes the result to its output port. The second one adds this result to the value of a third sine generator.


Cosimulation with
SC_Simple_Adder
S3 {SineGen@Data Flow Models}
Amplitude=1V
Offset=0V
Frequency=12e5Hz
123
inputB1 {Sink@Data Flow Models}
StartStopOption=Auto
S2 {SineGen@Data Flow Models}
Amplitude=1V
Offset=0V
Frequency=12e5Hz
123
inputB2 {Sink@Data Flow Models}
StartStopOption=Auto
SystemCCosim
Adder_n1 {SystemCCosim@Data Flow Models}
SystemCCosim
Adder_n2 {SystemCCosim@Data Flow Models}
123
outputA2 {Sink@Data Flow Models}
StartStopOption=Auto
S6 {SineGen@Data Flow Models}
Amplitude=1V
Offset=0V
Frequency=12e5Hz
123
inputA1 {Sink@Data Flow Models}
StartStopOption=Auto
123
outputA1_inputA2 {Sink@Data Flow Models}
StartStopOption=Auto

8. Finally, we run the simulation and observe the results. A graph with the name *Adder_graph* can be found in the *Adder* folder inside the workspace. Alternatively, you can check the results using the *Adder_Data* dataset (again inside the "Adder" folder).



## Example 2: Template SystemC Model and SystemC Datatypes

This example demonstrates how to configure the SystemCCosim model for a Template SystemC model and how to specify the I/O page for various SystemC datatypes.

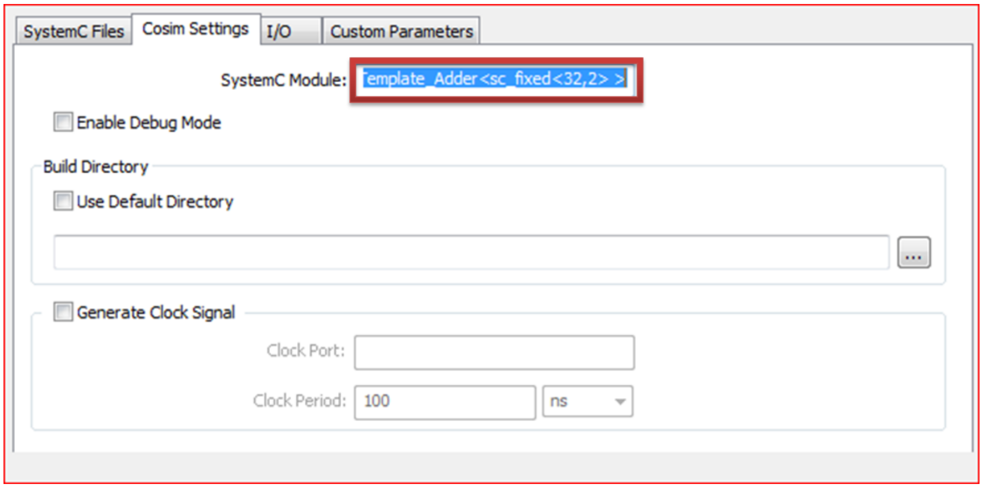
The design (schematic) used by this example is the *Adder design* in the *Combinational* folder inside the IO_Customizations.wsv workspace. The template SystemC adder model is shown below, where the datatypes of all the input and output ports are defined by the template parameter T.

```
template<typename T>
class Template_Adder:
    public sc_module
{
public:

    Template_Adder(sc_module_name name);
    ~Template_Adder();

    sc_in<T>    inputA;
    sc_in<T>    inputB;
    sc_out<T>    result;

private:

    void    add();

};
```

For the SystemCCosim model *Adder_Fxp_L* on *Adder design* schematic, the template datatype T is set to SystemC fixed-point with word length 32 bit and integer word length 2 bits. The corresponding format in SystemC would look like this: sc_fixed<32,2 >

Comparing to the SC_Simple_Adder model in the previous example (Example 1), here are the differences:

1. The SystemC Module field should specify the instantiated template model, i. e. Template_Adder< sc_fixed<32,2 > >



SystemC Module: Template_Adder<sc_fixed<32,2> >

□ Enable Debug Mode

Build Directory

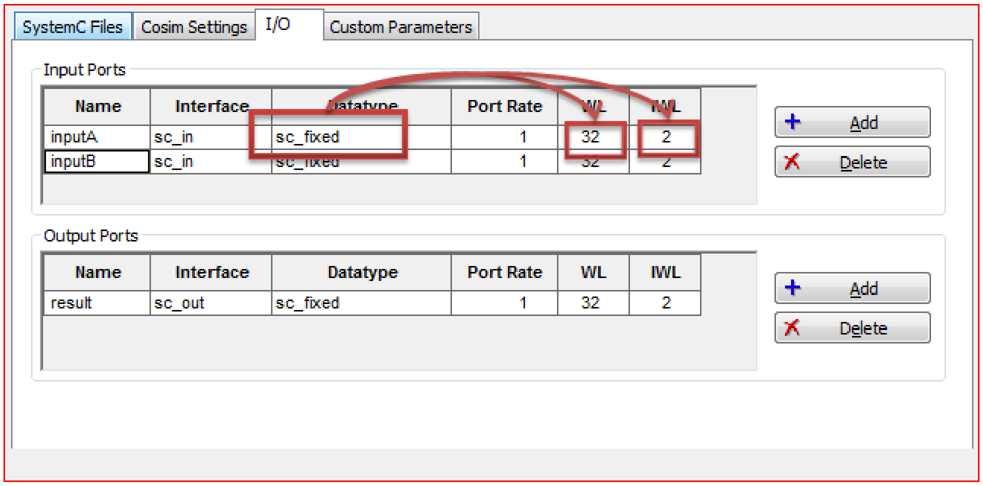
□ Use Default Directory

□ Generate Clock Signal

Clock Port:

Clock Period: 100 ns

2. For *sc_fixed* datatype (the same applies to *sc_ufixed*), the WL and IWL fields must specify the right word length and integer word length:



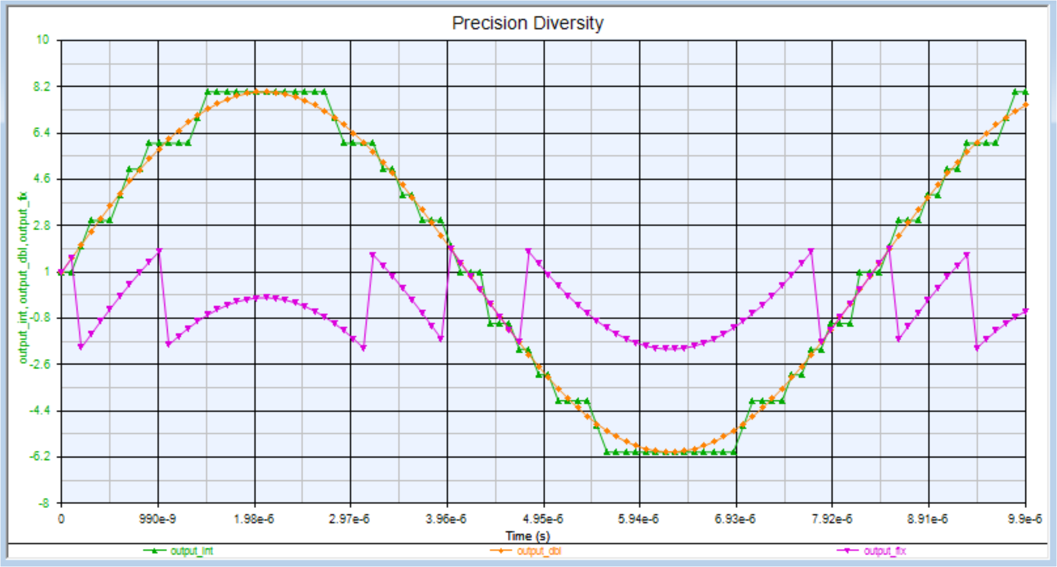
Input Ports

| Name | Interface | Datatype | Port Rate | WL | IWL |
|---|---|---|---|---|---|
| inputA | sc_in | sc_fixed | 1 | 32 | 2 |
| inputB | sc_in | sc_fixed | 1 | 32 | 2 |

Output Ports

| Name | Interface | Datatype | Port Rate | WL | IWL |
|---|---|---|---|---|---|
| result | sc_out | sc_fixed | 1 | 32 | 2 |

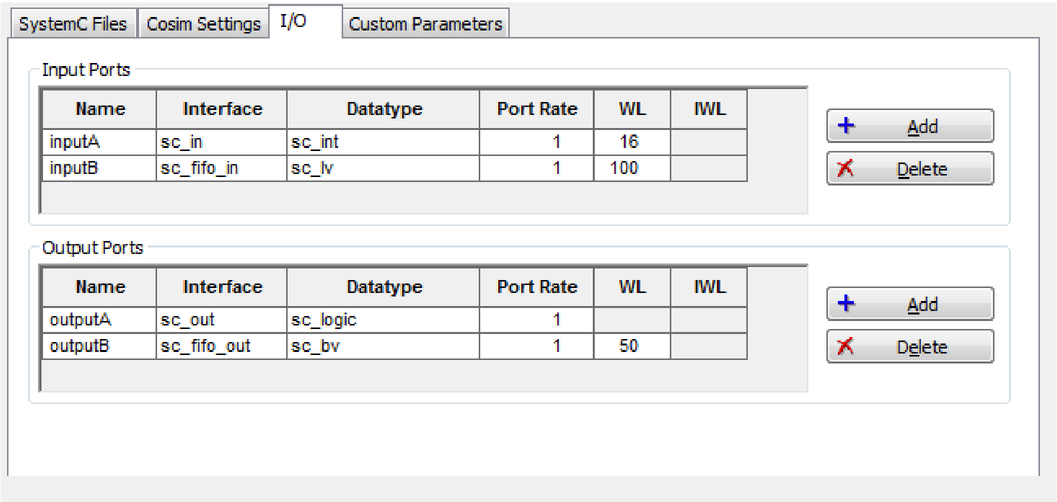
The same SystemC model template is used by the other two SystemCCosim models *Adder_Int* and *Adder_Dbl*, where the SystemC Module field is set to Template_Adder<int> and Template_Adder<double> respectively. The screenshot below shows the outputs of the three SystemC adder models using double, int, and sc_fixed<32,2> datatypes. The quantization effects are clearly shown in the graph when comparing the floating point double precision output (orange) with integer (green) and fixed point (purple) representations.



Finally, in order to illustrate the use of various SystemC datatypes, we use the sample code below:

```
template<typename T>
class SampleSystemCModel:
    public sc_module
{
public:
    SampleSystemCModel(sc_module_name name);
    ~SampleSystemCModel();

    sc_in< sc_int<16> >            inputA;
    sc_fifo_in< sc_lv<100> >    inputB;
    sc_out< sc_logic >            outputA;
    sc_fifo_out< sc_bv<50> >        outputB;
private:
    void    add();
};
```

With the above sample code, the corresponding I/O configuration should look like this:



Input Ports

| Name | Interface | Datatype | Port Rate | WL | IWL |
|---|---|---|---|---|---|
| inputA | sc_in | sc_int | 1 | 16 | |
| inputB | sc_fifo_in | sc_lv | 1 | 100 | |

Output Ports

| Name | Interface | Datatype | Port Rate | WL | IWL |
|---|---|---|---|---|---|
| outputA | sc_out | sc_logic | 1 | | |
| outputB | sc_fifo_out | sc_bv | 1 | 50 | |

## Example 3: Custom Parameters

This example describes how to correctly use parameters of a SystemC model. In the context of SystemCCosim, "Custom Parameters" refer to the constructor parameters of a SystemC model. Detailed information on the acceptable datatypes of the constructor parameters can be found in the documentation of Custom Parameters Tab. In this example, two custom parameter cases are demonstrated:

- Array Parameter
- Enumeration Parameter

### Array Parameter

We use the design (schematic) *FIR_cust_coef* in folder *FIR_custom_coefficients* of the SystemC_Parameters_Passing.wsv workspace to illustrate array parameters.

In this design, a SystemC FIR filter is being evaluated with the use of Additive White Gaussian noise as input. To acquire the Power response of the filter, a Spectrum Analyzer has been connected at its output. The type of the filter is configured by an Array Parameter (*iCoefs[]*), where the coefficients are defined. In the current design, the coefficients have been defined in such a way that the SystemC model represents a low-pass FIR filter.

The corresponding SystemC source files can be found inside the folder "source\fir". The constructor of this model is as follows:

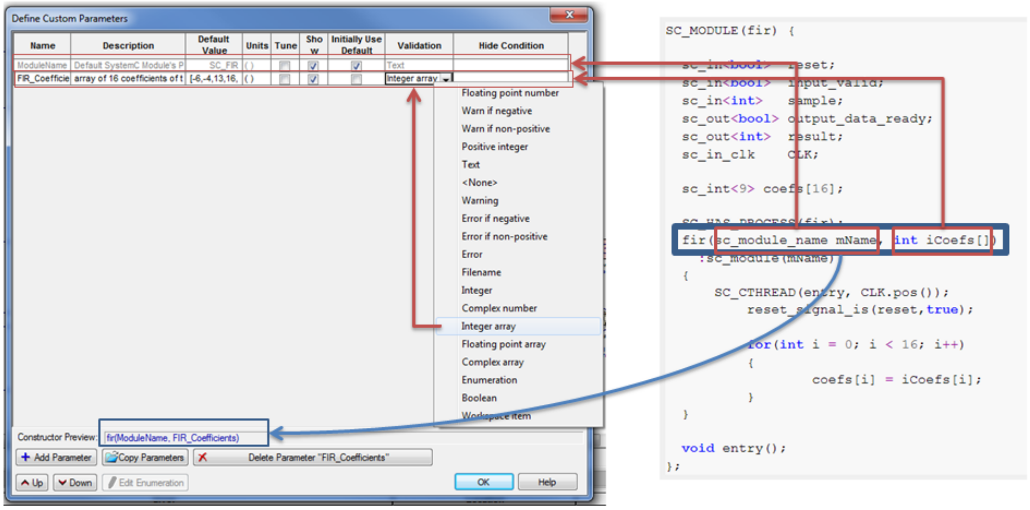
- *fir( sc_module_name mName, int iCoefs[ ] )*

With the above constructor, the "Custom Parameters" of the SystemCCosim block should define two parameters: *mName* and *iCoefs[]*.

The first parameter *mName* is a default parameter that every SystemC module constructor must contain. For user's convenience, this parameter is created automatically with the instantiation of a new SystemCCosim block. The name of the parameter inside the parameter list is "ModuleName" and the default value is the Designator of the block (part). In this case, *mName* will take the value "SC_FIR".

The second parameter is an integer array. To define this parameter, the user should:

1. Click on the Define Custom Parameters... button of the Custom Parameters page. A new dialog will open (see the screenshot below).
2. Choose Add Parameter.
3. Define the name of the parameter (it is not necessary to be the same as the name of the constructor parameter, what matters is that the order of the parameters must match the constructor). The rest fields are optional except the Validation.
4. Choose the correct Validation type. In this case, Integer Array is the proper one.
5. Finally, verify the correct order of the parameters by looking at the Constructor Preview field.



Define Custom Parameters

| Name | Description | Default Value | Units | Tune | Show | Initially Use Default | Validation | Hide Condition |
|---|---|---|---|---|---|---|---|---|
| ModuleName | Default SystemC Module's P | SC_FIR | ( ) | ☐ | ☑ | ☑ | Text | |
| FIR_Coefficie | array of 16 coefficients of t | [-6,-4,13,16, | ( ) | ☐ | ☑ | ☐ | Integer array | |

Constructor Preview: fir(ModuleName, FIR_Coefficients)

```
SC_MODULE(fir) {

  sc_in<bool>  reset;
  sc_in<bool>  input_valid;
  sc_in<int>   sample;
  sc_out<bool> output_data_ready;
  sc_out<int>  result;
  sc_in_clk    CLK;

  sc_int<9> coefs[16];

  SC_HAS_PROCESS(fir);
  fir(sc_module_name mName, int iCoefs[])
    :sc_module(mName)
  {
      SC_CTHREAD(entry, CLK.pos());
          reset_signal_is(reset,true);

          for(int i = 0; i < 16; i++)
          {
                  coefs[i] = iCoefs[i];
          }
  }

  void entry();
};
```

As explained before, the coefficients have been defined in such a way that the FIR filter functions as a low-pass filter. However, the user is encouraged to update these values appropriately, to obtain different filter types.

### Enumeration Parameter

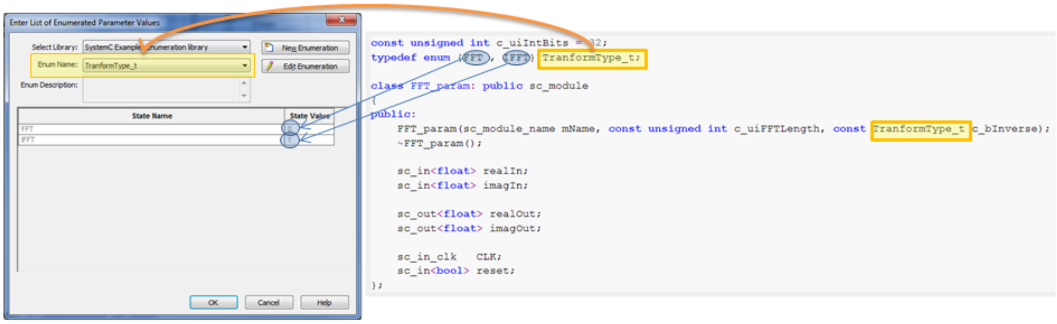
For the case of enumeration parameters, we will use the *FFT design* schematic in folder *FFT_parameters_passing* of the SystemC_Parameters_Passing.wsv workspace as the example.

In this design, two FFT SystemC models are being evaluated. Two equivalent numeric built-in FFT blocks are used to verify the correct functionality of the SystemC ones, respectively. Each SystemC block contains two parameters: One to define the FFT size and one to define the direction (forward or inverse). The point of interest here is the second one, as this is implemented as an enumeration parameter. In the current example, we use forward direction for both SystemC models. The user is encouraged to test the inverse case, by using the enumeration parameter provided.

The source code for "FFT_param" can be found in folder "source\fft\fft_ifft".

The steps to define such a parameter are described as follows:

1. Follow the same procedure to add new parameters (Click Define Custom Parameters... button and then Add Parameter in the popup window).
2. In the Validation field, choose Enumeration.
3. Click the Edit Enumeration button at the bottom of the Define Custom Parameters window.
4. In the opened dialog, click the New Enumeration button.
5. In the Enum Name field, put the name of the enumeration type in the SystemC code (yellow boxes in the picture below). In this case, put "TranformType_t" as defined in the source code.
6. Then, define the states of the enumeration. The State Value field should have the values defined (explicitly or implicitly) in the source code. Therefore, if the declaration is *enum{ FFT = 3, IFFT = 2}* (explicit definition) , the values for state FFT should be 3 and the value for state IFFT should be 2. If there is no explicit definition, like in the present case ( *enum{ FFT , IFFT }* ) , the State Value should take values starting from 0 and increase by 1, for every subsequent state.



Select Library: SystemC Example Enumeration library

Enum Name: TranformType_t

Enum Description:

| State Name | State Value |
|---|---|
| FFT | 0 |
| IFFT | 1 |

```
const unsigned int c_uiIntBits = 32;
typedef enum {FFT, IFFT} TranformType_t;

class FFT_param: public sc_module
{
public:
    FFT_param(sc_module_name mName, const unsigned int c_uiFFTLength, const TranformType_t c_bInverse);
    ~FFT_param();

    sc_in<float> realIn;
    sc_in<float> imagIn;

    sc_out<float> realOut;
    sc_out<float> imagOut;

    sc_in_clk   CLK;
    sc_in<bool> reset;
};
```

**CAUTION** Follow the Note in the SystemC_Parameters_Passing workspace to load the "SystemC Examples Enumeration library.xml" in order to see the predefined enumerations for the FFT_param example.

## Example 4: Multirate Ports

SystemCCosim supports two types of SystemC ports, the *sc_signal*, and the *sc_fifo*. The intent of using multirate for *sc_signal* and *sc_fifo* ports is different.

The Port Rate property for a *sc_signal* port defines how many times this signal should be sampled during *one firing* of the SystemCCosim model, which corresponds to *one period* of the SystemC simulation. In SystemC side, *one period* means the simulation duration that the SystemC simulation is running, for every firing of the SystemCCosim model inside SystemVue. During *one period*, the signal at the *sc_signal* port is sampled for "Port Rate" number of times at evenly spaced time instances. The sampling rate of the signal in SystemC is the defined by the sampling rate of the signal inside SystemVue.

For *sc_fifo* ports, the Port Rate property specifies how many samples should be read from (or written to) the FIFO channel of the input (or output) *sc_fifo* port of the SystemC model, in *one period* of the SystemC simulation. Here *one period* again corresponds to *one firing* of the SystemCCosim model.

From data flow point of view, in each firing, the SystemCCosim model consumes a constant number of samples from each input port, send the samples to the SystemC process for simulation of one period, receive the resulting samples from SystemC side, and produces a constant number of samples to each output port. The number of samples consumed (produced) at the input (output) port is defined by the corresponding Port Rate property in the I/O page.

A complete analysis of the way SystemC data are mapped to SystemVue samples, with respect to the Port Rate property, can be found in SystemVue-SystemC Synchronization. In the rest of this example, we will present two multirate designs that are included in the IO_Customizations.wsv workspace.

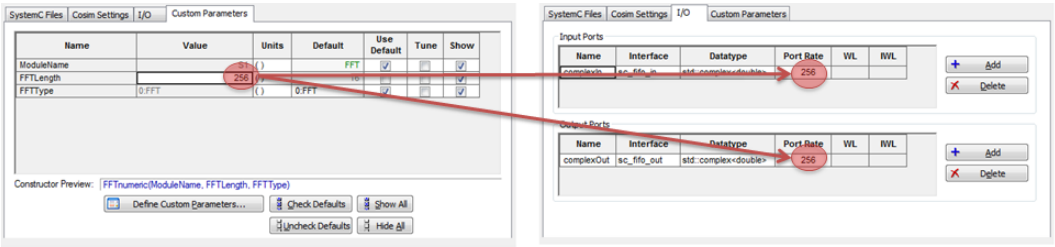
### Numeric Model with sc_fifo Ports

Inside the *FFT numeric design* schematic, there is a SystemCCosim block called *FFT* . This block represents a SystemC FFT model (FFT.h and FFT.cpp in folder "source\fft\fft_ifft") which has a constructor parameter that determines the size of the FFT and two *sc_fifo* ports, one for input and one for output. The behavior of the SystemC model is described as follows:

When the number of available samples in the input FIFO port is equal to the size of the FFT (which is defined by the constructor parameter), the model computes the FFT of the input samples and writes the resulting samples to the output FIFO port. The number of output samples in one computation is equal to the FFT size.

Since the data flow semantics requires that in each firing, a SystemCCosim model consumes and produces the numbers of samples as defined by the corresponding Port Rate properties, therefore, the Port Rate of the input and output FFT *sc_fifo* ports should be set to the value of the "FFTLength" parameter that defines the size of the FFT.

In the current design, we have chosen the FFT size to be 256 and consequently, Port Rate values for both ports have been set to the same value, as shown in the screenshot below. The user can update this number to another value (power of 2) for different computation of the FFT.



| Name | Value | Units | Default | Use Default | Tune | Show |
|---|---|---|---|---|---|---|
| ModuleName | S1 | ( ) | FFT | ☑ | ☐ | ☑ |
| FFTLength | 256 | ( ) | 16 | ☐ | ☐ | ☑ |
| FFTType | 0:FFT | ( ) | 0:FFT | ☑ | ☐ | ☑ |

Constructor Preview: FFTnumeric(ModuleName, FFTLength, FFTType)

Input Ports

| Name | Interface | Datatype | Port Rate | WL | IWL |
|---|---|---|---|---|---|
| complexIn | sc_fifo_in | std::complex<double> | 256 | | |

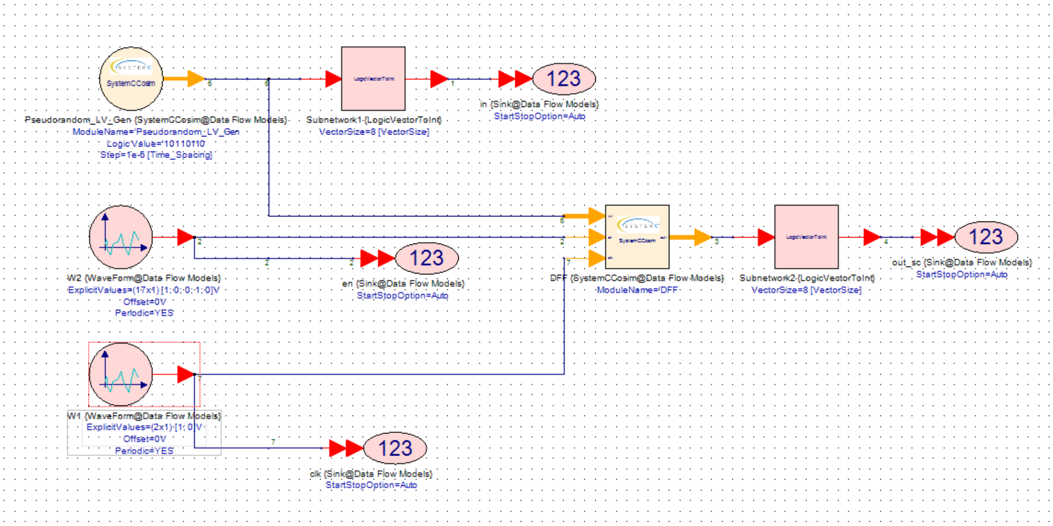
Output Ports

| Name | Interface | Datatype | Port Rate | WL | IWL |
|---|---|---|---|---|---|
| complexOut | sc_fifo_out | std::complex<double> | 256 | | |

The behavior of the SystemC FFT model that was described above matches the one of the numeric built-in FFT model. For this reason, in the same design, a built-in FFT model is used in parallel to the SystemC one, so that user can verify the equal behavior.

## Sequential Logic Design

In the same workspace, the design named *DFF* in folder *Sequential* contains a *DFF* SystemCCosim block representing a SystemC implementation of a D-Flip-Flop Register (DFF). The SystemC DFF model, which can be found in "source\dff\dff.h", uses only *sc_signal* for its inputs and outputs. The input ports are a *clock (clk)* signal, an *enable (en)* signal and a *logic vector (in1)*. The output is a *logic vector (out1)*.



Pseudorandom_LV_Gen {SystemCCosim@Data Flow Models}
ModuleName="Pseudorandom_LV_Gen
LogicValue="10110110
Step=1e-6 [Time_Spacing]

Subnetwork1 {LogicVectorToInt}
VectorSize=8 [VectorSize]

in {Sink@Data Flow Models}
StartStopOption=Auto

W2 {WaveForm@Data Flow Models}
ExplicitValues=(17x1) [1;-0;-0;-1;-0]V
Offset=0V
Periodic=YES

en {Sink@Data Flow Models}
StartStopOption=Auto

DFF {SystemCCosim@Data Flow Models}
ModuleName="DFF

Subnetwork2 {LogicVectorToInt}
VectorSize=8 [VectorSize]

out_sc {Sink@Data Flow Models}
StartStopOption=Auto

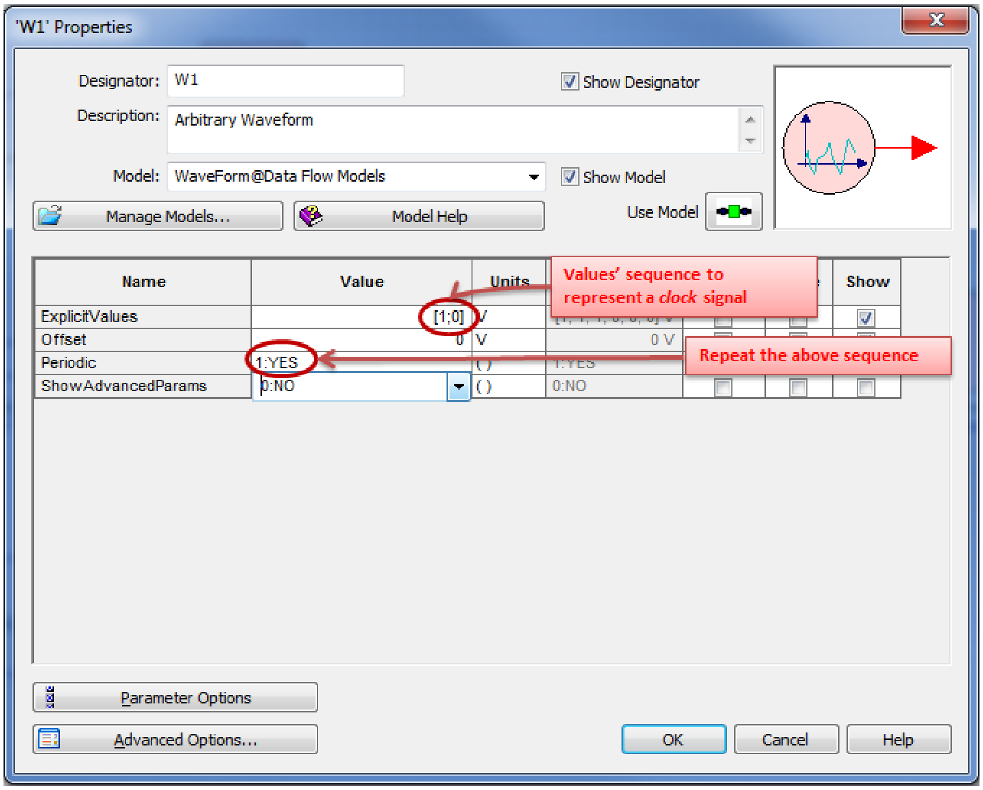
W1 {WaveForm@Data Flow Models}
ExplicitValues=(2x1) [1; 0]V
Offset=0V
Periodic=YES

clk {Sink@Data Flow Models}
StartStopOption=Auto

To generate digital signals for the DFF SystemCCosim model, we use WaveForm generators.

For the *clock* signal:

1. We set the ExplicitValues to 1 and 0 (to represent one clock cycle).
2. We set the above sequence to be periodic.



'W1' Properties

Designator: W1 ☑ Show Designator

Description: Arbitrary Waveform

Model: WaveForm@Data Flow Models ☑ Show Model

Use Model

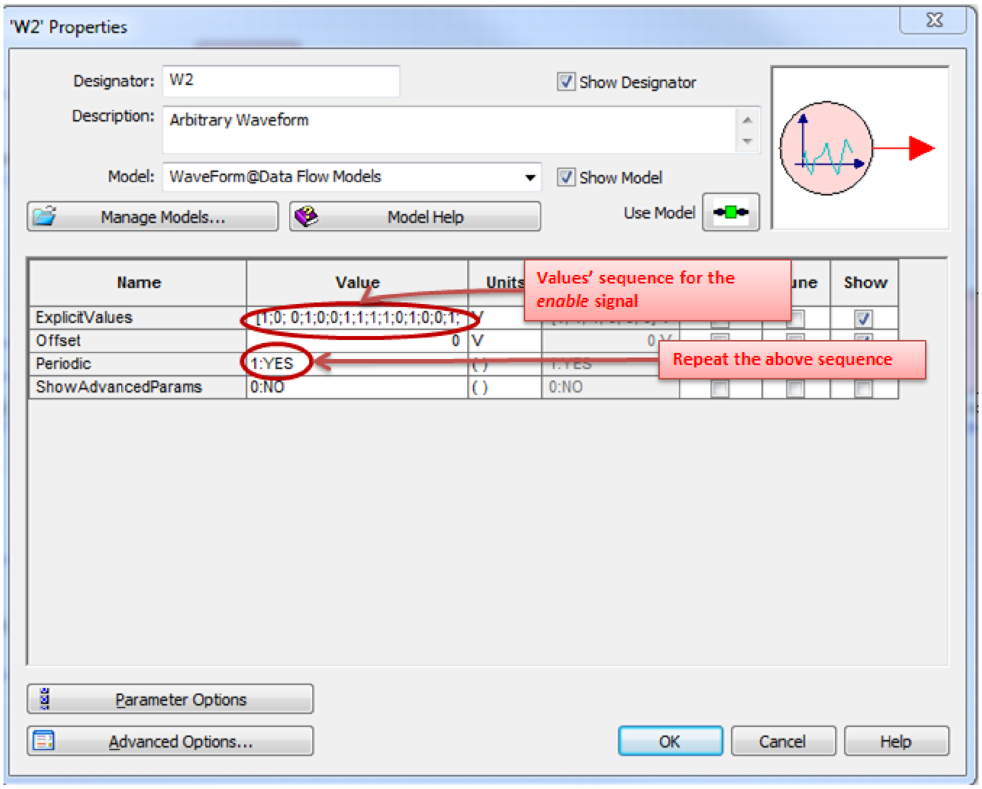
| Name | Value | Units | | | | Show |
|---|---|---|---|---|---|---|
| ExplicitValues | [1;0] | V | [illegible] | ☐ | ☐ | ☑ |
| Offset | 0 | V | 0 V | ☐ | ☐ | ☑ |
| Periodic | 1:YES | ( ) | 1:YES | | | |
| ShowAdvancedParams | 0:NO | ( ) | 0:NO | ☐ | ☐ | ☐ |

Values' sequence to represent a *clock* signal

Repeat the above sequence

Similarly, for the *enable* signal:

1. We set the ExplicitValues to the sequence as shown in the following screenshot.
2. We set the above sequence to be periodic.



'W2' Properties

Designator: W2 ☑ Show Designator

Description: Arbitrary Waveform

Model: WaveForm@Data Flow Models ☑ Show Model

Use Model

| Name | Value | Units | | | Tune | Show |
|---|---|---|---|---|---|---|
| ExplicitValues | [1;0; 0;1;0;0;1;1;1;1;0;1;0;0;1; | V | [illegible] | ☐ | ☐ | ☑ |
| Offset | 0 | V | 0 V | ☐ | ☐ | ☑ |
| Periodic | 1:YES | ( ) | 1:YES | | | |
| ShowAdvancedParams | 0:NO | ( ) | 0:NO | ☐ | ☐ | ☐ |

Values' sequence for the *enable* signal

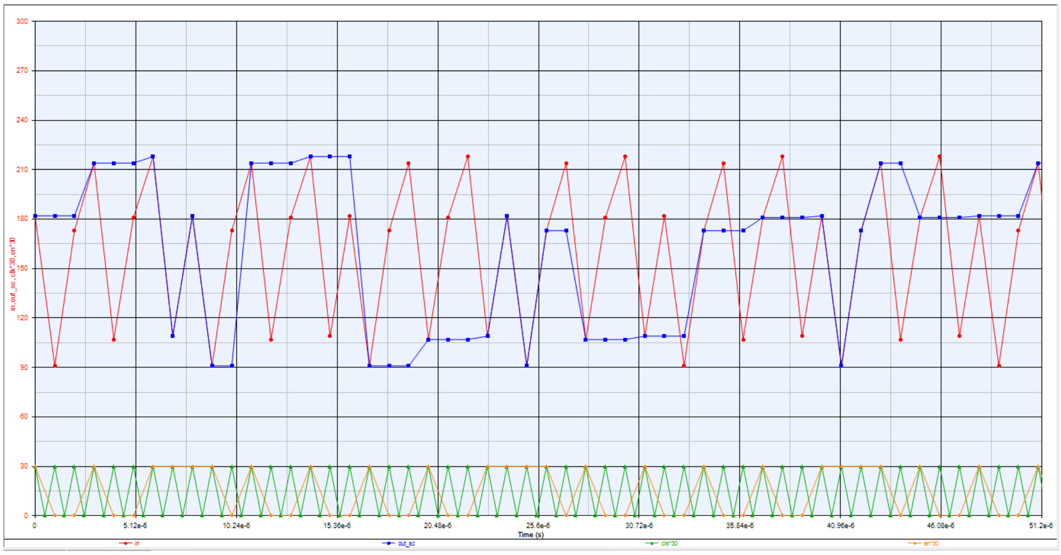
Repeat the above sequence

Finally, for the *Logic Vector* input *in1*, we use another SystemCCosim block "Pseudorandom_LV_Gen" to generate the pseudo random logic sequence. This block uses the parameter "Logic Value" to specify the initial logic value and then generates pseudo-random sequences every time step, which is defined by the parameter_Step_. In the example, we set the value of the *Step* parameter to 1us.

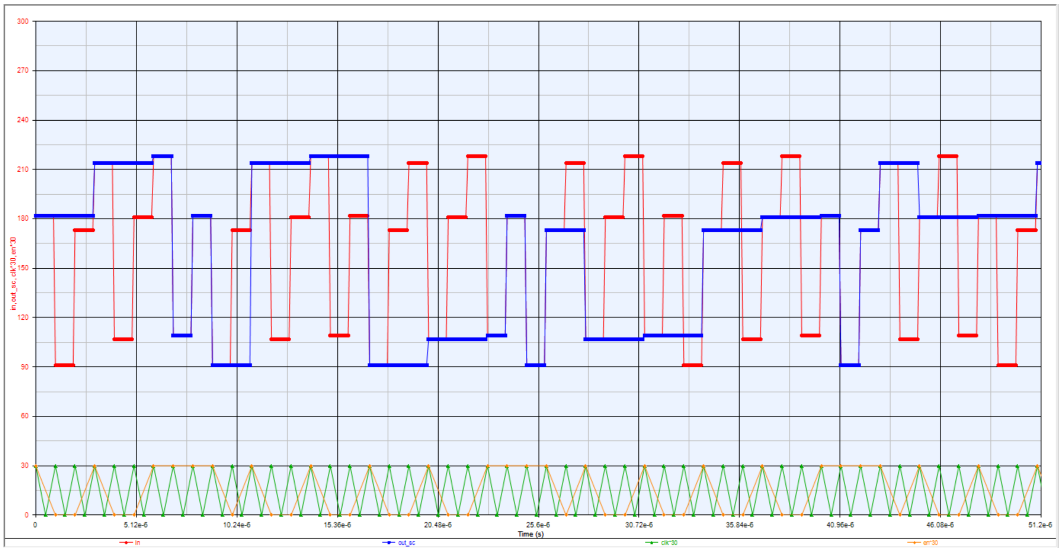
In this example, DFF is a digital synchronous component, thus, the state of system's ports are important only on clock events (positive clock edge in this case). In order to model correctly this functionality in a data-flow environment, it is important to sample the values of the *sc_signal* ports at the time instances of positive clock edges. To achieve this, the Port Rate of the enable (*en*), input (*in1*), and output (*out1*) signals of the *DFF* block should be a multiple of the Port Rate of the clock signal. Thus, if the clock rate is 2, the rate of the rest signals should be 2, 4, 8, 10, 12, ... etc. This way it is ensured that the signals will be "sampled" at the clock edges. In order to "sample" the signals only at the positive clock edges, the Port Rate of the clock (*clk*) port should be doubled (because two samples represent one clock period). Thus, if a Port Rate of the signals (*en*, *in1*, *out1*) is 1, the Port Rate of the *clk* port should be 2. The condition described above holds for the following port rate specification:

- *clk* --> 2
- *en* --> 1
- *in1* --> 1
- *out1* --> 1

However, the user is encouraged to use different port rate values that follow the aforementioned condition (e.g. 12 for the *clk*, 6 for the *en* and 6 for the *in1* and *out1* ports) and notice that the resulting graph stays unmodified. The only difference to the previous port rate specification is that, in the latter case, one firing of the "DFF" SystemCCosim Block will process 6 input samples. The resulting graph for both cases is shown in the figure below:



As a variation of the above case, increased Port Rate of *in1* and *out1* ports can be chosen. This way an oversampling of the values of these ports can be achieved and thus, a more accurate waveform of these signals can be rendered. Note that port rate of *in1* and *out1* should still be an integer multiple of the *en* signal. The figure below shows this variation when we apply port rate 12 for the *clk* port, 6 for the *en* and 60 for the *in1* and *out1*:



In both figures before, the curves of the *in1* ( red color) and *out1* ( blue color) signals have the same response in time, however in the latter picture, the signals are much more accurate due to the increased port rate values of the ports. This increased accuracy reveals also the Digital nature of the signals as a quantization of values can be noticed.

### Example 5: Clock signals

SystemVue is capable of automatically generating one SystemC clock signal ( *sc_clock*) inside the SystemC simulation and binding it to a user-defined clock input port. This option is only applicable to SystemC input ports of type:

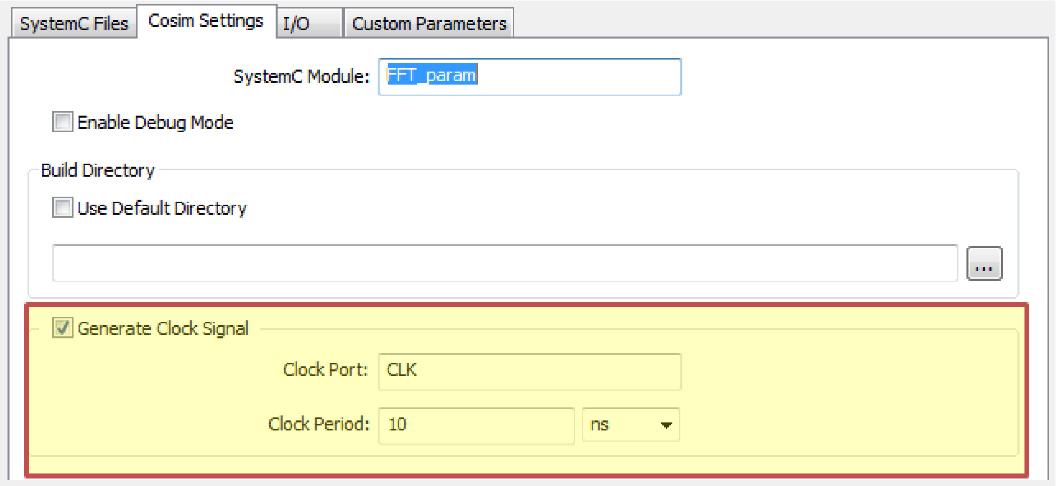
- *sc_clk_in*, or
- *sc_in<bool>*

To connect an input port to a sc_clock, user has to follow these simple steps:

1. Go to the Cosim Settings Tab of the SystemCCosim custom UI.
2. Check the Generate Clock Signal checkbox.
3. Insert the name of the port in the SystemC model to be bound to the clock signal in the Clock Port text field.
4. Set clock period using the Clock Period text field combined with the time unit field.

WARNING If you check the **Generate Clock Signal** checkbox, do **not** declare this port inside the *I/O* tab.

One of the examples that use this feature is the *FFT design* schematic in folder *FFT_parameters_passing* of t he SystemC_Parameters_Passing.wsv w orkspace.



SystemC Module: FFT_param

☐ Enable Debug Mode

Build Directory

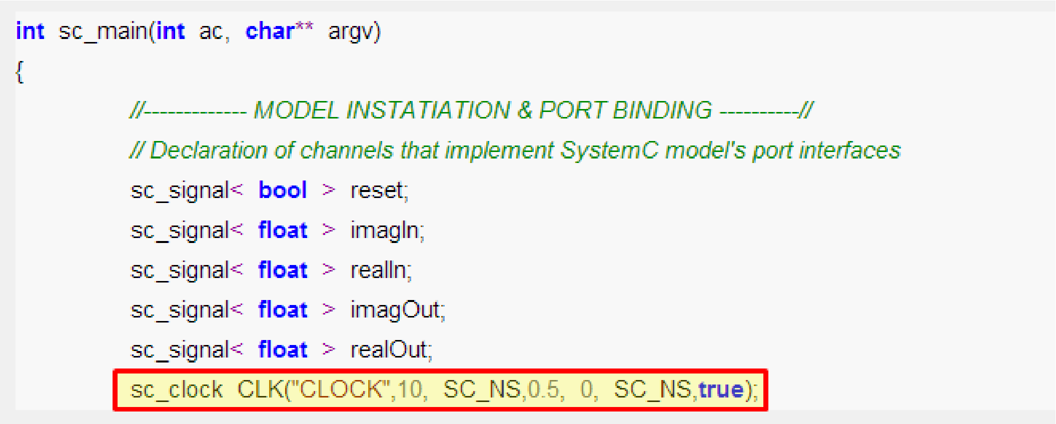
☐ Use Default Directory

☑ Generate Clock Signal

Clock Port: CLK

Clock Period: 10 ns

The automatically generated sc_main function for cosimulation will look like this:



```
int  sc_main(int  ac,  char**  argv)
{
            //------------- MODEL INSTATIATION & PORT BINDING ----------//
            // Declaration of channels that implement SystemC model's port interfaces
            sc_signal<  bool  >  reset;
            sc_signal<  float  >  imagIn;
            sc_signal<  float  >  realIn;
            sc_signal<  float  >  imagOut;
            sc_signal<  float  >  realOut;
            sc_clock  CLK("CLOCK",10,  SC_NS,0.5,  0,  SC_NS,true);
```

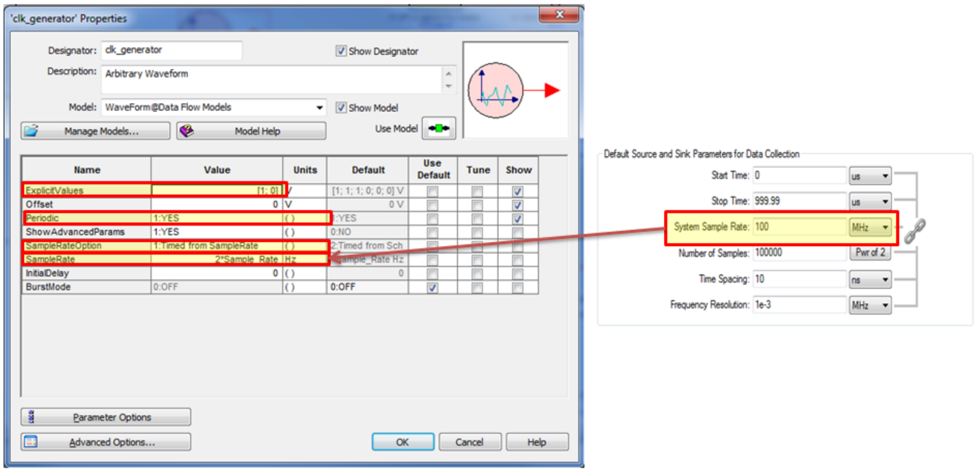
Here, a clock signal with a period of 10 *ns*, starting from *high* value, having half period *duty cycle* and without any starting delay is generated.

If the user wants to modify any of the aforementioned characteristics, he can do this by using the *Debug Mode* feature. The definition of the clock signal is taking place inside the *sc_main* function.

An alternative is to create a clock signal externally inside SystemVue. One way to do that is to use the WaveForm model (see Sequential Logic Design example). To create the same clock signal externally, for the design above, the users can follow the steps below:

1. Declare the clock port as a normal input port (of type sc_in<bool>) in the I/O Tab.

2. Add a WaveForm model in the design and configure it to represent a clock signal. In order to have a clock period of 10 ns, the clock frequency should be 100 MHz. To create one complete clock period, the WaveForm model should execute twice to generate 1 and 0 states, thus, the *SampleRate* parameter should be set to twice of the sampling rate of the other signal sources.



'clk_generator' Properties

Designator: clk_generator ☑ Show Designator

Description: Arbitrary Waveform

Model: WaveForm@Data Flow Models ☑ Show Model

| Name | Value | Units | Default | Use Default | Tune | Show |
|---|---|---|---|---|---|---|
| ExplicitValues | [1; 0] | V | [1; 1; 1; 0; 0; 0] V | □ | □ | ☑ |
| Offset | 0 | V | 0 V | □ | □ | ☑ |
| Periodic | 1:YES | ( ) | 1:YES | □ | □ | ☑ |
| ShowAdvancedParams | 1:YES | ( ) | 0:NO | □ | □ | □ |
| SampleRateOption | 1:Timed from SampleRate | ( ) | 2:Timed from Sch | □ | □ | □ |
| SampleRate | 2*Sample_Rate | Hz | Sample_Rate Hz | □ | □ | □ |
| InitialDelay | 0 | ( ) | 0 | □ | □ | □ |
| BurstMode | 0:OFF | ( ) | 0:OFF | ☑ | □ | □ |

Default Source and Sink Parameters for Data Collection

Start Time: 0 us

Stop Time: 999.99 us

System Sample Rate: 100 MHz

Number of Samples: 100000

Time Spacing: 10 ns

Frequency Resolution: 1e-3 MHz

3. Finally, to have both 1 and 0 states of the clock signal to be captured, the Port Rate of the clock port in the I/O Tab can be set to twice of the other ports, in general.

Users need to generate external clock signals manually if the following two cases happen:

- The SystemC clock input port is of type different than the ones mentioned at the beginning of this subsection (e.g. case of *sc_in<sc_logic>* type)
- The SystemC model contains more than one input clock ports. One can be auto-generated but the rest should be created externally.

# Measurement Automation

## Measurement Automation

- Getting Started with Measurement Automation
- Using Command Expert to Create Custom Instrument Links
- Using Command Expert In MATLAB Script
- Using Waveform Sequencer Composer

## Getting Started with Measurement Automation

Prerequisites

- Installation of Keysight IO Library.
- Installation of Keysight Command Expert
- (Optionally for IVI-COM Exploration) Installation of IVI-COM drivers through Keysight Technical Support for Test & Measurement web (Go to Drivers, Updates & Examples and look into IVI under By Class)

### Introduction

There are two mechanisms to control and communicate with instruments:

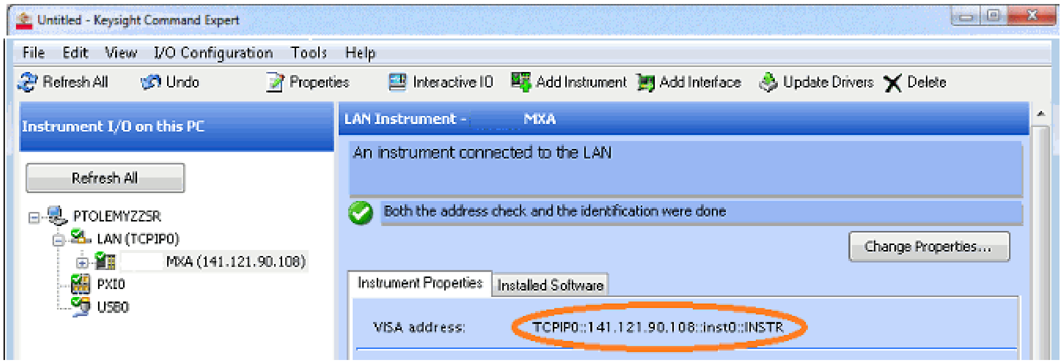
- The primary mechanism is through Keysight Command Expert. In schematic, use CommandExpertLink part, while in MATLAB Script Equations environment, use the agRunSequence function.
    - Refer to Using Command Expert to Create Custom Instrument Links for details on using CommandExpertLink part.
    - Refer to Using Command Expert In MATLAB Script for details on using Keysight Command Expert in MATLAB Script Equations environment.
- The secondary mechanism is only for instruments that are LXI compliant, in which case the tcpip function in MATLAB Script can be used to communicate with instruments.
    - Refer to Use Tcpip to Control Instrument in MATLAB Script Equations for an example.

NOTE For applications that need to **manage the sequence/order of instrument control and running the simulation**, e.g. to adjust a DC bias on a device before running a simulation that requires data measured over that device, refer to Using MATLAB Script For Sequence Control.

### Before You Start - Instrument VISA Address

It is highly recommended to use complete VISA address for the instrument in Keysight Command Expert.
Th easiest way to find an instrument's VISA address is through Keysight Connection

Expert (which is automatically installed with Keysight IO Library).
The screen capture below indicates where in Keysight Connection Expert to find an instrument's VISA address after the instrument is registered/added into it.

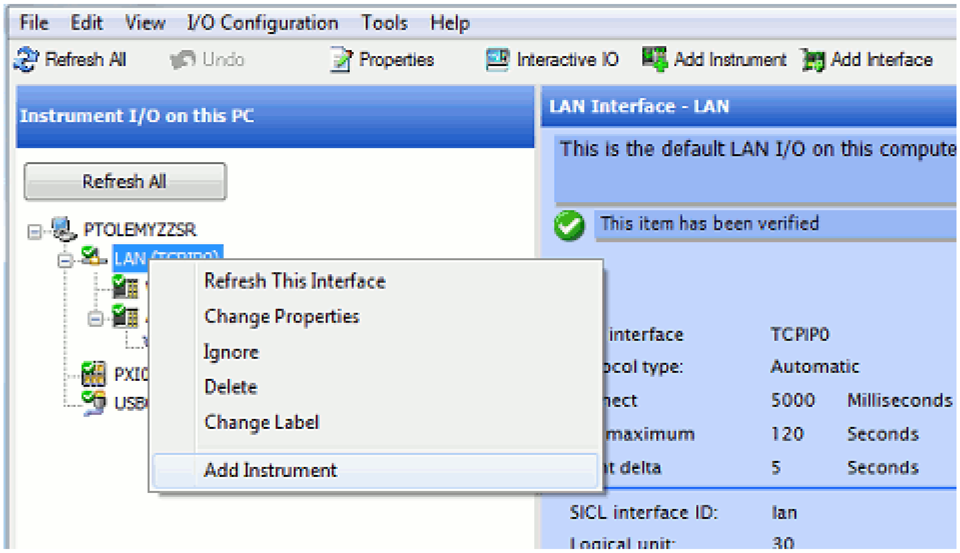


An instrument connected to the LAN

Both the address check and the identification were done

VISA address: TCPIP0::141.121.90.108::inst0::INSTR

The screen capture below demonstrates how to register/add an LXI instrument into Keysight Connection Expert. Simply right-click on LAN (TCPIP0) and select Add Instrument and fill in the IP information on the popped up dialog.



This is the default LAN I/O on this compute

This item has been verified

interface TCPIP0
ocol type: Automatic
nect 5000 Milliseconds
maximum 120 Seconds
t delta 5 Seconds
SICL interface ID: lan
Logical unit: 30

## A Brief Introduction to Keysight Command Expert Software

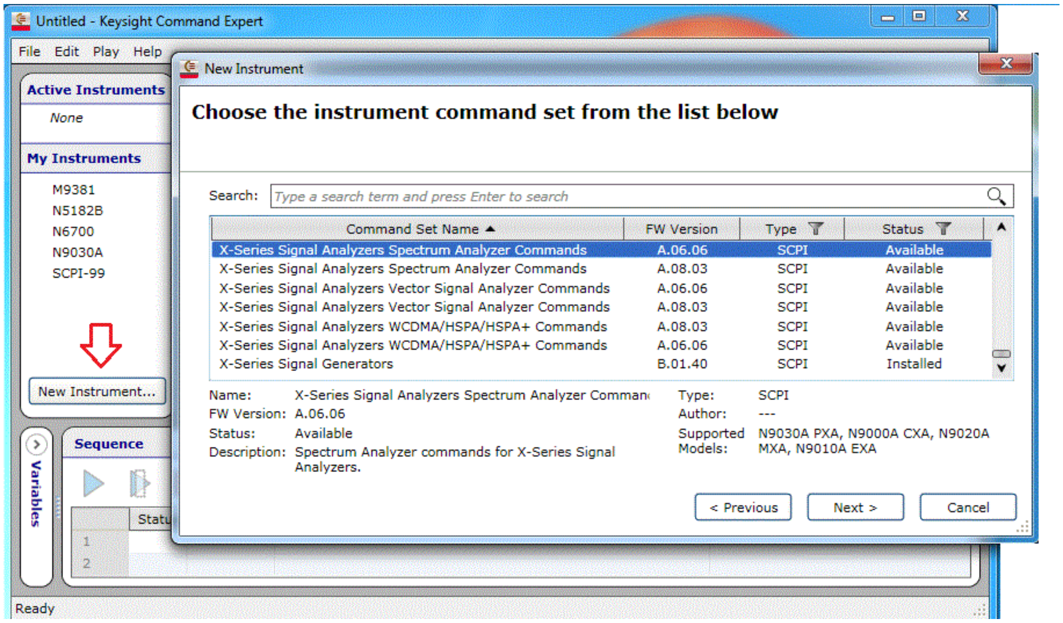
Keysight Command Expert is a FREE software suite that makes it easy to control instruments and retrieve measurement data either through SCPI commands or IVI-COM drivers.

NOTE Refer to the Help manual of **Keysight Command Expert** software for more details

CAUTION Install Keysight Command Expert on your PC and **start it directly** and go over the following materials.

## Connect to an instrument

1. As shown in the screen capture, click New Instrument... to bring up New Instrument dialog to add a new instrument.
   a. Make sure to fill in the instrument address
   b. It is convenient to select the command set for the instrument too.
2. Click the Connect button to connect to the instrument.



Choose the instrument command set from the list below

| Command Set Name | FW Version | Type | Status |
|---|---|---|---|
| X-Series Signal Analyzers Spectrum Analyzer Commands | A.06.06 | SCPI | Available |
| X-Series Signal Analyzers Spectrum Analyzer Commands | A.08.03 | SCPI | Available |
| X-Series Signal Analyzers Vector Signal Analyzer Commands | A.06.06 | SCPI | Available |
| X-Series Signal Analyzers Vector Signal Analyzer Commands | A.08.03 | SCPI | Available |
| X-Series Signal Analyzers WCDMA/HSPA/HSPA+ Commands | A.08.03 | SCPI | Available |
| X-Series Signal Analyzers WCDMA/HSPA/HSPA+ Commands | A.06.06 | SCPI | Available |
| X-Series Signal Generators | B.01.40 | SCPI | Installed |

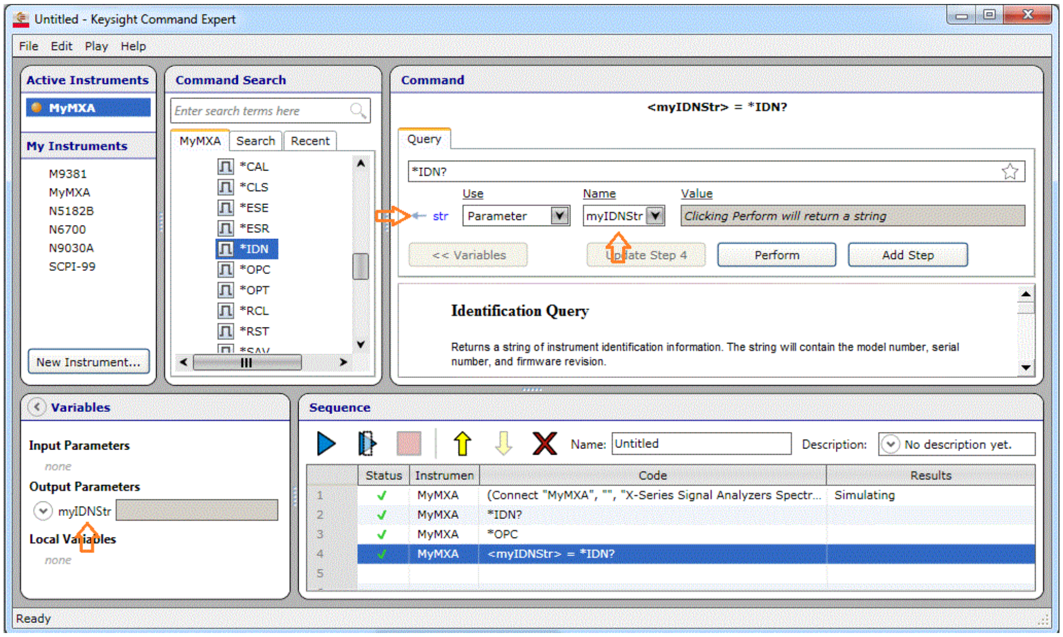
Name: X-Series Signal Analyzers Spectrum Analyzer Comman
FW Version: A.06.06
Status: Available
Description: Spectrum Analyzer commands for X-Series Signal Analyzers.

Type: SCPI
Author: ---
Supported Models: N9030A PXA, N9000A CXA, N9020A MXA, N9010A EXA

## Use SCPI Commands

1. Make sure the correct SCPI command set is chosen.
2. Browse and select the SCPI command and click Perform to send the command to the instrument.

   - Alternatively, you can Add Step instead Perform and then execute all steps continuously by clicking the Play Sequence ▶ .
   - You can also use parameters with the SCPI commands and use break points (as shown as the red dot next to the line number).



<myIDNStr> = *IDN?

*IDN?

str

| Use | Name | Value |
|---|---|---|
| Parameter | myIDNStr | Clicking Perform will return a string |

**Identification Query**

Returns a string of instrument identification information. The string will contain the model number, serial number, and firmware revision.

Input Parameters

none

Output Parameters

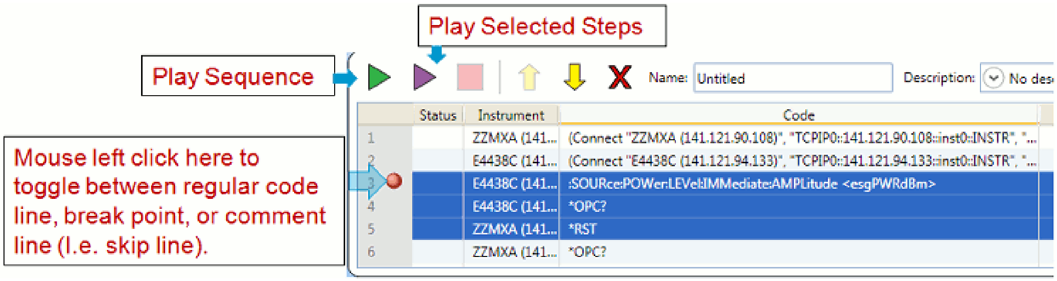
myIDNStr

Local Variables

none

Name: Untitled Description: No description yet.

|  | Status | Instrumen | Code | Results |
|---|---|---|---|---|
| 1 | ✓ | MyMXA | (Connect "MyMXA", "", "X-Series Signal Analyzers Spectr... | Simulating |
| 2 | ✓ | MyMXA | *IDN? |  |
| 3 | ✓ | MyMXA | *OPC |  |
| 4 | ✓ | MyMXA | <myIDNStr> = *IDN? |  |
| 5 |  |  |  |  |



Play Selected Steps

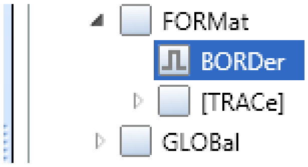
Play Sequence

Mouse left click here to toggle between regular code line, break point, or comment line (I.e. skip line).

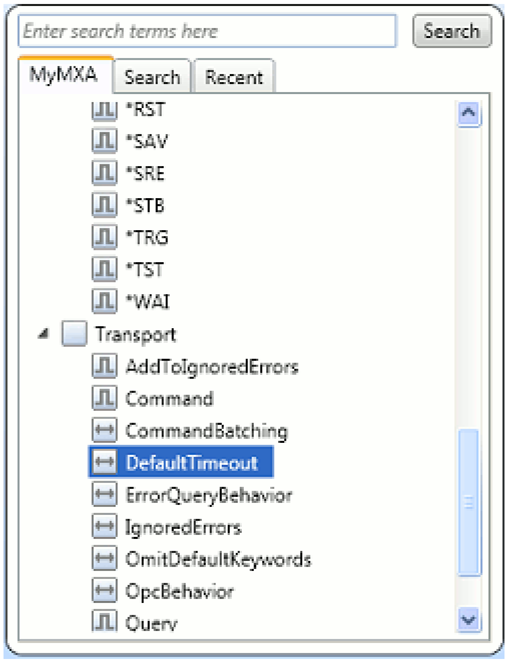
Name: Untitled Description: No des

|  | Status | Instrument | Code |
|---|---|---|---|
| 1 |  | ZZMXA (141... | (Connect "ZZMXA (141.121.90.108)", "TCPIP0::141.121.90.108::inst0::INSTR", "... |
| 2 |  | E4438C (141... | (Connect "E4438C (141.121.94.133)", "TCPIP0::141.121.94.133::inst0::INSTR", "... |
| 3 | ● | E4438C (141... | :SOURce:POWer:LEVel:IMMediate:AMPLitude <esgPWRdBm> |
| 4 |  | E4438C (141... | *OPC? |
| 5 |  | ZZMXA (141... | *RST |
| 6 |  | ZZMXA (141... | *OPC? |

**NOTE** When downloading or uploading data, you may encounter **Big-Endian/Small-Endian** issue (i.e. data byte order in the instrument is opposite to data byte order on your computer) and look for **FORMat.BORDer** command to resolve it.



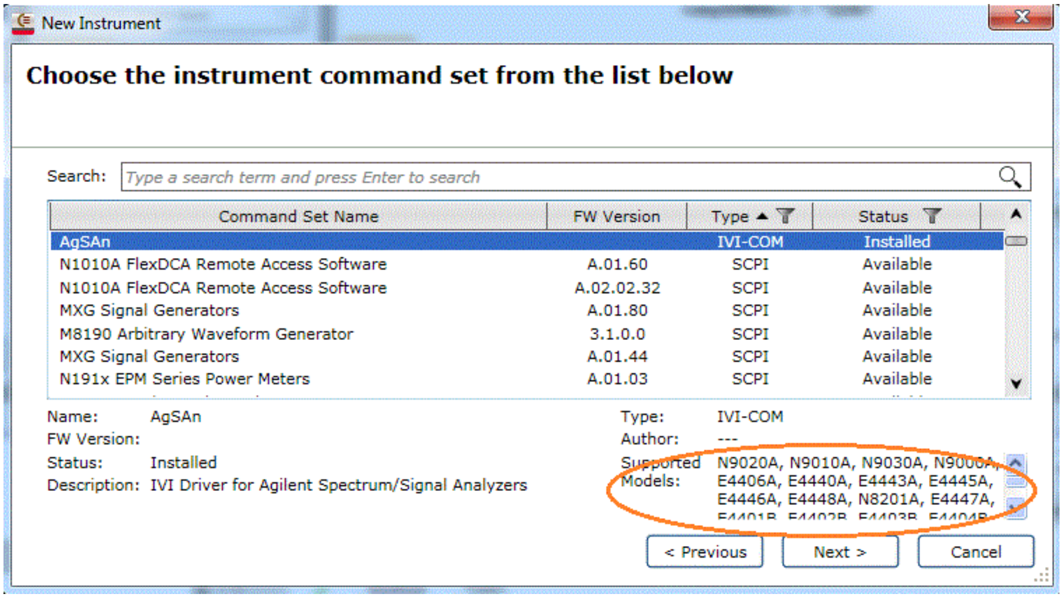
**NOTE** When downloading or uploading a large amount of data, consider increasing the communication timeout limit through the **DefaultTimeout** command.



## Use IVI-COM Driver

**CAUTION** **Some IVI-COM driver only supports 32-bit operating system.** In such cases, when running 64 bit SystemVue, you will see an error message `Unable to load command set: ...` in SystemVue error log, which indicates you have to use 32 bit SystemVue with this specific IVI-COM driver.

1. Use the complete VISA address for the instrument.
2. Make sure you select the IVI-COM driver targeted for your specific instrument.
3. Browse and select IVI-COM functions to control the instruments. Here is an Example that would search peak power frequency point between 1.975 GHz and 2.025 GHz. Download it and use Command Expert menu File -> Open Sequence to open it and add the VISA address of your MXA, then run it. (Ideally, you would have an RF synthesizer to provide a CW signal within this frequency range, but that's not necessary for this exercise).



Choose the instrument command set from the list below

Search: Type a search term and press Enter to search

| Command Set Name | FW Version | Type | Status |
|---|---|---|---|
| AgSAn | | IVI-COM | Installed |
| N1010A FlexDCA Remote Access Software | A.01.60 | SCPI | Available |
| N1010A FlexDCA Remote Access Software | A.02.02.32 | SCPI | Available |
| MXG Signal Generators | A.01.80 | SCPI | Available |
| M8190 Arbitrary Waveform Generator | 3.1.0.0 | SCPI | Available |
| MXG Signal Generators | A.01.44 | SCPI | Available |
| N191x EPM Series Power Meters | A.01.03 | SCPI | Available |

Name: AgSAn
FW Version:
Status: Installed
Description: IVI Driver for Agilent Spectrum/Signal Analyzers

Type: IVI-COM
Author: ---
Supported Models: N9020A, N9010A, N9030A, N9000A, E4406A, E4440A, E4443A, E4445A, E4446A, E4448A, N8201A, E4447A, E4401B, E4402B, E4403B, E4404B

## Use Tcpip to Control Instrument in MATLAB Script Equations

For instruments that are LXI compliant, you can use MATLAB Script tcpip function in Equations to directly send SCPI commands to instruments.

### A Simple Example

```
% Create tcpip communication with RF source at IP
address 111.222.333.444 using port 5025
rfSource = tcpip('111.222.333.444', 5025);
fopen(rfSource);

% Set the power level to -10 dBm
fprintf(rfSource, ':POW -10');
% make sure the power level is settled by checking *OPC
fprintf(rfSource, '*OPC?');
statusRes = fgets(rfSource);

% Close the communication channel once done
fclose(rfSource)
```

**NOTE** Keysight instruments typically use port **5025** for Tcpip communication. Refer the instrument manual on how to find or configure IP address.

### Notes and Links

- How do I get instrument SCPI Command Sets and IVI-COM drivers?
    - Once you select an instrument SCPI command set in Keysight Command Expert, it will be automatically downloaded and installed on your computer if it has not been installed.

    - You can find IVI-COM drivers through Keysight Technical Support for Test & Measurement web. Go to Drivers, Updates & Examples tab and select IVI class drivers.
- Download Keysight Command Expert software
- Download Keysight IO Suite.

## Using Command Expert to Create Custom Instrument Links

Prerequisites

- Installation of Keysight IO Library.
- Installation of Keysight Command Expert
- (Optionally for IVI-COM Exploration) Installation of IVI-COM drivers through Keysight Technical Support for Test & Measurement web (Go to Drivers, Updates & Examples and look into IVI under By Class)

Associated tutorial workspace

- Examples\Instruments\CommandExpert\CommandExpertRFSources.wsv for configuring Sink mode
- Examples\Instruments\CommandExpert\CommandExpert_LTE_FDD_UL_Throu wsv for configuring Source mode.

**CAUTION** We recommend you to copy the complete directory of **Examples\Instruments\CommandExpert** to your local directory to avoid file writing permission issues on default installation directory.

### Introduction

- CommandExpertLink part is the component used in schematic to interface to Keysight Command Expert
- CommandExpertLink part is designed to have two Link Type: Sink type and Sourcetype.
    - Sink linkage is to download simulation data into instruments. The most common usage is to download baseband I/Q waveform data into arbitrary waveform generators or RF signal sources.
    - Source linkage is to upload data from instruments for the simulation to analyze or process (e.g. to perform digital demodulation). The most common usage is to upload measurement data captured by signal analyzers or oscilloscopes.
    - The following sections cover details on how to use CommandExpertLink part.

- The challenge in using CommandExpertLink part is actually on understanding the command set so as to find the correct SCPI command or IVI-COM function for the desired result. Contact Keysight support team for your specific instrument to help you understand the instrument's command set.

### Before You Start

- It is always a good idea to explore your instrument with Keysight Command Expert before you start working on using CommandExpertLink part in SystemVue to control the instrument.
- Refer to A Brief Introduction to Keysight Command Expert Software for a quick review on how to use Keysight Command Expert.
- Once you can communicate with a specific instrument in Keysight Command Expert and if the instrument is the same model as what is used in a SystemVue's example, you can just replace the instrument's VISA address in the correspondent CommandExpertLink part's Address filed (as highlighted in the screen capture below) and be able to run the simulation.
- To summarize, these are the recommended steps to using CommandExpertLink part:
  Step 1. Explore your instrument in Keysight Command Expert using simple commands such as identity query. If you want to use the instrument in an existing example, make sure select the same SCPI Command Set or IVI-COM driver as what is used in the example.
  Step 2. Use instrument in CommandExpertLink part in SystemVue examples or configure a new CommandExpertLink part (which will be described in the following sections).

**NOTE** It is highly recommended that you have the corresponding example open when you go over the materials on how to configure CommandExpertLink part for each Link Type's targeted application.

### A Brief Introduction to CommandExpertLink Part's Dialog

For details, please refer to CommandExpertLink part.
Even though it supports instrument Offline mode, we highly recommend you to explore with actual instrument online.
The general steps to set up a CommandExpertLink part are:

1. Select one or more instrument(s) and appropriate command set for each instrument.

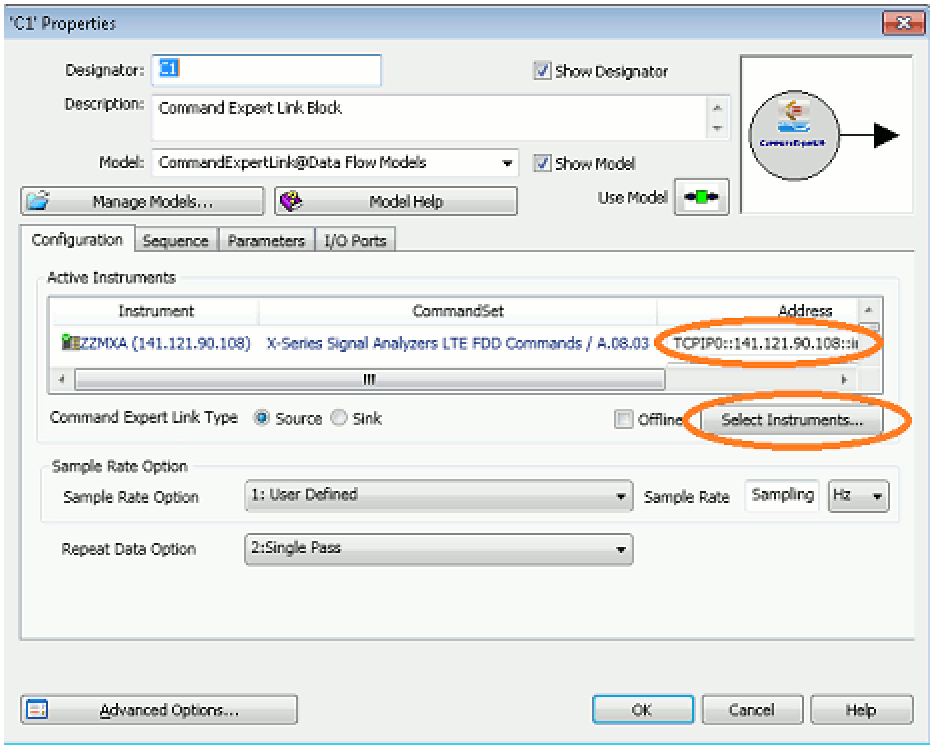
   **NOTE**
   - If you use an existing workspace, you can type (or copy /paste) in the VISA address of the instrument.
   - To select an instrument, make sure that you have registered it in Keysight Connection Expert.

2. Select appropriate Link Type for the instrument.
3. Create instrument control Sequence through Keysight Command Expert that is invoked by the Edit button in Sequence page.
4. Set up Parameters and I/O Ports

NOTE

- The parameters configured inside **Command Expert** during **Sequence** creation are directly reflected in the **Parameters** page with one exception for **Source**linkage.
  - For **Source** linkage, array parameter(s) in **Run** sequence block (as described later) will be automatically treated as output port(s).
- For **Sink** linkage, you have to select one or more parameter (s) in **Parameter** page and designate it/them as input port (s). More details on this later.

CAUTION **I/O Ports** are created dynamically based on the **Link Type** and the **Sequence** code. That's why CommandExpertLink part does **not** have any I/O pins when initially dropped onto a schematic.



'C1' Properties

Designator: C1

☑ Show Designator

Description: Command Expert Link Block

Model: CommandExpertLink@Data Flow Models

☑ Show Model

Active Instruments

| Instrument | CommandSet | Address |
|---|---|---|
| ZZMXA (141.121.90.108) | X-Series Signal Analyzers LTE FDD Commands / A.08.03 | TCPIP0::141.121.90.108::i |

Command Expert Link Type ● Source ○ Sink

□ Offline

Sample Rate Option

Sample Rate Option: 1: User Defined

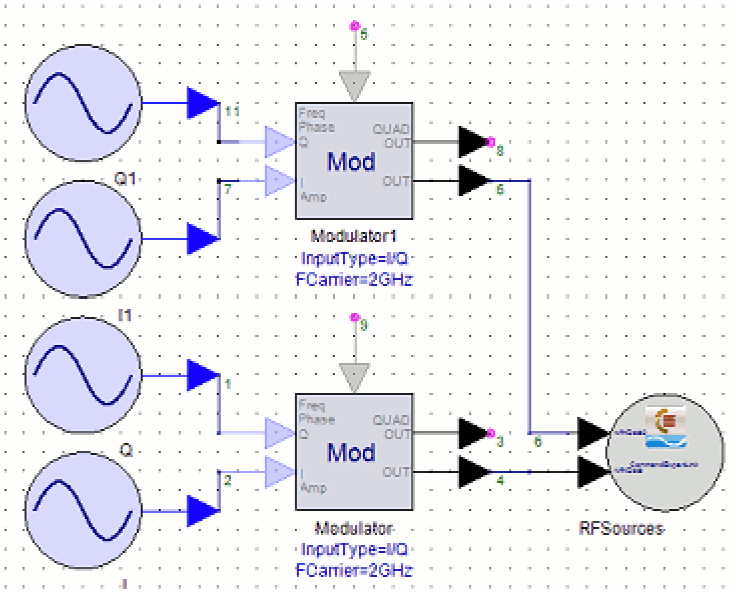
Sample Rate: Sampling Hz

Repeat Data Option: 2:Single Pass

## Steps to Configure CommandExpertLink Part to Download Simulation Data into Instrument(s)

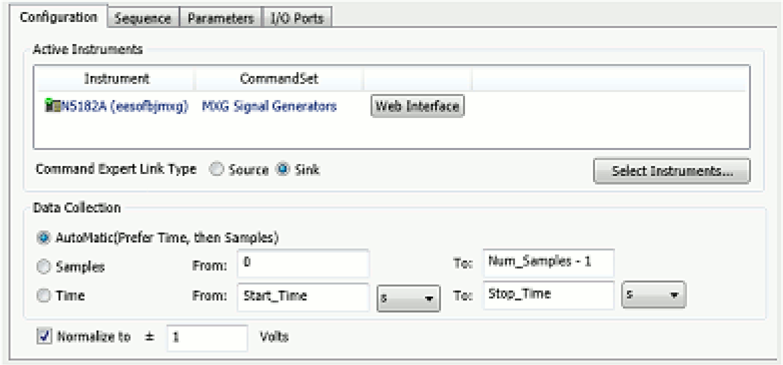
Use the **CommandExpertRFSources.wsv** example in examples archive under **Instruments\CommandExpert* directory to help you with your exploration.**

**This example is designed to *generate two modulated RF signals** using Agilent RF synthesizers such as MXG or ESG . It can be easily extended for 2x2 MIMO applications with additional SCPI commands for synchronization control across the two RF sources.



## I. Configure the Linkage Type

Typically, the instrument requires to normalize simulation generated waveform data within 1.0 Volt peak-to-peak. Refer to the instrument manual for the actual limit.



Active Instruments

| Instrument | CommandSet | |
|---|---|---|
| N5182A (eesofbjmxg) | MXG Signal Generators | Web Interface |

Command Expert Link Type ○ Source ● Sink

Data Collection

● AutoMatic(Prefer Time, then Samples)

○ Samples From: 0 To: Num_Samples - 1

○ Time From: Start_Time s To: Stop_Time s

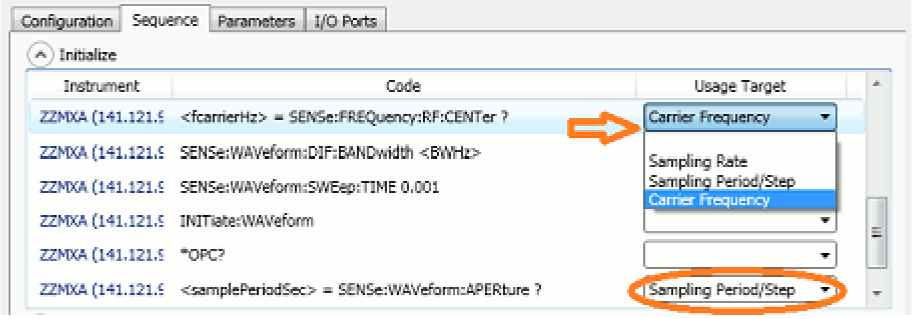
☑ Normalize to ± 1 Volts

## II. Create Sequence Code

- In Initialize block: All of the instrument initialization/configuration goes here. In particular:
    - Consider providing an SCPI command (or IVI-COM function) that configures sampling rate or sampling period (also known as aperture) of the instrument and notify CommandExpertLink part by setting the Usage Target for its parameter accordingly.
    - for RF signal, consider providing an SCPI command (or IVI-COM function) that configures carrier frequency of the instrument and notify CommandExpertLink part by setting the Usage Target to Carrier Frequency.

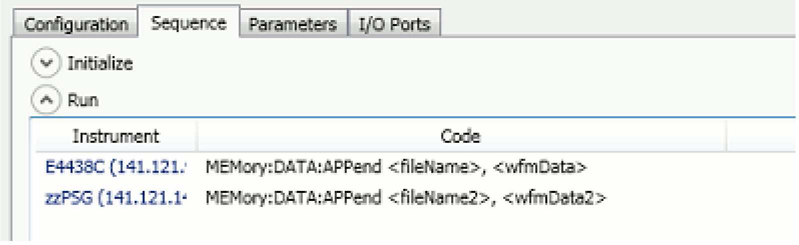
- other typical initialization operations are:
  - Reset instrument, or turn off signal outputs
  - For baseband signal generation: Clean up waveform memory
  - For RF signal generation: RF power level

**CAUTION** When downloading a large amount of waveform data (e.g. over 1 M I/Q sample pairs), add instrument communication timeout control **DefaultTimeout** to provide ample time for waveform downloading. Refer to Getting Started with Measurement Automation for more details.



| Instrument | Code | Usage Target |
|---|---|---|
| ZZMXA (141.121.! | <fcarrierHz> = SENSe:FREQuency:RF:CENTer ? | Carrier Frequency |
| ZZMXA (141.121.! | SENSe:WAVeform:DIF:BANDwidth <BWHz> | |
| ZZMXA (141.121.! | SENSe:WAVeform:SWEep:TIME 0.001 | |
| ZZMXA (141.121.! | INITiate:WAVeform | |
| ZZMXA (141.121.! | *OPC? | |
| ZZMXA (141.121.! | <samplePeriodSec> = SENSe:WAVeform:APERture ? | Sampling Period/Step |

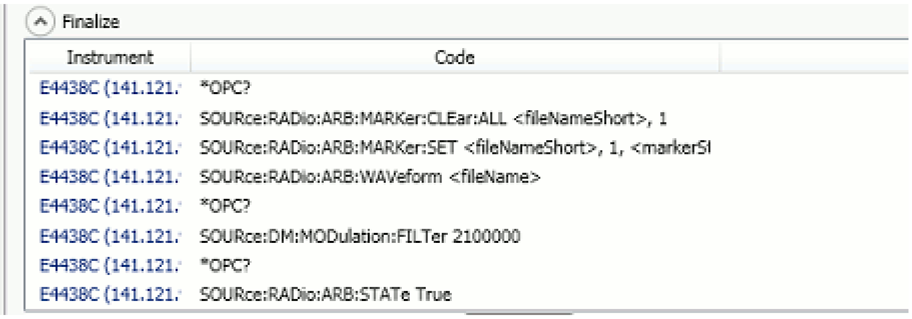
- Run block should only contain operation to download waveform data. Also, use waveform append operation if available.



| Instrument | Code |
|---|---|
| E4438C (141.121.' | MEMory:DATA:APPend <fileName>, <wfmData> |
| zzPSG (141.121.1' | MEMory:DATA:APPend <fileName2>, <wfmData2> |

- Finalize block is where to configure things such as Event Marker, multi-source MIMO synchronization, base band filtering, turn ON baseband signal or RF output, etc. to happen.

**CAUTION** It is **highly recommended** to have the very first operation in this block to check the operation completeness of waveform downloading. In the case of SCPI programming, use ***OPC?** command.

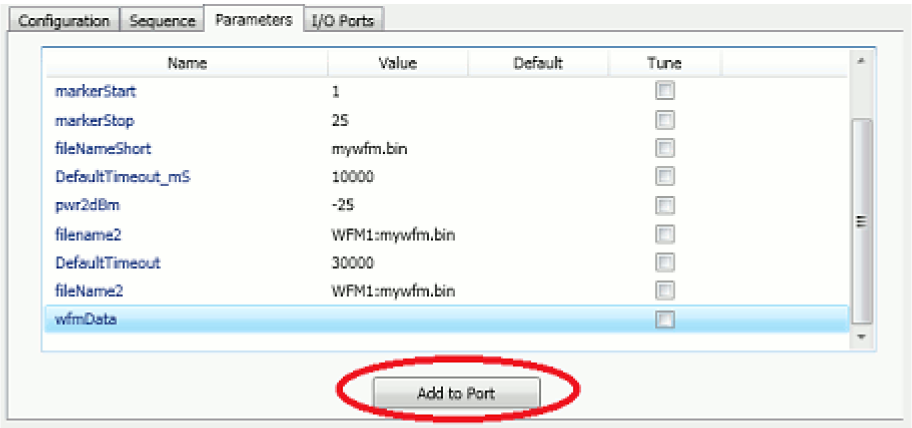


| Instrument | Code |
|---|---|
| E4438C (141.121.' | *OPC? |
| E4438C (141.121.' | SOURce:RADio:ARB:MARKer:CLEar:ALL <fileNameShort>, 1 |
| E4438C (141.121.' | SOURce:RADio:ARB:MARKer:SET <fileNameShort>, 1, <markerSt |
| E4438C (141.121.' | SOURce:RADio:ARB:WAVeform <fileName> |
| E4438C (141.121.' | *OPC? |
| E4438C (141.121.' | SOURce:DM:MODulation:FILTer 2100000 |
| E4438C (141.121.' | *OPC? |
| E4438C (141.121.' | SOURce:RADio:ARB:STATe True |

### III. Set up Parameters

All parameters defined in Keysight Command Expert will be exposed in Parameters page.

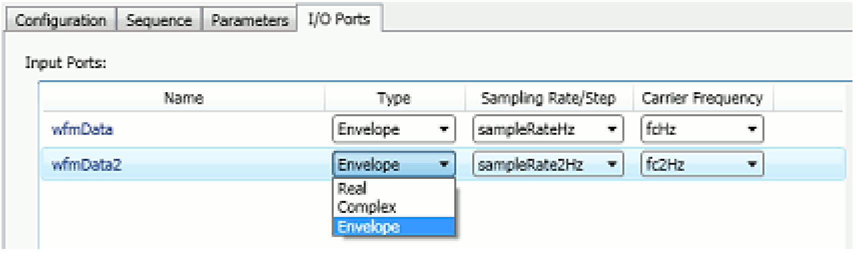
- You must explicitly select the parameter(s) designated for waveform data and convert it(them) to input port(s). If you make a mistake, go to I/O Ports page and convert it back to parameter with Add to Parameter button.
- Pay close attention to waveform file name syntax for downloading operation and for Event Marker configuration operation if applicable.



| Name | Value | Default | Tune |
|---|---|---|---|
| markerStart | 1 | | ☐ |
| markerStop | 25 | | ☐ |
| fileNameShort | mywfm.bin | | ☐ |
| DefaultTimeout_mS | 10000 | | ☐ |
| pwr2dBm | -25 | | ☐ |
| filename2 | WFM1:mywfm.bin | | ☐ |
| DefaultTimeout | 30000 | | ☐ |
| fileName2 | WFM1:mywfm.bin | | ☐ |
| wfmData | | | ☐ |

### IV. Configure Output Port(s)

Make sure to select correct data type for each port (note Envelope for RF signal) and associate it with appropriate sampling rate and (for RF) carrier frequency too.

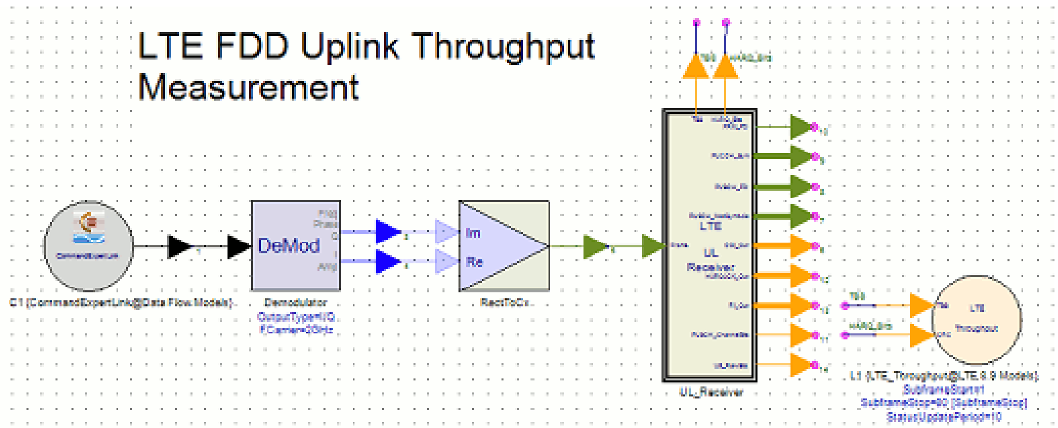


Input Ports:

| Name | Type | Sampling Rate/Step | Carrier Frequency |
|---|---|---|---|
| wfmData | Envelope | sampleRateHz | fcHz |
| wfmData2 | Envelope | sampleRate2Hz | fc2Hz |

### Steps to Configure CommandExpertLink Part to Upload Data from Instrument(s)

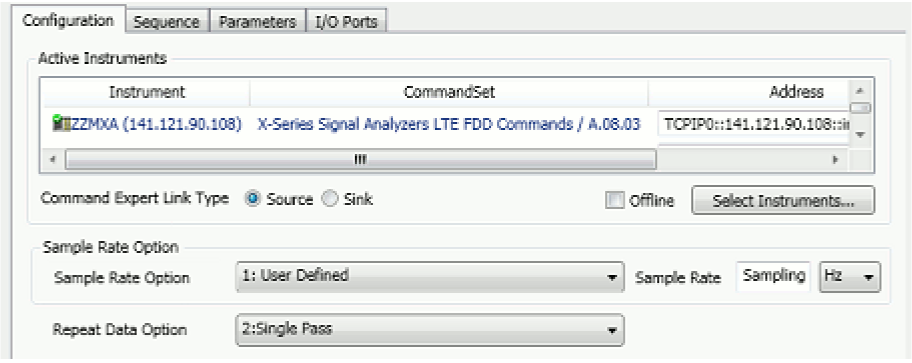
Use the CommandExpert_LTE_FDD_UL_Throughput.wsv example in examples archive under Instruments\CommandExpert* directory to help you with your exploration. This example is designed to *perform an LTE Throuput (BLER/BER) measurement using {CommandExpertLink] part to get data captured by Keysight MXA Vector Signal Analyzer.



I. Configure the Linkage Type

- Typically you would choose User Defined for Sampling Rate Option and provide the sampling rate specific to the digital demodulation protocol. Since instrument may be configured at fixed choices of BW or sampling rate, CommandExpertLink part will provide appropriate re-sampling if necessary.

- Use Repeat Data Option to control whether reuse, recapture or pad zeros if data from one single capture is not enough for the simulation need.



Active Instruments

| Instrument | CommandSet | Address |
|---|---|---|
| ZZMXA (141.121.90.108) | X-Series Signal Analyzers LTE FDD Commands / A.08.03 | TCPIP0::141.121.90.108::i |

Command Expert Link Type ● Source ○ Sink  □ Offline

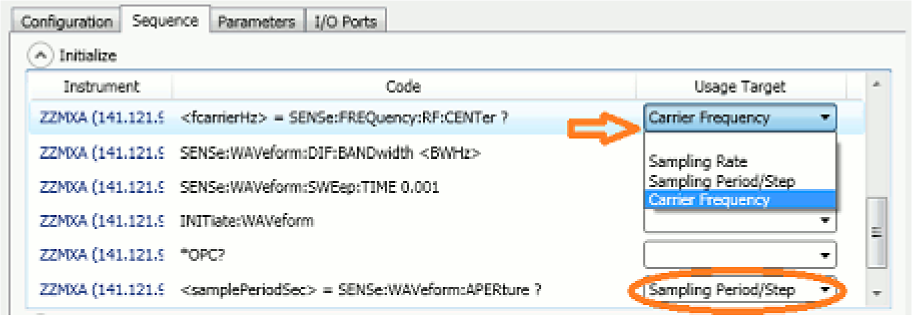
Sample Rate Option

Sample Rate Option: 1: User Defined  Sample Rate: Sampling Hz

Repeat Data Option: 2:Single Pass

II. Create Sequence Code

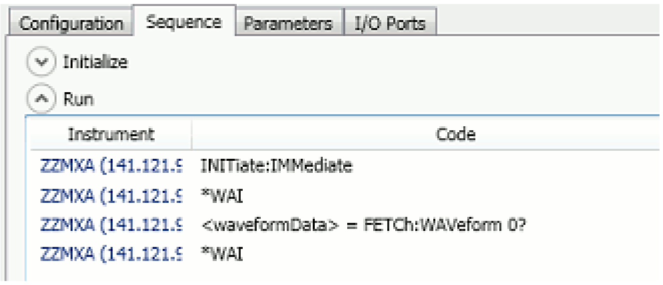
- In Initialize block: All of the instrument initialization/configuration goes here. In particular:
  - You must get sampling rate or sampling period (also known as aperture) from the instrument and notify CommandExpertLink part by setting the Usage Target for its parameter accordingly.
  - For RF signal, you must get carrier frequency from the instrument and notify CommandExpertLink part by setting the Usage Target to Carrier Frequency.
  - Other typical initialization operations are:
    - reset instrument
    - set up measurement trigger/synchronization, input attenuation levels, and measurement/capture duration.



| Instrument | Code | Usage Target |
|---|---|---|
| ZZMXA (141.121.9 | <fcarrierHz> = SENSe:FREQuency:RF:CENTer ? | Carrier Frequency |
| ZZMXA (141.121.9 | SENSe:WAVeform:DIF:BANDwidth <BWHz> | |
| ZZMXA (141.121.9 | SENSe:WAVeform:SWEep:TIME 0.001 | |
| ZZMXA (141.121.9 | INITiate:WAVeform | |
| ZZMXA (141.121.9 | *OPC? | |
| ZZMXA (141.121.9 | <samplePeriodSec> = SENSe:WAVeform:APERture ? | Sampling Period/Step |

- Run block should only contain SCPI command (or IVI-COM function) to perform a capture followed by command (or IVI-COM function) to retrieve captured data. Make sure captured data are collected into a parameter in Keysight Command Expert. Note this parameter will automatically become an output port in CommandExpertLink part. Also, note in the case of SCPI command programming, use *WAIcommand to guarantee measurement and data retrieving completion.

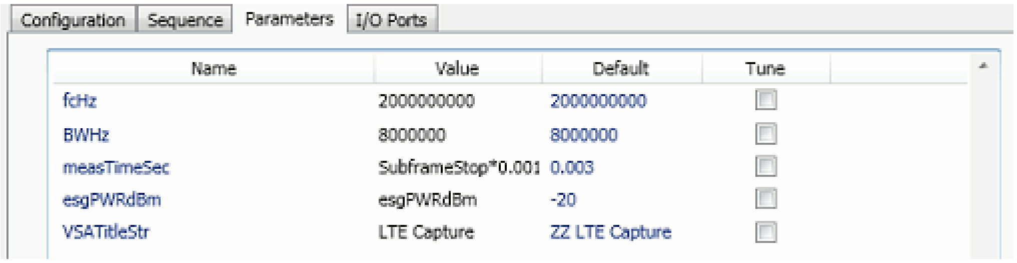
**CAUTION** In the case of SCPI programming, when retrieving data, you may encounter **Big-Endian/Small-Endian** issue (i.e. data byte order in instrument is opposite to data byte order on your computer), look for FORMat.BORDer command to resolve it.



| Instrument | Code |
|---|---|
| ZZMXA (141.121.9 | INITiate:IMMediate |
| ZZMXA (141.121.9 | *WAI |
| ZZMXA (141.121.9 | <waveformData> = FETCh:WAVeform 0? |
| ZZMXA (141.121.9 | *WAI |

### III. Setup Parameters

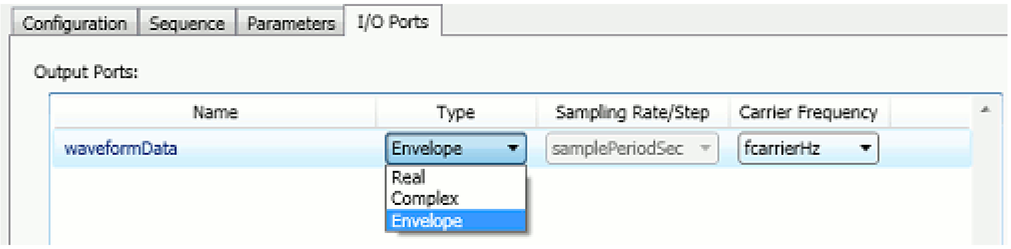
Typically measurement configurations such as carrier frequency, bandwidth, measurement duration, etc. should be exposed as parameters in Keysight Command Expert and thus exposed in the Parameters page.



| Name | Value | Default | Tune |
|---|---|---|---|
| fcHz | 2000000000 | 2000000000 | ☐ |
| BWHz | 8000000 | 8000000 | ☐ |
| measTimeSec | SubframeStop*0.001 | 0.003 | ☐ |
| esgPWRdBm | esgPWRdBm | -20 | ☐ |
| VSATitleStr | LTE Capture | ZZ LTE Capture | ☐ |

### IV. Configure Output Port(s)

Make sure to associate appropriate data type (Envelope type for RF capture), sampling rate, and carrier frequency (for RF capture) to the dynamically generated output port(s).



Output Ports:

| Name | Type | Sampling Rate/Step | Carrier Frequency |
|---|---|---|---|
| waveformData | Envelope | samplePeriodSec | fcarrierHz |

### Notes

One single CommandExpertLink part allows controlling more than one instruments. As shown in both examples mentioned earlier. In particular:

- In CommandExpertRFSources.wsv example it configures two RF sources and download waveform data into both of them.
- In CommandExpert_LTE_FDD_UL_Throughput.wsv example it controls the RF signal generator's power level in addition to uploading data captured by MXA. Additionally, by making the RF power level tunable, the example sweeps RF power level hence producing a throughput vs. RF power measurement.

## Using Command Expert in MATLAB Script

Prerequisites

- Installation of Keysight IO Library.
- Installation of Keysight Command Expert
- (Optionally for IVI-COM Exploration) Installation of IVI-COM drivers through Keysight Technical Support for Test & Measurement web (Go to Drivers, Updates & Examples and look into IVI under By Class)

Associated tutorial workspace

- Examples\Instruments\CommandExpert\MXA Source_AutoRange.wsv is an SCPI based example.
- Examples\Instruments\CommandExpert\MXA PeakFrequency.wsv is an IVI-COMbased example.

**CAUTION** We recommend you to copy the complete directory of **Examples\Instruments\CommandExpert** to your local directory to avoid file writing permission issues on default installation directory.

### Introduction

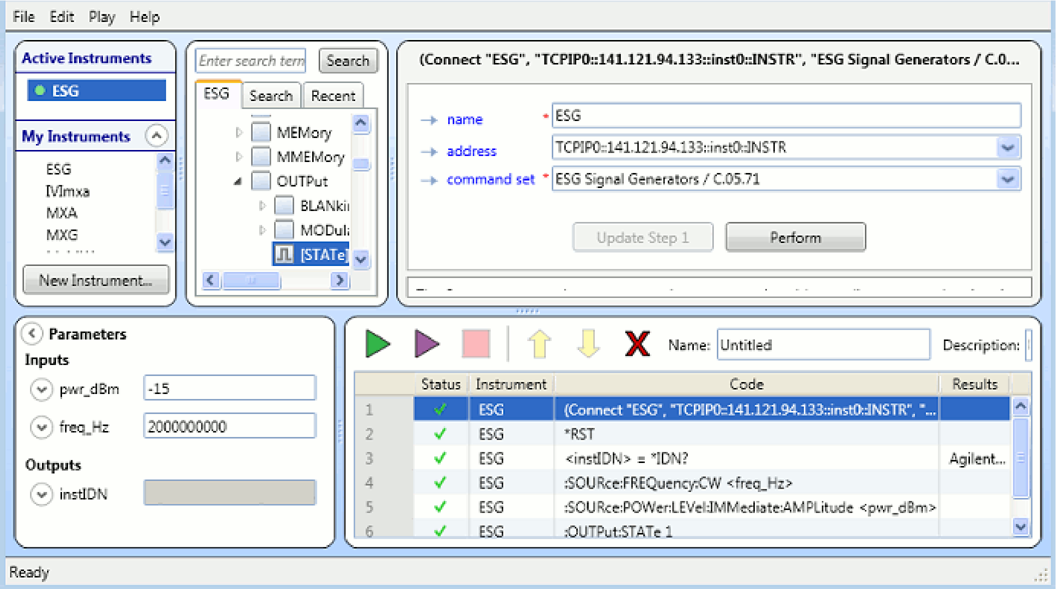
Use Command Expert access function in MATLAB Script Equations environment for flexible instrument control.

All you need to do is:

1. Develop instrument control sequence (either in SCPI commands or IVI-COM functions) in Command Expert application and save the sequence. (Refer to Getting Started with Measurement Automation for how to use Command Expert application).
2. Use MATLAB Script function agRunSequence to exercise the sequence file.

## Create Control Sequence in Command Expert

- Develop instrument control sequence (either in SCPI commands or IVI-COM functions) in Command Expert application as described in Getting Started with Measurement Automation
  Pay close attention to the parameters created in Command Expert application.
  - Parameters for SCPI commands or IVI-COM functions will directly correspond to parameters passed into agRunSequence function according to their order. (Refer to the following Use Command Expert Sequence in Equations Environment section for details).
  - Parameters that hold return results from SCPI (query) commands or IVI-COM functions will correspond to return results of agRunSequence function execution. (Refer to the following Use Command Expert Sequence in Equations Environment section for details).
    Note in the screen capture below for Keysight ESG RF Synthesizer, we expose its carrier frequency and power level as input parameters, i. e. freq_HZ and pwr_dBm, and expose one return result, i.e. parameter instIDN, to return *IDN? query result.



(Connect "ESG", "TCPIP0::141.121.94.133::inst0::INSTR", "ESG Signal Generators / C.0...

name * ESG
address TCPIP0::141.121.94.133::inst0::INSTR
command set * ESG Signal Generators / C.05.71

Inputs
pwr_dBm -15
freq_Hz 2000000000

Outputs
instIDN

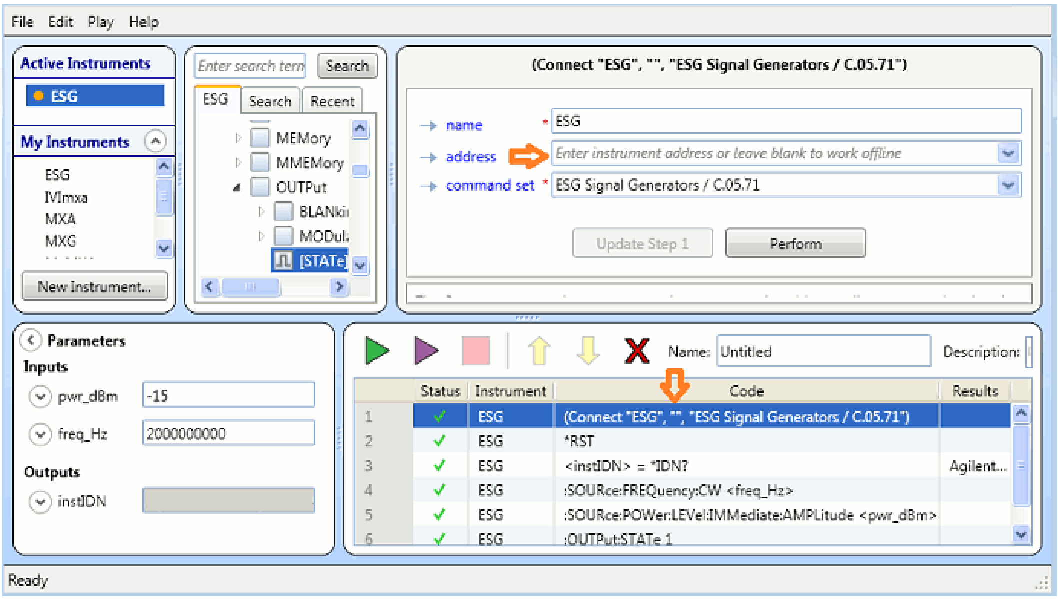
Name: Untitled
Description:

| | Status | Instrument | Code | Results |
|---|---|---|---|---|
| 1 | ✓ | ESG | (Connect "ESG", "TCPIP0::141.121.94.133::inst0::INSTR", "... | |
| 2 | ✓ | ESG | *RST | |
| 3 | ✓ | ESG | <instIDN> = *IDN? | Agilent... |
| 4 | ✓ | ESG | :SOURce:FREQuency:CW <freq_Hz> | |
| 5 | ✓ | ESG | :SOURce:POWer:LEVel:IMMediate:AMPLitude <pwr_dBm> | |
| 6 | ✓ | ESG | :OUTPut:STATe 1 | |

- Once the sequence development is complete, save them into a Command Expert sequence file (.iseq).
  - Use Command Expert menu File -> Save Sequence

- We recommend removing the address information (and don't forget to click the Update Step 1 button to ensure it is removed from the actual operation sequence) prior to saving. This way, the saved sequence shows no specific instrument address.

- For convenience, we recommend saving the sequence .iseq file into the same directory of the workspace that will use the sequence file.



(Connect "ESG", "", "ESG Signal Generators / C.05.71")

name * ESG
address Enter instrument address or leave blank to work offline
command set * ESG Signal Generators / C.05.71

Parameters
Inputs
pwr_dBm: -15
freq_Hz: 2000000000
Outputs
instIDN

Name: Untitled
Description:

| | Status | Instrument | Code | Results |
|---|---|---|---|---|
| 1 | ✓ | ESG | (Connect "ESG", "", "ESG Signal Generators / C.05.71") | |
| 2 | ✓ | ESG | *RST | |
| 3 | ✓ | ESG | <instIDN> = *IDN? | Agilent... |
| 4 | ✓ | ESG | :SOURce:FREQuency:CW <freq_Hz> | |
| 5 | ✓ | ESG | :SOURce:POWer:LEVel:IMMediate:AMPLitude <pwr_dBm> | |
| 6 | ✓ | ESG | :OUTPut:STATe 1 | |

## Use Command Expert Sequence in Equations Environment

Use the MATLAB Script function agRunSequence to exercise the sequence file. Make sure:

- The very first parameter of agRunSequence must be the Command Expert sequence .iseqfile.
  - If the file is in the same directory as the workspace, you can just use the file name and you can also omit the .iseq file extension.
  - Regardless of where the file resides, you can always use the full file path (including full directory information and complete file name with .iseq extension).
- The second parameter of agRunSequence must be the full VISA address of the instrument.

  **NOTE** If you have Keysight IO Library installed, you can use Keysight Connection Expert to find the complete VISA address after you add your instrument to it.

- You must provide the same number of input parameters as specified in the sequence of the .iseq file to the agRunSequence function following the sequence file name and instrument VISA address.

  **NOTE**

Parameters are associated based on the order in which they appear in the **Parameters** panel of Command Expert application when creating the sequence. ( **Not the order these parameters appear in the sequence.**)

- You must provide the same number of MATLAB Script variables as parameterized query results of the sequence in the .iseq file to receive return results from agRunSequence function call.

NOTE Similar to passing input parameters, variables are associated with the parameterized query results based on the order in which these parameterized queries appear in the **Parameters** panel of Command Expert application when creating the sequence.

For our screen capture example, the MATLAB Script code would be:

```
myRFSrc = 'TCPIP0::111.222.333.444::inst0::INSTR'; %
VISA address for the RF synthesizer
rfPowerDBm = -20; % -20 dBm RF power
rfFreqHz = 1.9e9; % 1.9 GHz center frequency
myIDN = agRunSequence('RFSourceDemo', myRFSrc,
rfPowerDBm, rfFreqHz);
```

CAUTION Once again, notice the power level is before carrier frequency in agRunSequence function call, while as shown in the screen capture earlier, carrier frequency command is executed before the RF power command in the commands sequence.

## Examples

There are a couple examples under the example archive directory Instruments /CommandExpert/ directory.

- The example MXA Source_AutoRange.wsv is for Keysight MXA Vector Signal Analyzer. It uses two SCPI command based sequences, i.e. MXA Source_AutoRange_Init.iseq and MXA Source_AutoRange.iseq, in the Equations for the schematic.
- The example MXA PeakFrequency.wsv is also for Keysight MXA Vector Signal Analyzer. It uses IVI-COM based sequence MXA PeakFrequency.iseq.

Here is an excerpt from example MXA PeakFrequency.wsv just to highlight how to handle multiple return results, in this case, three return results that will be stored in variables amplitude, peakX and peakY:

```
address = 'TCPIP0::111.222.333.444::inst0::INSTR';
start = 1.975e9; % Start frequency
stop = 2.025e9;   % Stop frequency
```

```
% Run command sequence.
[amplitude, peakX, peakY] = agRunSequence('MXA
PeakFrequency', address, start, stop);
```

## Using Waveform Sequencer Composer

### Using Waveform Sequencer Composer

The Waveform Sequencer Composer is a tool that can be used to create a custom waveform from multiple waveform segments. This tutorial contains three examples to demonstrate various capabilities of the Waveform Sequencer Composer:

1. Example 1: Waveform Sequence Composer Playback
   This example demonstrates the basic functionality of the Waveform Sequence Composer by combining three different signals and validate the combined signal using the Waveform Sequence Composer source.
2. Example 2: Sweeping parameters in Waveform Sequence Composer
   This example shows how to use variables and sweeps with Waveform Sequence Composers.
3. Example 3: Frequency Hopping Waveforms Using Waveform Sequence Composer
   This example illustrates the use of Waveform Sequence Composer to construct a frequency hopping waveform for radar applications. In this examples, the details of using the M8190 instrument through the Waveform Sequence Composer are illustrated.

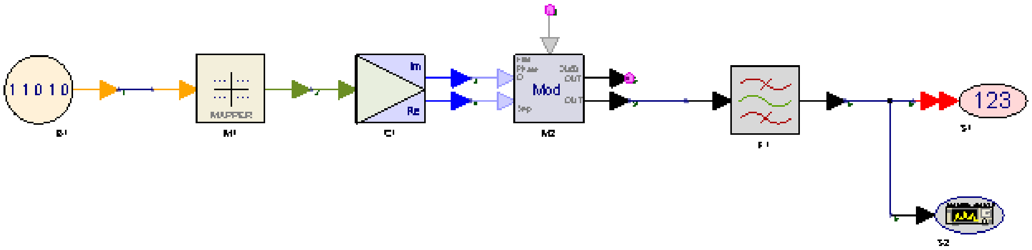
### Waveform Sequence Composer Playback

Open the tutorial example "Examples\Tutorials\Measurement_Automation\WaveformSequencerPlayback.wsv".

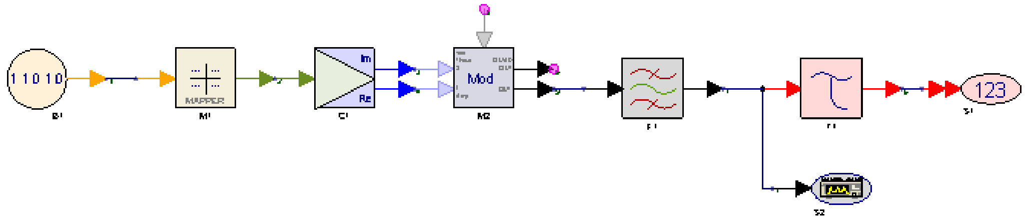
### Initial Setup

Three designs (Design1, Design2, and Design3) exist in the workspace and the goal is to sequence the three waveforms together in order.

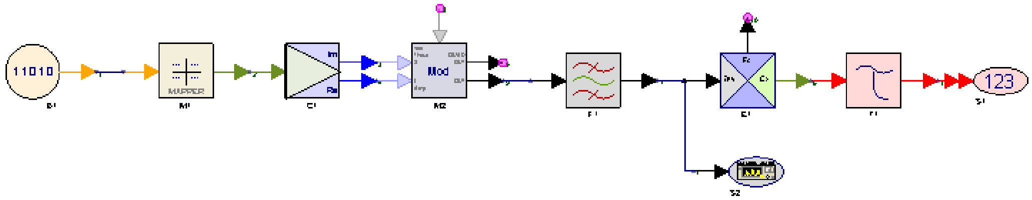
Design1 uses a QPSK modulation type to create a signal at a carrier frequency of 1001.5 MHz which outputs envelope data.



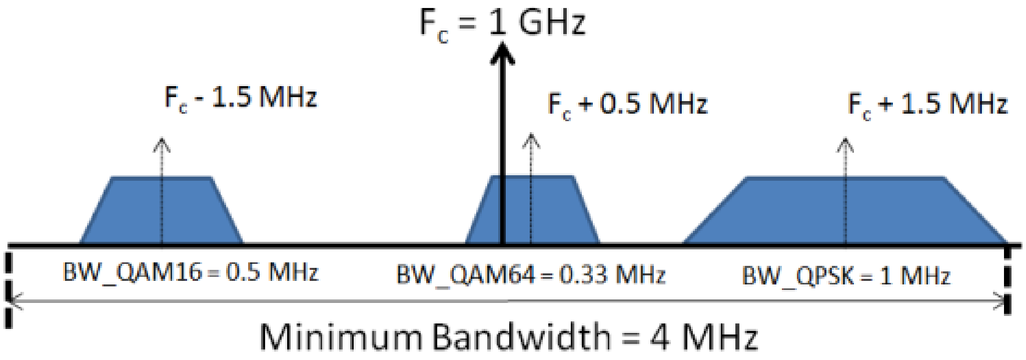
Design2 uses a QAM 16 modulation type to create a signal at a carrier frequency of 998.5 MHz which also outputs envelope data. Additionally, a time delay has been added to the signal which is equal to the length of the QPSK waveform.



Design3 users a QAM 64 modulation type to create a signal at a carrier frequency of 1000.5 MHz. The output for this signal has been converted to complex data. A time delay has been added as well which is equal to the length of the QPSK and QAM16 waveform.

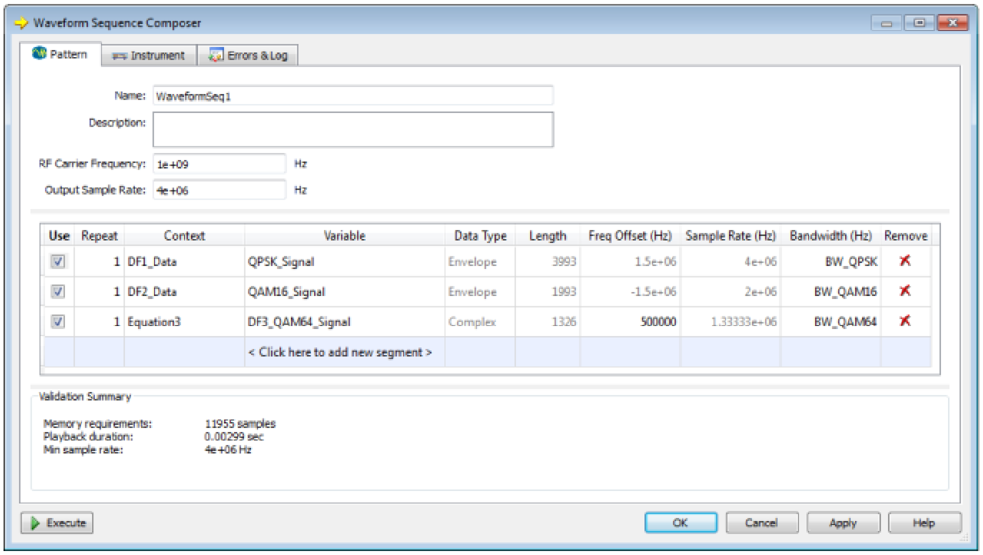


The figure below illustrates a simplified representation of the spectrum of resulted from combining all the three signals into a single waveform with the carrier frequency at 1 GHz. Note that the minimum bandwidth is based on the distance between the center carrier frequency and the furthest waveform from it. The bandwidth of the signals is based on the worst case scenario when the roll-off factor of the pulse shaping filter is equal to 1.



To setup a Waveform Sequence Composer, right-click on the workspace tree and select Add > Add Waveform Sequence Composer ... option. The Waveform Sequence Composer window is displayed to configure the Waveform Sequence Composer parameters. Apply the following configurations:

1. Set the RF Carrier Frequency, which is the carrier frequency of the instrument, to 1e+9 ( 1 GHz ), and then set the output sample rate to 4e+6. The minimum sample rate can be inferred from the minimum bandwidth shown in the figure of the simplified spectrum representation of the combined signal.
2. Select Click here to add new segment to define a new segment. In the Context entry, select from the drop down box which dataset or equation that defines the variable that holds the segment data samples. Once a valid context is selected, then you can select a variable from that context in the next column i.e., Variable column. Selecting a variable will populate the rest of the columns. Perform this step for the variables QPSK_Signal and QAM16_Signal and DF3_QAM64_Signal as shown in the figure below:



Name: WaveformSeq1
Description:
RF Carrier Frequency: 1e+09 Hz
Output Sample Rate: 4e+06 Hz

| Use | Repeat | Context | Variable | Data Type | Length | Freq Offset (Hz) | Sample Rate (Hz) | Bandwidth (Hz) | Remove |
|---|---|---|---|---|---|---|---|---|---|
| ☑ | 1 | DF1_Data | QPSK_Signal | Envelope | 3993 | 1.5e+06 | 4e+06 | BW_QPSK | ✗ |
| ☑ | 1 | DF2_Data | QAM16_Signal | Envelope | 1993 | -1.5e+06 | 2e+06 | BW_QAM16 | ✗ |
| ☑ | 1 | Equation3 | DF3_QAM64_Signal | Complex | 1326 | 500000 | 1.33333e+06 | BW_QAM64 | ✗ |
|  |  |  | < Click here to add new segment > |  |  |  |  |  |  |

Validation Summary

Memory requirements: 11955 samples
Playback duration: 0.00299 sec
Min sample rate: 4e+06 Hz

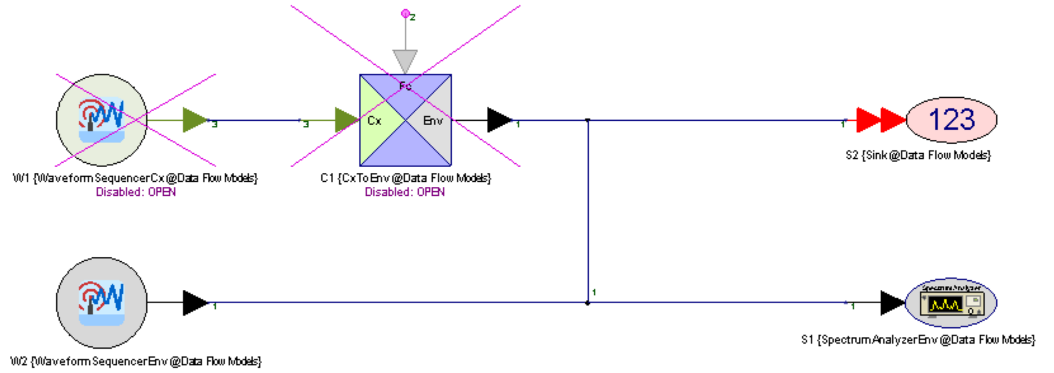
3. Note that each segment bandwidth is set automatically to the segment sample rate. This will make the minimum acceptable output sample rate around 7 MHz. However, if the actual bandwidth of the segment is known, then it can be applied to the segment bandwidth setting. In this example, set the bandwidth of each segment i.e., BW_QPSK, BW_QAM16 and BW_QAM64 as shown in the figure above. These variables are calculated in GlobalParameters equation page based on the following equation:
BW = ( 1 + Roll_Off ) * SymbolRate

### Verify the Combined Waveform

If the configurations of the Waveform Sequence Composer are valid, a validation summary at the bottom of the pattern page will provide the memory requirements, playback duration and minimum sample rate. Since the output sample rate is different than the sample rate of some segments, these segments will be resampled to the output sample rate. Detailed segments information after resampling can be obtained from the Errors & logs page of the Waveform Sequence Composer user interface.

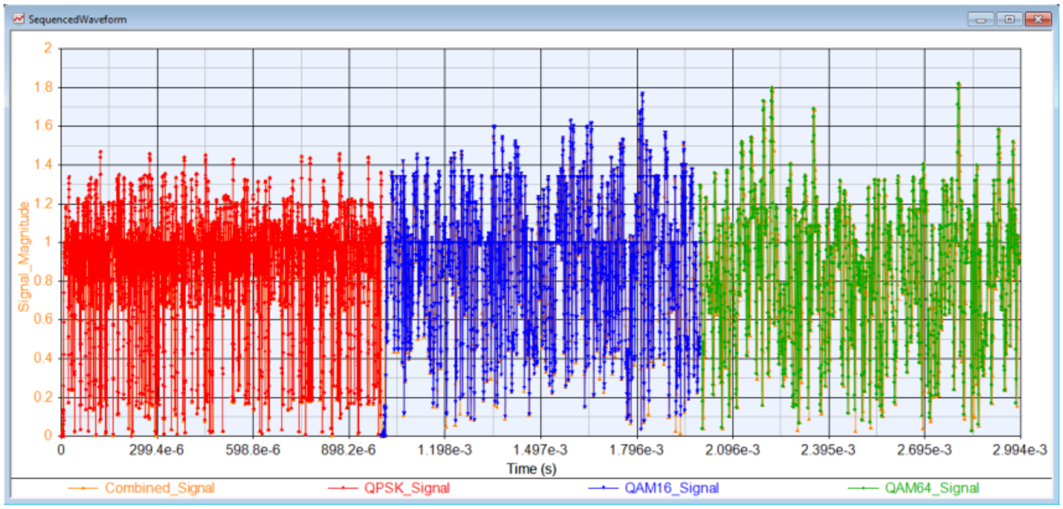
To verify the spectral and time domain characteristics of the combined waveform, a Waveform Sequence Composer source can be used by dragging the Waveform

Sequence Composer item in the workspace tree and dropping it into a schematic. In this example, the Waveform Sequence Composer has been dropped onto the Design4 schematic twice in order to show the two different models of the Waveform Sequence Composer source i.e., the envelope model and the complex model as shown in the figure below

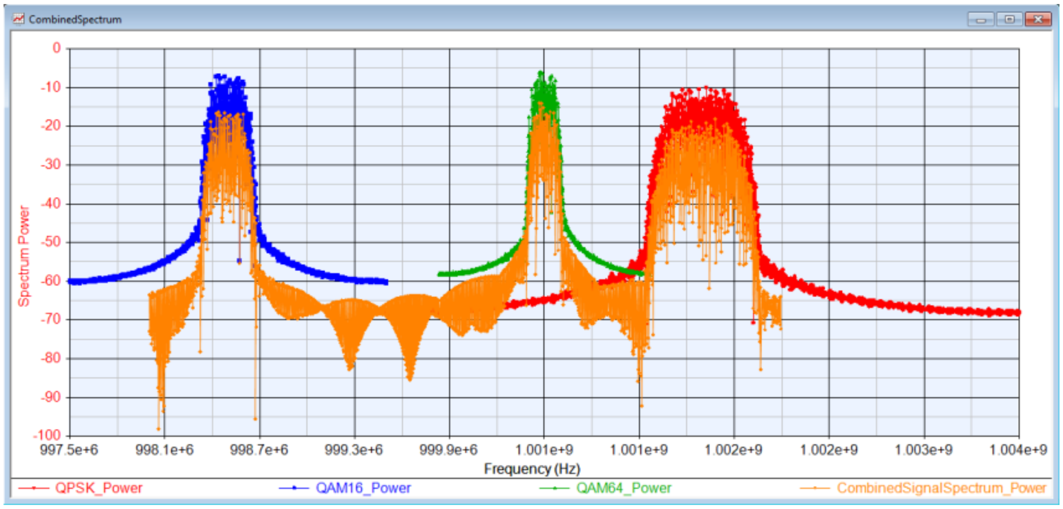


Fc
Cx
Env
123
W1 {WaveformSequencerCx@Data Flow Models}
Disabled: OPEN
C1 {CxToEnv@Data Flow Models}
Disabled: OPEN
S2 {Sink@Data Flow Models}
W2 {WaveformSequencerEnv@Data Flow Models}
S1 {SpectrumAnalyzerEnv@Data Flow Models}

Comparing the time domain waveform segments to the composed waveform in the SequencedWaveform graph we can see the data matches.



Below you can see the comparison of the individual and composed spectrum.



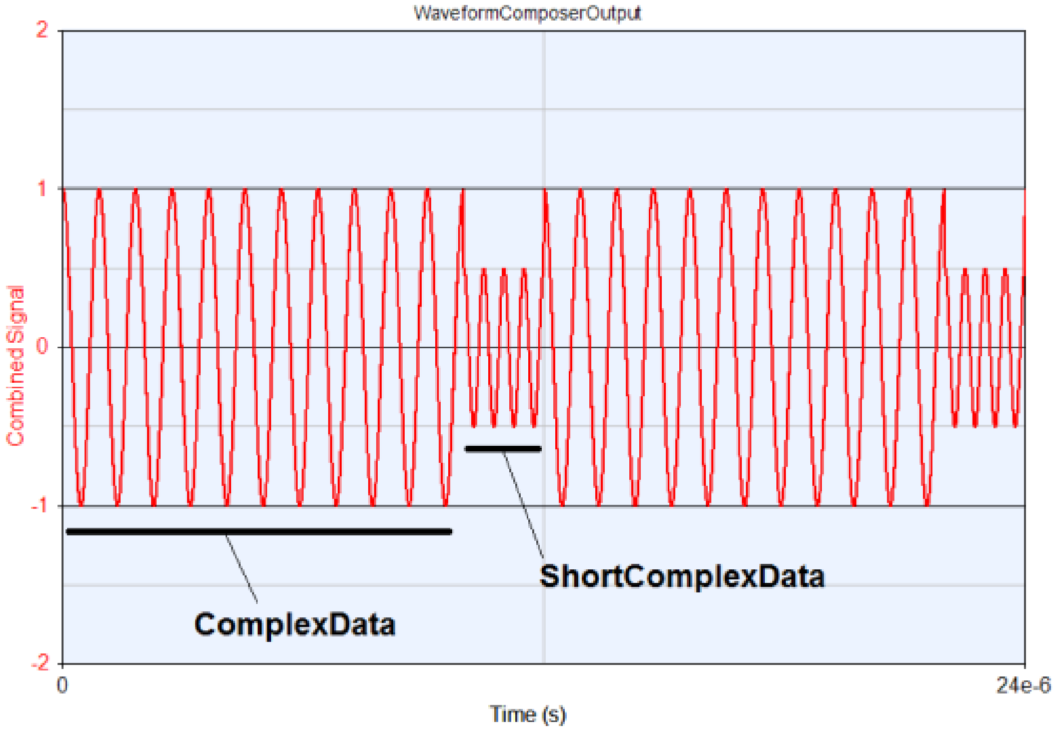
CAUTION The difference between the combined spectrum level and the spectrum level of each signal is due to the frequency resolution used in each of these spectrum plots.

### Sweeping parameters in Waveform Sequence Composer

Open the tutorial example "Examples\Tutorials\Measurement_Automation\WaveformSequenceComposerSweep
wsv".

### Initial Setup

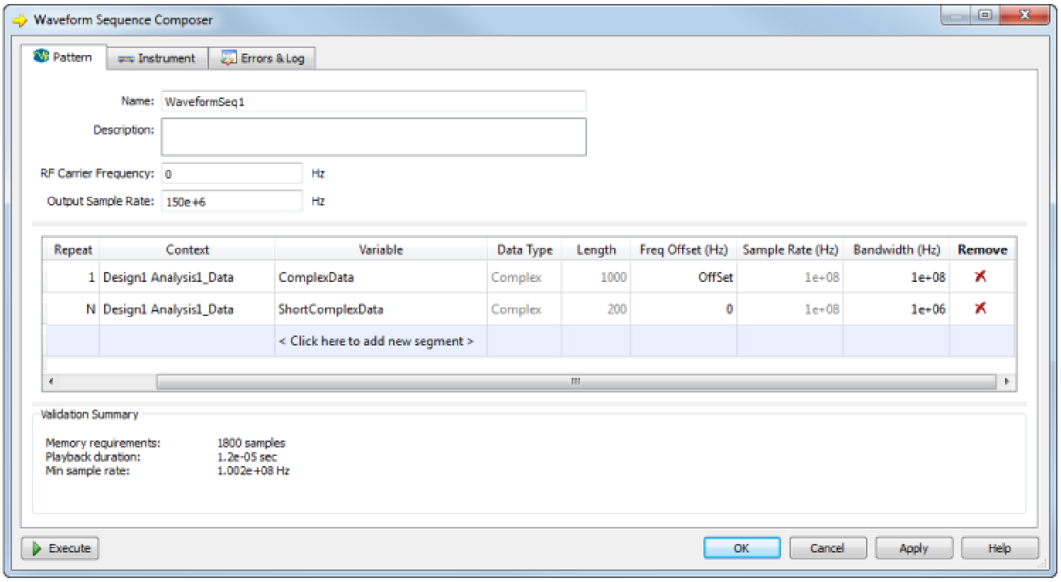
In this example, two complex sinusoidal signals i.e., ComplexData and ShortComplexData are combined into a single waveform using the Waveform Sequence Composer as shown in the figure below:



The combination is parameterized using two variables that are defined in "Equation 1" as tunable variables:

1. OffSet: is used to define the "Offset" parameter of the "ComplexData" segment. The default value is set to 0.
2. N: is used to define the "Repeat" parameter of the "ShortComplexData" segment. The default value is set to 1.

These variables are used in the Waveform Sequence Composer as shown in the figure below:



Name: WaveformSeq1

Description:

RF Carrier Frequency: 0 Hz

Output Sample Rate: 150e+6 Hz

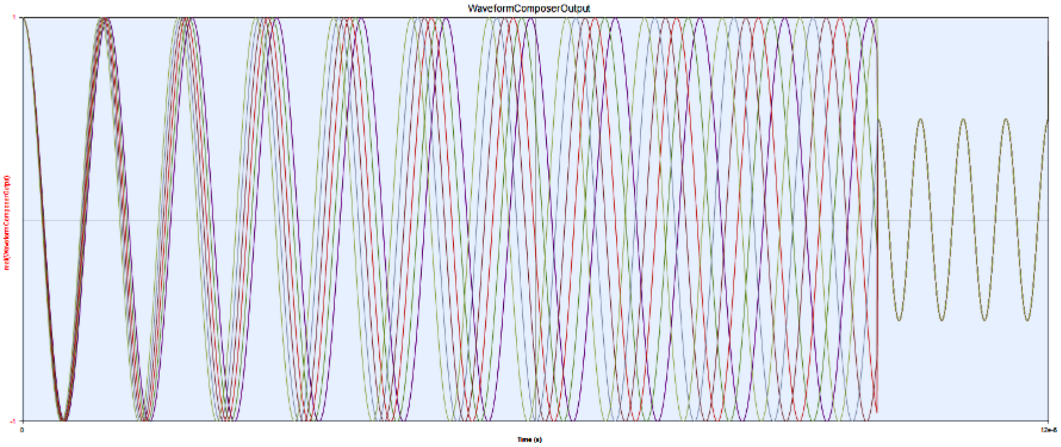
| Repeat | Context | Variable | Data Type | Length | Freq Offset (Hz) | Sample Rate (Hz) | Bandwidth (Hz) | Remove |
|---|---|---|---|---|---|---|---|---|
| 1 | Design1 Analysis1_Data | ComplexData | Complex | 1000 | OffSet | 1e+08 | 1e+08 | ✗ |
| N | Design1 Analysis1_Data | ShortComplexData | Complex | 200 | 0 | 1e+08 | 1e+06 | ✗ |
| | | < Click here to add new segment > | | | | | | |

Validation Summary

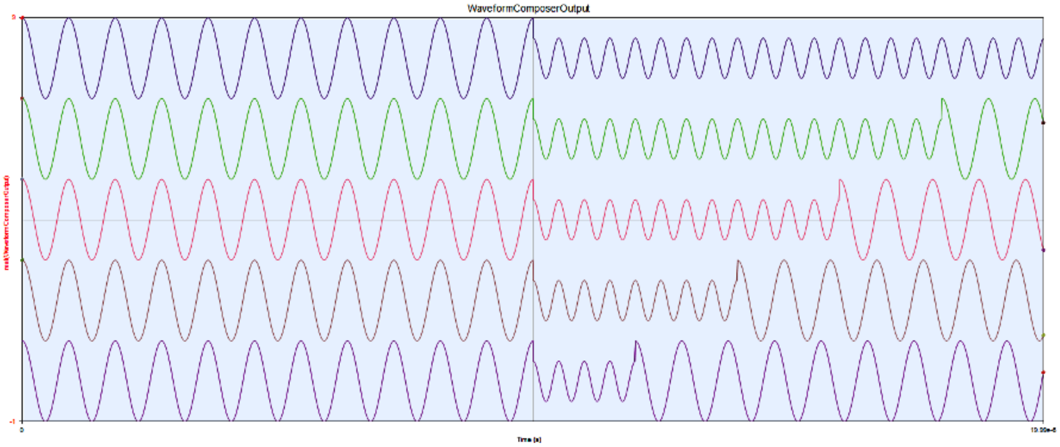
Memory requirements: 1800 samples
Playback duration: 1.2e-05 sec
Min sample rate: 1.002e+08 Hz

## Verify Combined Waveform

Use "Sweep1" to examine various values of the OffSet parameter. In the figure below, the combined waveforms when the Freq Offset of the ComplexData segment is set at various values from 1e4 - 1e5.



Similarly, use Sweep2 to examine various values of the N parameter. In the figure below, the combined waveforms when N=1,2,...,5 are displayed.



## Frequency Hopping Waveforms Using Waveform Sequence Composer

Open the tutorial example Examples\Instruments\Radar\Waveform Sequencing\EW_Targets_Frequency_Hopping_M8190.wsv

### Initial Setup

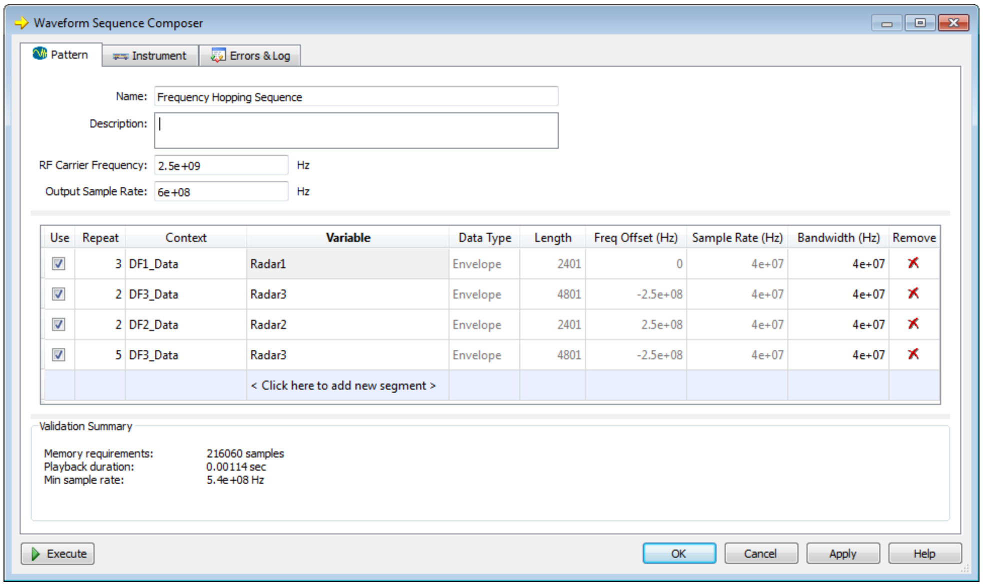
This example shows how to generate a sequence of radar signals with frequency hopping and play them using the M8190A signal generator. The example workspace generates three radar signals Radar1, Radar2 and Radar3 at different center frequency of 2.5 GHz, 2.75 GHz and 2.25 GHz respectively.

In this example, the three radar signals are combined as follows:

- Radar1: Repeated 3 times.
- Radar3: Repeated 2 times.
- Radar2: Repeated 2 times.
- Radar3: Repeated 5 times.

To apply this combination configuration to the Waveform Sequence Composer, follow the steps below:

1. Run the 3 Analyses that generate the three radar signals by clicking the "Go" button in "Generate Signals" Equations page or run the Analyses, namely "DF1", "DF2" and "DF3" one by one.
2. Open the Waveform Sequence Composer item on the workspace tree named "Frequency Hopping Sequence". On the pattern page, apply the settings as shown in the figure below:



Name: Frequency Hopping Sequence

Description:

RF Carrier Frequency: 2.5e+09 Hz

Output Sample Rate: 6e+08 Hz

| Use | Repeat | Context | Variable | Data Type | Length | Freq Offset (Hz) | Sample Rate (Hz) | Bandwidth (Hz) | Remove |
|---|---|---|---|---|---|---|---|---|---|
| ☑ | 3 | DF1_Data | Radar1 | Envelope | 2401 | 0 | 4e+07 | 4e+07 | ✗ |
| ☑ | 2 | DF3_Data | Radar3 | Envelope | 4801 | -2.5e+08 | 4e+07 | 4e+07 | ✗ |
| ☑ | 2 | DF2_Data | Radar2 | Envelope | 2401 | 2.5e+08 | 4e+07 | 4e+07 | ✗ |
| ☑ | 5 | DF3_Data | Radar3 | Envelope | 4801 | -2.5e+08 | 4e+07 | 4e+07 | ✗ |
|  |  |  | < Click here to add new segment > |  |  |  |  |  |  |

Validation Summary

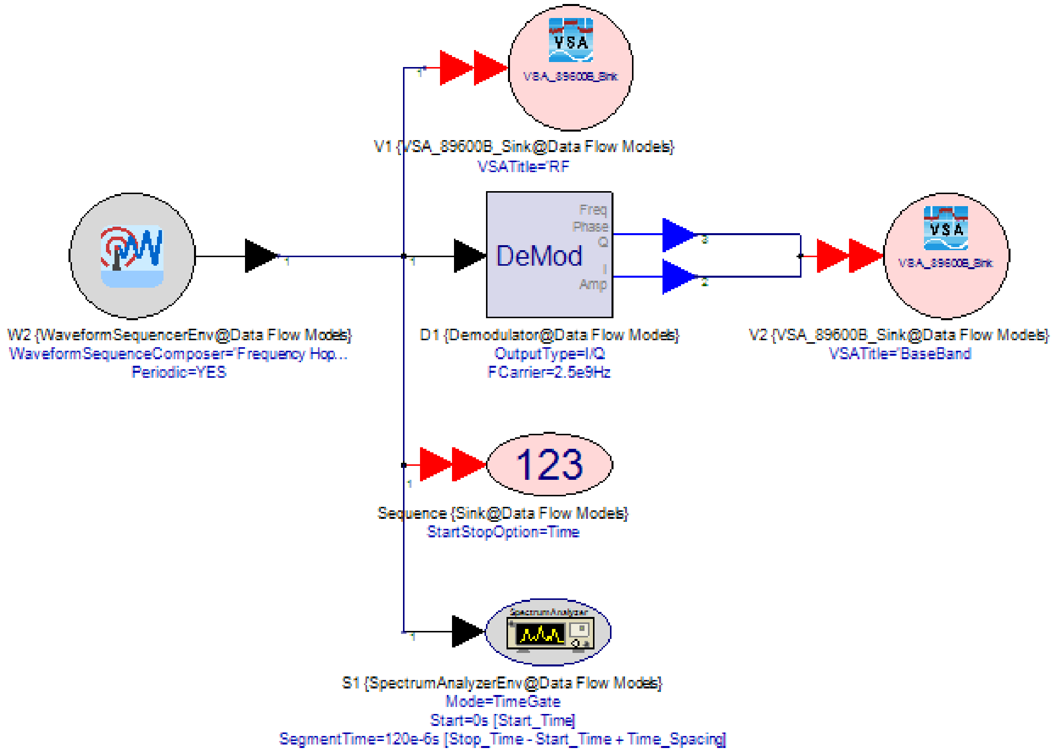
```
Memory requirements:     216060 samples
Playback duration:       0.00114 sec
Min sample rate:         5.4e+08 Hz
```

3. Go to "Instrument" tab. Type the IP address of the M8190 instrument into the "Address" cell of in the table under "Instrument Selection" area. As an alternative, you can click the "Select Instrument..." button to pick the instrument if it has been added into the "Keysight Connection Expert", or add it in the "Select Instruments" dialog.

4. Adjust the instrument parameters if needed. Note that "AutoScale" parameter is set up to auto scale all waveforms.
5. Click the "Execute" button and the waveforms should be downloaded into the M8190 and the sequence will be played out.
6. Go to the Errors & Log page and review the details of the execution operation.

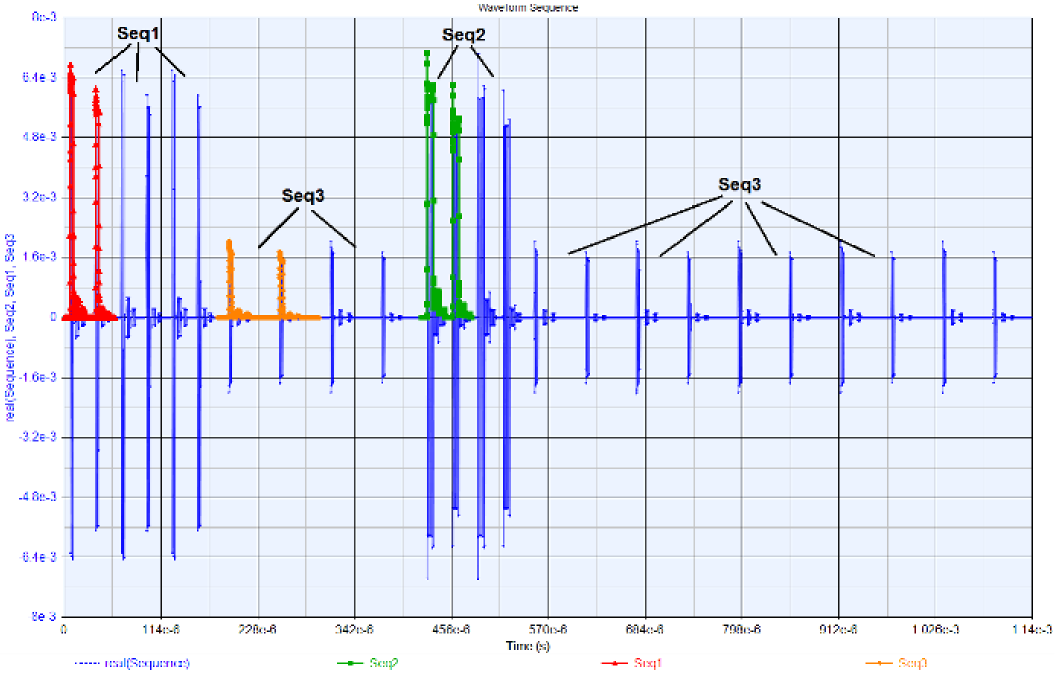
### Verify Combined Waveform

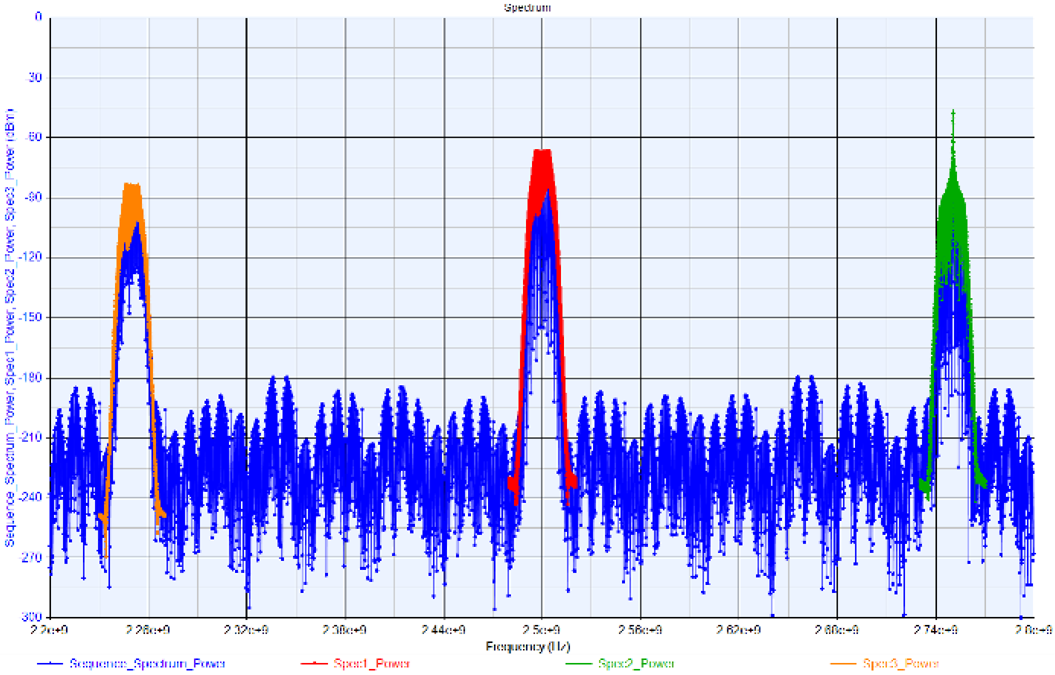
The validation of the combined signal is done by placing a WaveformSequencer source part in a schematic and examine its generated signal using a sink and Keysight 89600 Vector Signal Analyzer. See the "Sequence" schematic (its Analysis is "DF4") which is shown in the figure below:


VSA
VSA_89600B_Sink
V1 {VSA_89600B_Sink@Data Flow Models}
VSATitle="RF"
W2 {WaveformSequencerEnv@Data Flow Models}
WaveformSequenceComposer="Frequency Hop...
Periodic=YES
Freq
Phase
Q
I
Amp
DeMod
D1 {Demodulator@Data Flow Models}
OutputType=I/Q
FCarrier=2.5e9Hz
VSA
VSA_89600B_Sink
V2 {VSA_89600B_Sink@Data Flow Models}
VSATitle="BaseBand"
123
Sequence {Sink@Data Flow Models}
StartStopOption=Time
SpectrumAnalyzer
S1 {SpectrumAnalyzerEnv@Data Flow Models}
Mode=TimeGate
Start=0s [Start_Time]
SegmentTime=120e-6s [Stop_Time - Start_Time + Time_Spacing]

**NOTE** You can simply drag the "Frequency Hopping Sequence" icon on the workspace tree and drop it into the schematic and a "WaveformSequencer" part will be automatically placed on the schematic and properly configured.

Upon running the analysis "DF4", the combined signal is generated. The time signal and spectrum of that signal along with the three individual radar signals are plotted in the graphs WaveformSequence and Spectrum respectively. These graphs are shown also in the figures below:


Waveform Sequence
Seq1
Seq2
Seq3
Seq3
8e-3
6.4e-3
4.8e-3
3.2e-3
1.6e-3
0
-1.6e-3
-3.2e-3
-4.8e-3
-6.4e-3
-8e-3
real(Sequence), Seq2, Seq1, Seq3
0
114e-6
228e-6
342e-6
456e-6
570e-6
684e-6
798e-6
912e-6
1.026e-3
1.14e-3
Time (s)
real(Sequence)
Seq2
Seq1
Seq3


Spectrum
0
-30
-60
-90
-120
-150
-180
-210
-240
-270
-300
Sequence_Spectrum_Power, Spec1_Power, Spec2_Power, Spec3_Power (dBm)
2.2e+9
2.26e+9
2.32e+9
2.38e+9
2.44e+9
2.5e+9
2.56e+9
2.62e+9
2.68e+9
2.74e+9
2.8e+9
Frequency (Hz)
Sequence_Spectrum_Power
Spec1_Power
Spec2_Power
Spec3_Power

# Verification Test Bench Tutorial

In this tutorial, you will learn how to develop a SystemVue workspace to be used in Verification Test Bench (VTB) application.

## General VTB Development Flow

In VTB applications, a VTB producer develops the VTB workspace which defines the testing environment, i.e., the test inputs and outputs measurements, for a particular component or device under test (DUT). For example, the VTB workspace may contain a wireless transmitter simulation model which can be used to verify the performance of a particular component such as a filter, mixer, amplifier, etc. This workspace can be used by VTB consumers to conduct the performance test of that component under their own development environment, such as ADS or GoldenGate. This tutorial focuses on the VTB workspace creation process.

A typical flow for VTB workspace creation is as follows:

- Workspace Preparation: Create a data flow simulation workspace or modify an existing one.
- DUT Configuration: Insert the SVE_Link model to the simulation model of the DUT.
- Workspace Verification: Verify that the workspace is compatible with SystemVue Engine.
- SystemVueEngine Simulation: Simulate the workspace using SystemVue Engine
- Simulation Results Verification: Import the results of SystemVue Engine simulation to SystemVue.
- Workspace Publishing: Switch SVE_Link to be the active simulation model instead of the DUT simulation model.

The details of each stage in the workspace creation flow are illustrated through an example in VTB Example section.

## VTB Example

In this VTB example, a verification test bench workspace is developed for an up-conversion mixer in an 802.11ac transmitter using SystemVue. The following sections describe the details of each step in the VTB workspace creation flow.

### Workspace Preparation

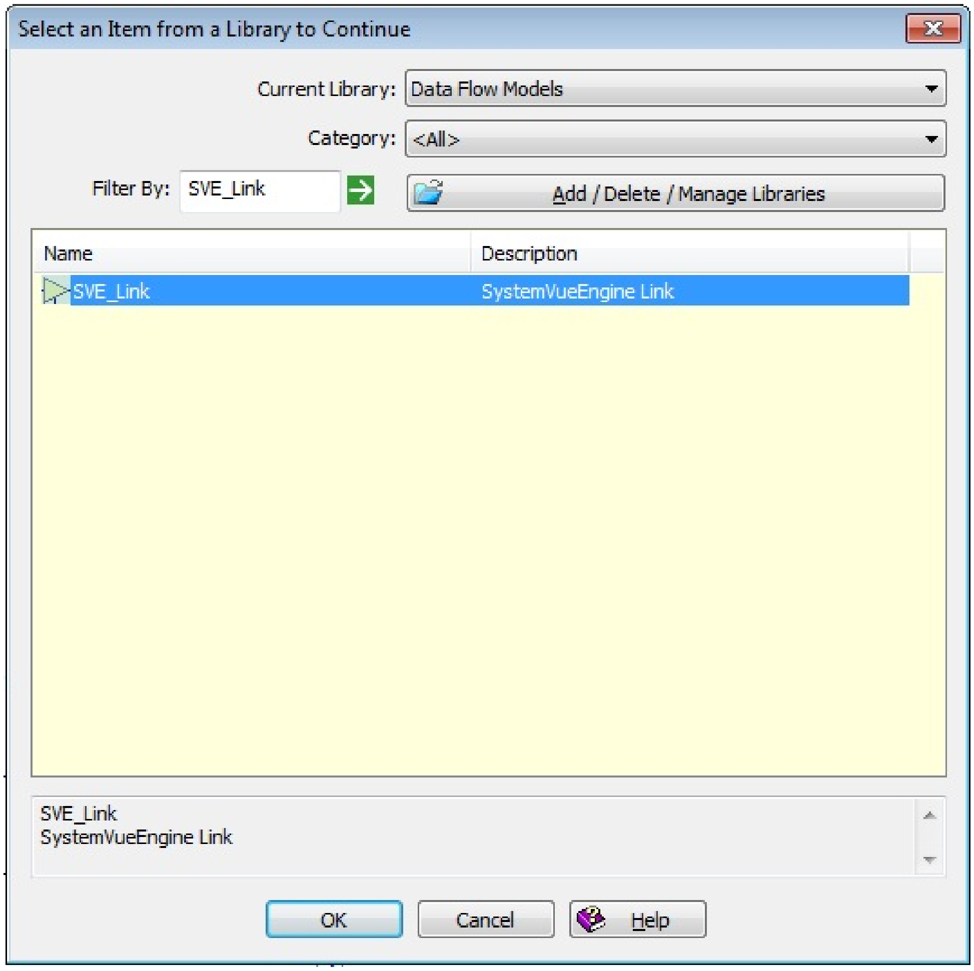
The minimum requirement for a valid VTB workspace is one data flow analysis component and one associated schematic that contains at most one VTB_link model. While, it is possible to create a workspace from scratch, the flow described in this example shows how to leverage an existing example.
Open VTB_Tutorial example that is located at "Examples\Tutorials\Verification_Test_Bench\VTB_Tutorial.wsv". This workspace contains a simulation model of an 802.11ac transmitter and conducts EVM

measurements on its output signal. The workspace exposes several parameters of different data types which can be used by VTB consumer to configure the simulation models.

## DUT Configuration

In this step, an SVE_Link is added to the DUT:

- Double click on the Mixer component "M3" to open its Part Properties dialog window.
- Click on Manage Models... Manage Models...
- Click on Add Model Add Model
- Click on From Library
- In the Filter By text box, type: *SVE_Link*



Select an Item from a Library to Continue

Current Library: Data Flow Models

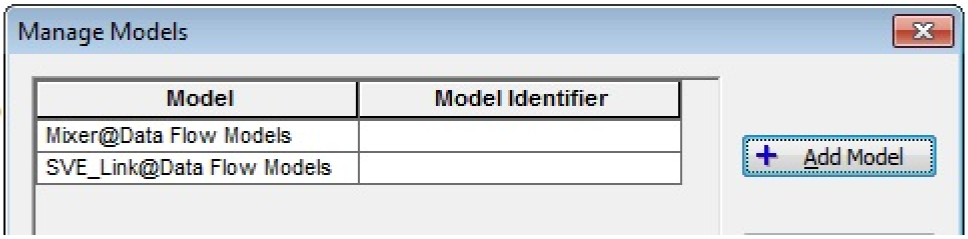
Category: <All>

Filter By: SVE_Link

| Name | Description |
|---|---|
| SVE_Link | SystemVueEngine Link |

SVE_Link
SystemVueEngine Link

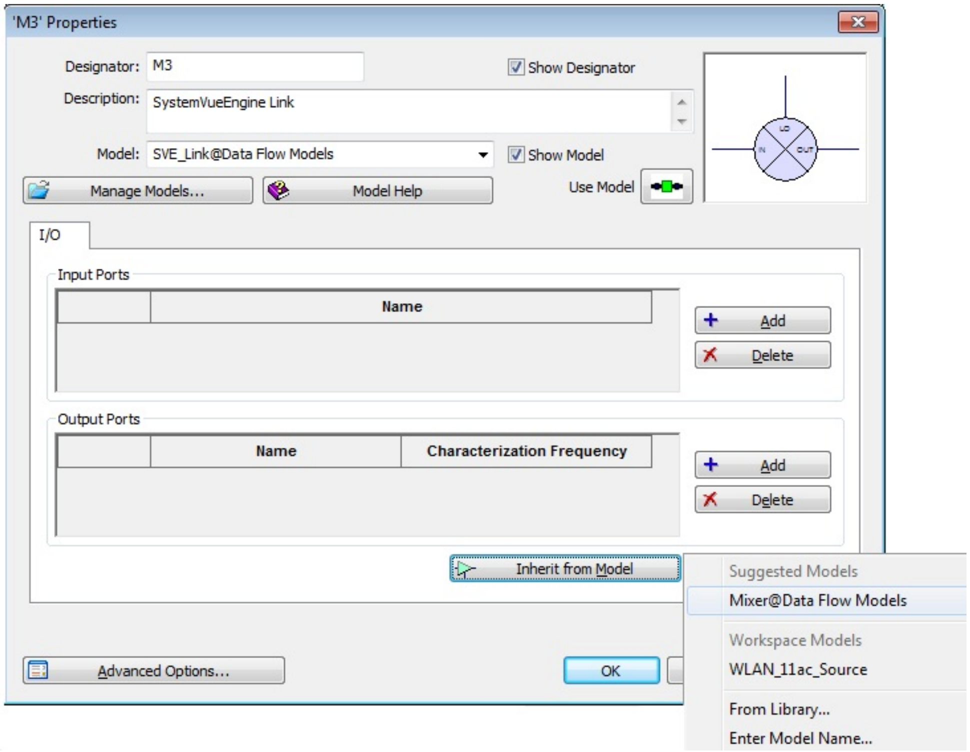
- Select SVE_Link and click OK. A new model "SVE_Link@Data Flow Models" is added to the manage models list. Click OK to close the mange models list.



Manage Models

| Model | Model Identifier |
|---|---|
| Mixer@Data Flow Models | |
| SVE_Link@Data Flow Models | |

- Go to the Model drop list in the Mixer property window and select the added model SVE_Link@Data Flow Models
- Click on Inherit from Model, and then select Mixer@Data Flow Models.



'M3' Properties

Designator: M3

☑ Show Designator

Description: SystemVueEngine Link

Model: SVE_Link@Data Flow Models

☑ Show Model

The input and output ports names of the SVE_Link model are configured automatically to match the original model i.e., Mixer@Data Flow Models.

- Set the Characterization Frequency of the output port to *1e9* and click OK.
- Since the workspace is not at its final stage to be published and used by a VTB consumer, switch the model back to the original one used by the DUT part (Mixer@Data Flow Models) and click OK.

## Workspace Verification

The compatibility of a SystemVue workspace for SystemVue Engine can be verified by following the steps below:

- On SystemVue menu, select Action,
- Click on Validate Workspace for SystemVueEngine....

A new Note item titled "SVE Validation Report" is added to the workspace tree. Note that the report lists the following incompatibilities in the current workspace:

- Part "S2" (Sink@Data Flow Models) does not support output data to File in SystemVueEngine.

To resolve these incompatibilities, follow the steps below:

- Double click on the Sink part "S2" and change Output Data To: setting to 0: Dataset.

  Run the validation procedure again and make sure the "S2" issue is resolved.

## SystemVueEngine Simulation

Use SystemVue Engine executable to examine the simulation results of the developed workspace. To do so, follow the steps below:

- Open a command prompt and go to the <Systemvue installation directory>\bin directory
- Run the following command:
  SystemVueEngine.exe "<Directory path where you saved your copy of VTB_Tutorial.wsv>\VTB_Tutorial.wsv" --output c:\tmp\VTB_Output.adx

NOTE The usage syntax of SystemvueEngine can be obtained by running **SystemvueEngine** with no argument at the command prompt. The usage syntax is:
**Usage: SystemVueEngine <Workspace.wsv> <Analysis Name> --output <Output File> --libpaths <modelDLLlibpath> <modelXMLlibpath> ... <modelEnumXMLlibpath>**

## Simulation Results Verification and Workspace Finalization

The output of SystemVue Engine simulation VTB_Output.adx can be imported by SystemVue by following the steps below:

- On SystemVue menu, select File
- Select Import
- Click on ADX File...

  A new DataSet titled Tx_1Ant_Data is created to the existing workspace. If you have the SVE_Tutorial workspace opened then rename the imported dataset to another name to have it as a reference dataset and run the simulation inside SystemVue and compare the results in both datasets.

  CAUTION A suffix "_1" could be added to the imported dataset name if a similar dataset name is found in the workspace tree.

  NOTE You can use graph to quickly compare the results of both datasets.

### Workspace Publishing

To finalize the workspace, switch the model in DUT part i.e., Mixer "M3" to the "SVE_Link@Data Flow Models" and save the workspace.