Agilent Technologies

SystemVue 2010.01
2010
Simulation

Acknowledgments Mentor Graphics is a trademark of Mentor Graphics Corporation in the U.S. and other countries. Microsoft®, Windows®, MS Windows®, Windows NT®, and MS-DOS® are U.S. registered trademarks of Microsoft Corporation. Pentium® is a U.S. registered trademark of Intel Corporation. PostScript® and Acrobat® are trademarks of Adobe Systems Incorporated. UNIX® is a registered trademark of the Open Group. Java™ is a U.S. trademark of Sun Microsystems, Inc. SystemC® is a registered trademark of Open SystemC Initiative, Inc. in the United States and other countries and is used with permission. MATLAB® is a U.S. registered trademark of The Math Works, Inc.. HiSIM2 source code, and all copyrights, trade secrets or other intellectual property rights in and to the source code in its entirety, is owned by Hiroshima University and STARC.

Errata The SystemVue product may contain references to "HP" or "HPEESOF" such as in file names and directory names. The business entity formerly known as "HP EEsof" is now part of Agilent Technologies and is known as "Agilent EEsof". To avoid broken functionality and to maintain backward compatibility for our customers, we did not change all the names and labels that contain "HP" or "HPEESOF" references.

Warranty The material contained in this document is provided "as is", and is subject to being changed, without notice, in future editions. Further, to the maximum extent permitted by applicable law, Agilent disclaims all warranties, either express or implied, with regard to this manual and any information contained herein, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. Agilent shall not be liable for errors or for incidental or consequential damages in connection with the furnishing, use, or performance of this document or of any information contained herein. Should Agilent and the user have a separate written agreement with warranty terms covering the material in this document that conflict with these terms, the warranty terms in the separate agreement shall control.

Technology Licenses The hardware and/or software described in this document are furnished under a license and may be used or copied only in accordance with the terms of such license.

Portions of this product is derivative work based on the University of California Ptolemy Software System.

In no event shall the University of California be liable to any party for direct, indirect, special, incidental, or consequential damages arising out of the use of this software and its documentation, even if the University of California has been advised of the possibility of such damage.

The University of California specifically disclaims any warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The software provided hereunder is on an "as is" basis and the University of California has no obligation to provide maintenance, support, updates, enhancements, or modifications.

Portions of this product include code developed at the University of Maryland, for these portions the following notice applies.

In no event shall the University of Maryland be liable to any party for direct, indirect, special, incidental, or consequential damages arising out of the use of this software and its documentation, even if the University of Maryland has been advised of the possibility of such damage.

The University of Maryland specifically disclaims any warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. the software provided hereunder is on an "as is" basis, and the University of Maryland has no obligation to provide maintenance, support, updates, enhancements, or modifications.

Portions of this product include the SystemC software licensed under Open Source terms, which are available for download at http://systemc.org/ . This software is redistributed by Agilent. The Contributors of the SystemC software provide this software "as is" and offer no warranty of any kind, express or implied, including without limitation warranties or conditions or title and non-infringement, and implied warranties or conditions merchantability and fitness for a particular purpose. Contributors shall not be liable for any damages of any kind including without limitation direct, indirect, special, incidental and consequential damages, such as lost profits. Any provisions that differ from this disclaimer are offered by Agilent only.
With respect to the portion of the Licensed Materials that describes the software and provides instructions concerning its operation and related matters, "use" includes the right to download and print such materials solely for the purpose described above.

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 Agilent 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.

Create Your First Data Flow Simulation

A workspace file in SystemVue is the basic database used to store anything related to a user *design* (users), *data* (users), *graphs* (users), *analysis* (users), *equations* (users) etc. A Data Flow simulation in SystemVue requires at least two basic components in a workspace.

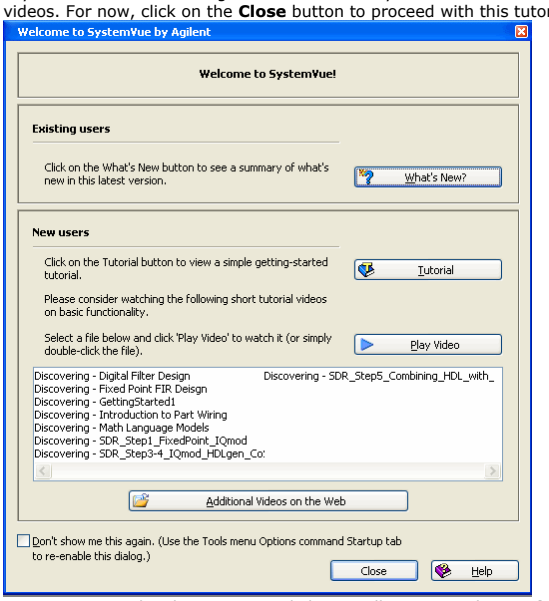
1. ***A Design*** (users): This is used to define how the data flow parts connect together to form a complete System. The Schematic is the graphical view of the design. You can also view the Design as a list of parts (the PartList tab).
2. ***A Data Flow Analysis*** (users): This is the simulation controller that determines sample rate and start time. For more details about data flow technology please read *Introduction to Data Flow Simulation* (sim).

In this tutorial, we will create a simple design using a *sine generator* (algorithm) and a *sink* (algorithm), run the simulation, and view the output in a *graph* (users).

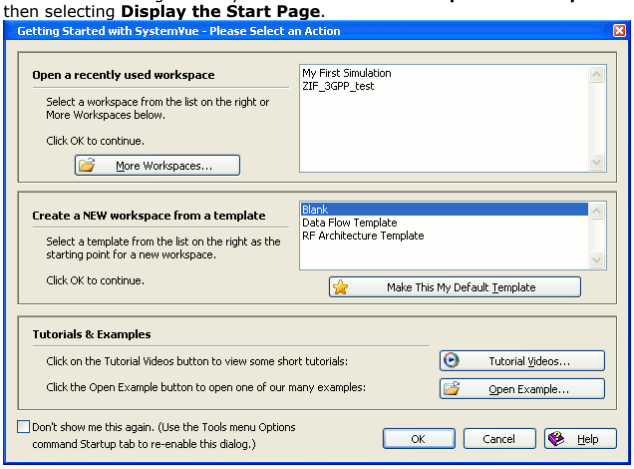
To build a simple simulation, let's start with the Blank workspace containing a Blank Design and a *Data Flow Analysis* (users).

Phase 1: Start SystemVue with a Blank Template

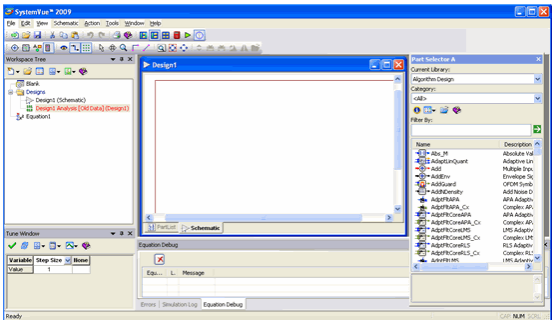
- Start SystemVue. If you have any problem in starting SystemVue then consult *Installation* (start) documentation to make sure the SystemVue has been installed properly.
- If you see a welcome dialogue as shown below you can also look into the **Tutorial** videos. For now, click on the **Close** button to proceed with this tutorial.



- A Getting Started with SystemVue dialogue will appear. Select a **Blank** template in this dialogue box as shown below and click **OK** button. Optionally, you could open other *templates* (users), watch tutorial videos, open *examples* (examples) shipped with SystemVue, or open a recently used workspace using this dialogue box. If you don't see this dialogue then you can enable it in **Startup** of **Tools->Options** and then selecting **Display the Start Page**.

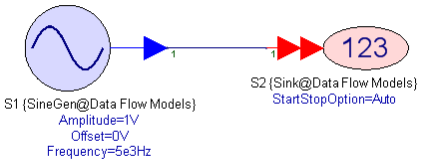


- The SystemVue will open the **Blank** template as shown below. The Blank template includes an schematic *Design* (users) with name **Design1 (Schematic)**, a *Data Flow Analysis* (users) with name **Design1 Analysis**, and an *Equation* (users) with name **Equation1**. Although we will not be using *Equations* (users) in this tutorial, it is useful to know that the *Equations* (users) are a powerful tool that enables post processing of data, control over inputs to simulations, and definition of user-defined custom models. You may look at *Equations documentation* (users) for more details on how to use equations.



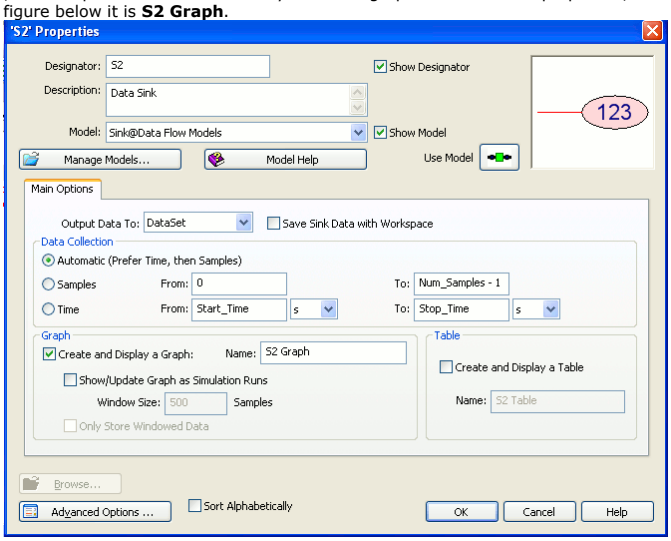
Phase 2: Create the System Design

- To add a **Sine Generator** to the design.
  1. Click inside of the schematic. The schematic window should highlight and the part selector may display (depending on its last state).
  2. If you have no part selector (usually it's docked on the right of the screen) click the Show Part Selector button in the the *Schematic Toolbar* (users).
  3. Under *Current Library:*, switch to the Algorithm Design library. You may have many libraries available.
  4. 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.
  5. Now, click the *SineGen* (algorithm) part and then click anywhere in the schematic to place the Sine Generator
- To add a **Data Sink** to the schematic
  1. Change the **sine** to **sink** in the Filter By: field and press Enter.
  2. Click the *Sink* (algorithm) part in the selector then click in the schematic to place it. If you click directly on the SineGen output pin the two parts will connect.
  3. If you didn't connect them automatically, we need to connect the Generator to the Sink. See *connecting parts* (users).
     - Connect using a line (connector)
       - Mouse over the SineGen output pin. The cursor will change 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 will turn green and stick to the SineGen pin as you drag the part.



Note that you don't 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. See *Appendix A Keystroke Commands* (users) for more information.

- To plot the results in a *Graph* (users) automatically after simulation double click the **Sink** to open its Properties as shown below and check **Create and Display a Graph** ; this will plot the data collected by sink on a graph named in Sink properties, in the figure below it is **S2 Graph**.

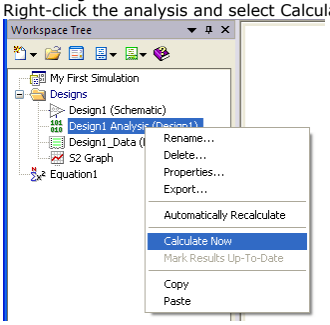


Phase 3: Run the Simulation

- To **run a simulation**, click on the Run Analysis Button in the the *Schematic Toolbar* (users). This will run the simulation, store the data collected by the *Sink* (algorithm) in a *Dataset* (users) named Design1_Data, and also create a graph

named **S2 Graph** as we have set it in *Sink* (algorithm) properties. For further details about dataset, read *Examining Datasets* (users) documentation. After running the simulation the **S2 Graph** will be 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 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

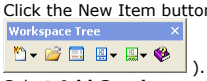


  - Click the calculator button in the top toolbar (or press F5) (this only updates out of date items).
  - Double-click the analysis (opening it to change it) and when done, click the Calculate Now button in the Analysis dialog.

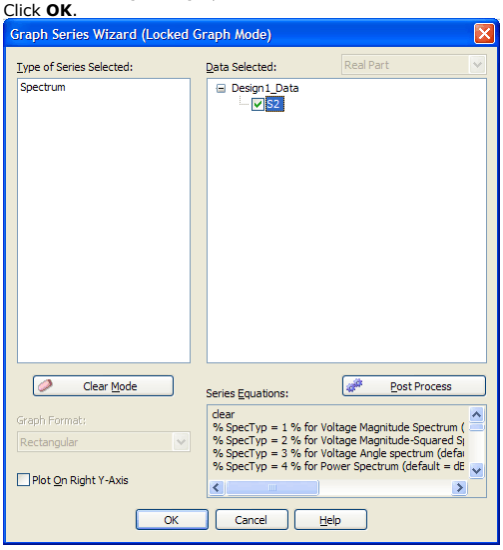
Phase 4: Creating Additional Graphs

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* (users) and *Graphs* (users) using this data, you can postprocess it, and you can compare data from multiple datasets/runs.

To create a graph do the following:

1. Click the New Item button ( ) on the Workspace Tree toolbar ().

2. Select **Add Graph...**, and the *Graph Series Wizard* (users) window will appear.
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* (users) window will appear.
6. If desired, change the graph **Name**, and add a title to the **Graph Heading**.
7. Click **OK**.



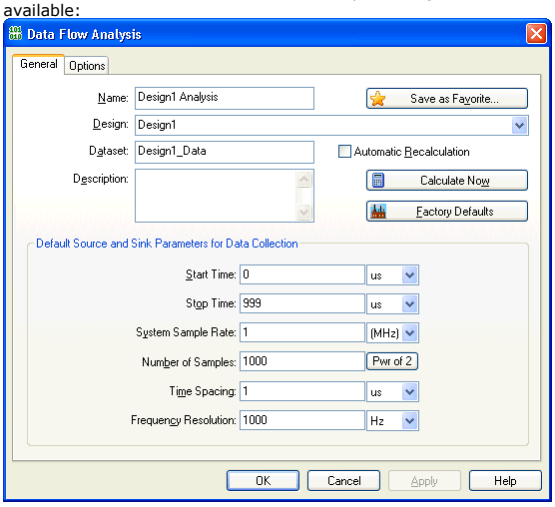
For more details about datasets read *Examining Datasets* (users) documentation. To learn about creating tables from data in dataset, read *Creating Tables* (users).

Setting up the Data Flow Analysis

The Data Flow *Analysis* (users) is the main engine behind the data flow simulation. In the Blank template (explained in *Create Your First Data Flow Simulation* (sim) section), it is added with name **Design1 Analysis**. You may double click on this Data Flow Analysis to open its properties. For more details about data flow simulation technology please read *Introduction to Data Flow Simulation* (sim).

Basic settings for the Data Flow Analysis

- The General tab view of the Data Flow Analysis dialog box shows the basic settings available:

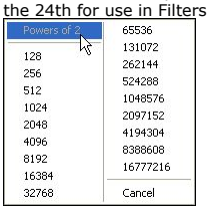


- The fields are populated with defaults for convenience:
  - **Name:** Design1 Analysis (DF1, etc.)
  - **Design:** Design1
  - **Dataset:** Design1_Data (DF1_Data, etc.) (Note:dataset name does not change automatically if you change design name.)
  - **Description:** (optional)

Default Source and Sink Parameters for Data Collection

- Similarly, the timing parameter fields are populated with convenient defaults:
  - **Start Time:** 0 us
  - **Stop Time:** 999 us
  - **System Sample Rate:** 1 MHz
  - **Number of Samples:** 1000
  - **Time Spacing:** 1 us
  - **Frequency Resolution:** 1000 Hz

- The timing parameters are dependent on each other and will automatically change as follows:
  - **Start Time:** this will stay fixed unless it is explicitly changed.
  - **Stop Time:** this will change if either Number of Samples or Frequency Resolution is modified.
  - **System Sample Rate:** will be affected only if its inverse (System Sample Rate) is modified.
  - **Number of Samples:** this will be affected if anything at all is modified.
  - **Time Spacing:** will be affected only if its inverse (Time Spacing) is modified.
  - **Frequency Resolution:** will be affected by changes in Start and Stop Times or in Number of Samples.
- The Number of Samples can conveniently be set to any power of 2 from the 7th to the 24th for use in Filters by clicking to button labeled "Pwr of 2".

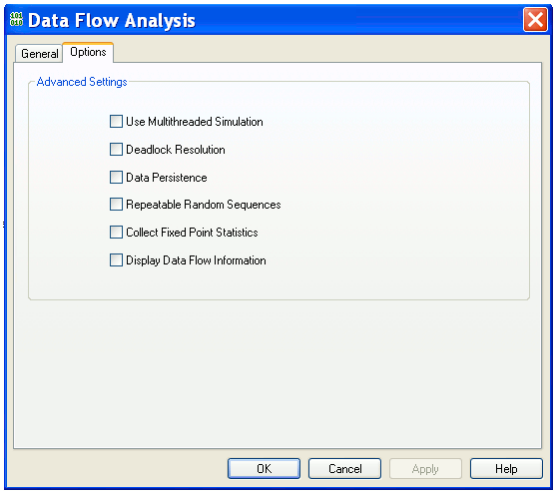


Other settings for the Data Flow Analysis

- The other settings for the Data Flow Analysis are described here:
  - **Factory Defaults:** this will set the timing parameters back to their defaults.
  - **Save as Favorite:** the current set of timing parameters will become the default for a new analysis.
  - **Calculate now:** the analysis will now run using the current set of timing parameters.
  - **Automatic Recalculation:** checking this box will cause the analysis to be run any time it is out of date and data is requested from it (for example, by a graph, table, or equation that uses data produced by it).

Options tab for the Data Flow Analysis

- Advanced Settings for the Data Flow Analysis are available under the Options tab and these too are mostly self explanatory:



- **Use Multithreaded Simulation:** checking this box will allow the analysis to run using the multithreaded scheduler. The multithreaded scheduler exploits parallelism that is present in the data flow schematic - examples of parallelism are parallel paths in a design or a highly multirate design. The multithreaded scheduler adds overhead during the simulation, in many cases this overhead is minimal compared to the performance gains achieved. In some cases, where there is minimal parallelism that can be exploited, the simulation will be slightly slower. For some fine grain simulations (where the parts on the schematic are simple parts such as add, multiply and gain) - the overhead can be very large. By default, the multithreaded scheduler is off. Note, there are few parts that do not support multithreaded simulations. These include VSA 89600 parts, all fixed-point parts, the MathLang part and the MATLAB cosimulation part. In these cases, the multithreaded scheduler will automatically be deactivated.
- **Deadlock Resolution:** checking this box will allow the simulator to attempt to resolve deadlocks by intelligent insertion of delays. Otherwise, no automatic insertion of delays will be performed and a deadlock will result in an error. See *Deadlock and Deadlock Resolution* (sim)
- **Data Persistence:** checking this box will cause the data produced by instances of the Sink with the default setting to be saved when the workspace is saved. Otherwise, data is discarded to save space.
- **Repeatable Random Sequences:** checking this box will result in repeatable random sequences from models that make use of random numbers, e.g. *RandomBits* (algorithm), *IID_Uniform* (algorithm), *IID_Gaussian* (algorithm), etc. Each model that needs to generate random numbers is provided a unique seed that is the same for all simulation runs. The seed is a function of the Data Flow Analysis name, the Design name, and the part instance name, so changing these names will change the random sequences generated by these models. If the box is not checked then each model that needs to generate random numbers is provided a unique seed that is different for each simulation run.
- **Collect Fixed Point Statistics:** checking this box enables any **fixed point** point models (in the associated design) to collect fixed point statistical analysis data. The statistical analysis data is shown in the *Fixed Point Analysis Table* (sim). If you are writing your own Custom C++ model containing fixed point inputs and/or outputs then please read *Writing a Fixed Point Model for Fixed Point Analysis* (users) for your model to work properly with this option.
- **Display Data Flow Information:** checking this box will collect data flow related information into a variable "DataFlowInfo" in the dataset associated with the analysis. A table named "(Dataset)_DataFlowInfo" will be created automatically to display the collected data flow information. Such information is useful for diagnostic and understanding data flow operations. Please refer to the following topic about Reading Data Flow Information Table.

Reading Data Flow Information Table

The screenshot below illustrates the data flow information table:

Design1_Data_DataFlowInfo

Part	Model	Domain	Repetition	Port	Direction	DataFlowRate	Delay	Type	SampleRate	TimeStep	StartTime	SyncSamples
S1	SineGen@Data Flow Models	Timed	1									
				output	Output	1	0	REAL	1.00000000e+006	1.00000000e-006	0.00000000e+000	0
G1	GainDbl@Data Flow Models	Untimed	1									
				input	Input	1	0	REAL	N/A	N/A	N/A	N/A
				output	Output	1	0	REAL	N/A	N/A	N/A	N/A
S2	Sink@Data Flow Models	Timed	1									
				input#1	Input	1	0	REAL	1.00000000e+006	1.00000000e-006	0.00000000e+000	0

The information associated with each part is collected in rows. Distinct parts are separated by empty rows. Distinct graph regions are separated by "---" lines. Each column represents an information item:

- **Part**: the name of a part.
- **Model**: the model associated with the part.
- **Domain**: a model can be either **Timed** or **Untimed**. See *Timing Method* (sim).
- **Repetition**: the data flow repetition count of the part after solving the balance equations of the system. See *Synchronous Data Flow* (sim).
- **Port**: the name of the port associated with the model. If a model has multiple ports, there will be multiple rows grouped together under the part. For example, part "G1" in the above table has ports "input" and "output".
- **Direction**: the direction of a port can either be **Input** or **Output**.
- **DataFlowRate**: the data flow **production rate** associated with the output port or the data flow **consumption rate** associated with the input port. It is a positive integer for most of the models. See *Synchronous Data Flow* (sim). The only exceptions are the *input* port of *DynamicPack_M* (algorithm) and the *output* port of

DynamicUnpack_M (algorithm). Under these two exceptions, the value is "Dynamic" meaning that the data flow rate can change dynamically. See *Introduction to Dynamic Data Flow Simulation* (sim).
- **Delay**: the data flow sample delay associated with the connection of the port. See *Synchronous Data Flow* (sim).
- **Type**: the data type for the port. See *Using Data Types* (sim).
- **SampleRate**: the resolved sampling rate associated with the port for a timed model. See *Timed Synchronous Data Flow* (sim). "N/A" if the model is untimed.
- **TimeStep**: the time step (inverse of the sampling rate) associated with the port for a timed model. See *Timed Synchronous Data Flow* (sim). "N/A" if the model is untimed.
- **StartTime**: the time stamp of the first incoming sample to the input port or the time stamp of the first outgoing sample from the output port for a timed model. See *Timing Method for Timed Models* (sim). "N/A" if the model is untimed.
- **SyncSamples**: the number of samples inserted for synchronizing the first incoming time stamps for multiple-input timed model. See *Timing Method for Timed Models* (sim). "N/A" if the model is untimed.

The relative data transfer ratio between any two ports *A* and *B* in the system can be derived as: (**Repetition** of the part of port *A* * **DataFlowRate** of port *A*) / (**Repetition** of the part of port *B* * **DataFlowRate** of port *B*).

Accessing Data Flow Analysis Settings from Equations

If you have noticed above, the Blank template has added **Equation1** to the workspace tree. The *Equations* (users) are a powerful tool that enable post processing of data, control over inputs to simulations, and definition of user-defined custom models. You may look at *Equations documentation* (users) for more details on how to use equations.

The values for many of the parameters that you set up for a Data Flow Analysis are accessible through Equations. The values for these variables are updated under two circumstances:

- You click OK in the Data Flow Analysis dialog box.
- At the beginning of a particular Data Flow simulation run.

The following table lists Data Flow Analysis parameters and their corresponding variable names that can be accessed from equations:

Data Flow Parameter	Variable Name
Start Time	Start_Time
Stop Time	Stop_Time
System Sample Rate	Sample_Rate
Number of Samples	Num_Samples
Time Spacing	Time_Spacing
Frequency Resolution	Freq_Resolution

On the Advanced Settings tab, checking or un-checking Data Persistence sets the value of the variable named Data_Persistence.

About the Data Flow Simulator

SystemVue provides the capabilities you need to evaluate and design modern communication systems and related products. Today's designs call for implementing DSP algorithms in an increasing number of portions in the total communications system path, from baseband processing to adaptive equalizers and phase-locked loops in the RF chain.

Using the Data Flow simulator you can:

- Find the best design topology using state-of-the-art technology with a large number of behavioral DSP and communication systems models
- Integrate intellectual property from previous designs
- Reduce the time-to-market for your products

SystemVue features:

- State-of-the-art data flow simulation technology for mixed baseband and RF systems with dynamic behavior possibly involved
- Easy-to-use interface for adding and sharing custom models
- Interface to test instruments
- Data display with post-processing capability

Outline

The following pages introduce SystemVue's data flow simulation technology in different topics:

1. *Introduction to Data Flow Simulation* (sim): this page introduces data flow fundamentals, theory, and simulation operations
2. *Timing Method* (sim): this page describes how SystemVue simulates time in data flow semantics
3. *Introduction to Dynamic Data Flow Simulation* (sim): this page introduces SystemVue's approach to model and simulate dynamic behavior
4. *Using Data Types* (sim): this page discusses various data types in SystemVue
5. *Envelope Signal* (sim): this page introduces SystemVue's envelope simulation technology for analog and RF systems

Introduction to Data Flow Simulation

SystemVue simulation environment is built based on **data flow models of computation**.

Data Flow Models of Computation

In the data flow modeling paradigm, the computational behavior of a system is represented as a **directed graph**

$G = (V, E)$

. A vertex (which is called a **block** or a ***Part*** (users) in SystemVue)

$v \in V$

represents a computational module or a hierarchically nested subgraph (which is called a **subnetwork** in SystemVue). A directed edge (which is called a **connection** in SystemVue)

$e \in E$

represents a **FIFO** (first-in-first-out) buffer that carries **data samples** from its source block

$src(e)$

to its sink block

$snk(e)$

. An edge

e

can have a non-negative integer **delay**

$del(e)$

associated with it, and the delay value specifies the number of initial samples that are buffered on the edge before the graph starts execution.

Data flow graphs operate based on **data-driven** execution: a block

v

can execute (**fire**) only when it has sufficient numbers of data samples on all of its input edges

$in(v)$

. When firing,

v

consumes a certain number of samples from its input edges, executes its computation, and produces a certain number of samples on its output edges

$out(v)$

.

Synchronous Data Flow

Synchronous Data Flow (**SDF**) [1] is the most mature data flow model of computation. In SDF, the number of samples produced onto an edge

e

by a firing of

$src(e)$

is restricted to a constant positive integer that must be known before simulation; this integer is referred to as the **production rate** of

e

and is denoted as

$prd(e)$

. Similarly, in SDF, the number of samples consumed from an edge

e

by a firing of

$snk(e)$

is restricted to a constant positive integer that must be known before simulation; this integer is referred to as the **consumption rate** of

e

and is denoted as

$cns(e)$

. In SystemVue, we say that an edge

e

represents a **multirate** connection if

$prd(e) \neq cns(e)$

.

The constant integer restriction makes SDF especially suited to modeling multirate systems and benefits SDF with the compile-time capabilities such as deadlock detection, bounded memory determination, and static scheduling.

The production and consumption rates of each individual SystemVue model are specified in its documentation page.

Scheduling SDF Graphs

Before execution, a **schedule** of a data flow graph is computed. Here, a schedule means a sequence of block firings, or in general, refers to any static or dynamic mechanism for executing blocks in a data flow graph. An SDF graph

$G = (V, E)$

has a valid schedule (is **consistent**) if it is **free from deadlock** (see Deadlock and Deadlock Resolution) and it is **sample rate consistent** — that is, it has a **periodic schedule** that fires each block at least once and produces no net change in the number of samples on each edge [2]. In more precise terms,

G

is sample rate consistent if there is a positive integer solution to the **balance equations**:

$\forall e \in E,\ prd(e) \times \boldsymbol{x}[src(e)] = cns(e) \times \boldsymbol{x}[snk(e)].$

Let

$\boldsymbol{x}[v]$

denote the number of firings of block

v

in a schedule, and suppose there is a positive integer solution to the vector

$\boldsymbol{x}$

. Then the balance equations can be easily interpreted as follows: for each edge, the total number of samples produced onto the edge is equal to the total number of samples consumed from the edge in one iteration of a complete schedule. Such balance of data production and consumption allows the schedule to be executed over and over again, within bounded memory.

When there exists solutions to the vector

$\boldsymbol{x}$

, the minimum positive integer solution is called the **repetitions vector** of the graph

G

, and is denoted by

$\boldsymbol{q}_G$

. For each block

v

,

$\boldsymbol{q}_G[v]$

is referred to as the **repetition count** of

v

. A **valid minimal periodic schedule** (which is abbreviated as a **schedule** hereafter in this documentation) is then a sequence of block firings in which each block

v

is fired

$\boldsymbol{q}_G[v]$

times, and the firing sequence obeys the data-driven restriction imposed by the SDF graph.

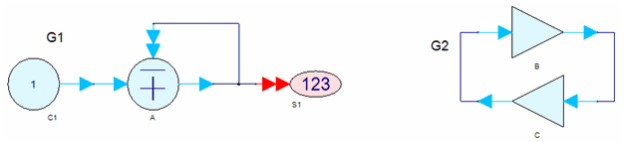
In SystemVue, a schedule is computed before simulation, and during simulation, the schedule is executed iteratively until certain termination conditions are satisfied (see Simulation Control).

It is user's responsibility to construct a **consistent** system that is **sample rate consistent** and **deadlock free**.

When the user's system is sample rate inconsistent, SystemVue will issue error messages to help identify locations where users can adjust, insert, and/or remove blocks to balance the data flow production and consumption rates.

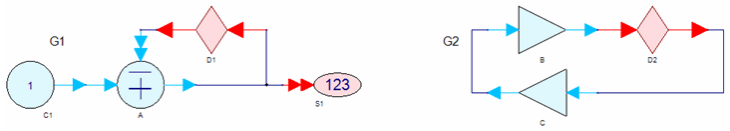
Deadlock and Deadlock Resolution

In a data flow graph, deadlock occurs when there exist cycles (or in more precise term, strongly connected components) without sufficient numbers of initial samples (delays). The following graphs illustrate simple conditions that cause deadlock. Based on data-driven execution, in graph G1, A cannot fire because one of its inputs is waiting for a data sample from itself. Similarly, in graph G2, neither B nor C can fire because they are waiting for data samples from each other.



The way to resolve deadlock is to introduce sufficient numbers of delays (initial samples) on the proper edges in the cycles. The initial data samples are buffered before a graph stars execution, and therefore, allow one of the deadlocked blocks to start firing.

SystemVue *Delay* (algorithm) model inserts initial samples that can be buffered on a connection before simulation. The number of initial samples is specified as parameter N in *Delay* (algorithm), and by default, the initial value is zero. The following graphs illustrate how to solve deadlock using the *Delay* (algorithm) model.



To avoid deadlock, all feedback loops (cycles) must have sufficient delays. When user's system is deadlocked, SystemVue will issue error messages to help identifying locations where users can insert delays.

If **Deadlock Resolution** is checked in *Data Flow Analysis Options* (sim), SystemVue will automatically insert delays (initial samples), if necessary, to avoid deadlock, and will identify the locations and the numbers of inserted samples.

Deadlock Resolution may modify system's behavior. It is user's responsibility to make sure the auto-inserted samples have correct behavior.

In addition to *Delay* (algorithm), SystemVue provides a number of specialized models that introduce delays for special needs. *InitDelay* (algorithm) can specify a specific value for initial samples. *DelayFxp* (hardware) can insert delays for fixed-point data type. *DelayEnv* (algorithm), in most cases (unless the Delay parameter is 0 and there is no control signal), inserts one delay to the RF connection, and therefore, can be used to solve deadlock in unit-rate cycles (where production and consumption rates are all equal to one).

Timed Synchronous Data Flow

Timed Synchronous Data Flow (TSDF) [3] operates on top of SDF. In TSDF, the flow of data samples is viewed as discrete-time signal or sampled version of continuous-time signal. To represent such time-domain signal in data flow, TSDF introduces the concept of sampling rate and time. In a TSDF graph

$G = (V, E)$

, an edge

$e \in E$

can be associated with **sampling rate**

$f_s(e)$

. With the sampling rate defined for a connection, the data samples flow through the connection can be **timed** by associating with time stamps that increase continuously by a constant **time step** (i.e., the inverse of the sampling rate).

Given a consistent (see the definition of "consistent" in Scheduling SDF Graphs) TSDF graph

$G = (V, E)$

, there exists a positive constant

C_G

such that

$$\forall e \in E,\ f_s(e) = prd(e) \times \boldsymbol{q}_G[src(e)] \times C_G = cns(e) \times \boldsymbol{q}_G[snk(e)] \times C_G.$$

The above expression is referred to as the **TSDF sampling rate equations** and can be further derived into the following form:
For each block

$v \in V$

,

$$\begin{cases} \forall e \in in(v), & f_s(e)/cns(e) = \boldsymbol{q}_G[v] \times C_G; \\ \forall e \in out(v), & f_s(e)/prd(e) = \boldsymbol{q}_G[v] \times C_G. \end{cases}$$

Suppose

C_G

is interpreted as the average rate for executing a complete minimum periodic schedule. Then

$q_G[v] \times C_G$

can represent the average execution rate for block

v

, and

$f_s(e)$

can represent the average rate at which

$src(e)$

generates samples, and equivalently, the average rate at which

$snk(e)$

consumes samples.

Given a TSDF graph

G

, once

C_G

is defined (or equivalently, once the sampling rate for a particular edge is defined), the sampling rates for all edges can be computed based on the TSDF sampling rate equations. In SystemVue, sampling rates are further used by the *Timing Method* (sim) to determine the timing behavior of the system.

Different connections may have different sampling rates due to multirate properties of the system.

Sampling Rate Resolution

SytemVue provides a set of models that can be used to set the sampling rates in a system. The source model (see *Sources Category* (algorithm)) with SampleRateOption "Timed from SampleRate" can set the sampling rate at its output connection. In addition, the *SetSampleRate* (algorithm) model can set the sampling rate for the incoming samples at its output connection.

SystemVue uses the following rules to resolve system sampling rates.

1. If at least one model (e.g., a *source* (algorithm) model with "Timed from SampleRate" option or a *SetSampleRate* (algorithm) 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.

   When there are multiple blocks 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.
2. Else if there exists at least one model that requires sampling rate (i.e., timed model, see *Timing Method* (sim)) and there exists at least one source block (i.e., a block with no input port), SystemVue will set the "System Sample Rate" (defined in the *Data Flow Analysis* (sim)) 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.
3. Else if there exists at least one model that requires sampling rate (i.e., timed model, see *Timing Method* (sim)) and there is no source block (e.g., a cycle), SystemVue will set the "System Sample Rate" (defined in the *Data Flow Analysis* (sim)) to the connections 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.
4. Else, only models that deal with pure numeric computations (i.e., numeric model, see *Timing Method* (sim)) exist in a system. In this case, SystemVue treats the system as a pure SDF graph and does not compute sampling rates.

Simulation Control

SystemVue uses *Sinks Category* (algorithm) to control the length of a simulation. Users can specify the start and stop conditions in each individual sink for data collection. A sink stops collecting data after its stop condition is met. Once all sinks meet their stop conditions, SystemVue stops simulation. For simulation efficiency, SystemVue data flow scheduler may intelligently choose to disable (not execute) a model if there is no sink demanding data.

For example, an user can choose "Time" option in the *Sink* (algorithm) model and specify data collection from *TimeStart* to *TimeStop*. Let *t0* denote the time stamp associated with the first incoming sample (*t0* may be larger than 0, see *Timing Method* (sim)). If *t0* < *TimeStart*, the *Sink* (algorithm) starts to collect samples when the associated time stamp is larger than or equal to *TimeStart* and stops collecting samples when the associated time stamp is larger than *TimeStop*. On the other hand, if *t0* >= *TimeStart*, the *Sink* (algorithm) collects samples starting from the first one (*t0*) and stops collecting samples when the associated time stamp is larger than *TimeStop*. In addition, an user can also choose "Samples" option in the *Sink* (algorithm) model and specify data collection from *SampleStart* to *SampleStop*. In this case, the *Sink* (algorithm) collects incoming data from the *SampleStart* -th sample to the *SampleStop* -th sample.

References

1. E. A. Lee and D. G. Messerschmitt, "Synchronous dataflow," *Proceedings of the IEEE*,

vol. 75, no. 9, pp. 1235-1245, Sept. 1987.
2. S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, *Software Synthesis from Dataflow Graphs*. Kluwer Academic Publishers, 1996.
3. J. L. Pino and K. Kalbasi, "Cosimulating synchronous DSP applications with analog RF circuits," in *Proceedings of the IEEE Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, Nov. 1998.

Timing Method

The SystemVue Data Flow simulator operates based on the **Synchronous Data Flow** (**SDF**) model of computation (see *Introduction to Data Flow Simulation* (sim)). In SDF, connections represent **first in first out** (FIFO) buffers for buffering data samples between models. A model can execute (fire) whenever it has a sufficient number of data samples in its input buffers. When firing, a model consumes certain number of data samples from its input buffers and produces a certain number of data samples to its output buffers. The numbers of samples produced and consumed, in a model firing, are restricted to constant positive integers throughout a simulation. These numbers are referred to as the **production rate** and **consumption rate** associated with the model. **SDF has no notion of time**, and models work purely on **flows of data samples**.

Timed Synchronous Data Flow (**TSDF**) operates on top of SDF. It introduces the concept of time. In TSDF, the flow of data samples on a connection, represents discrete-time signal or sampled version of continuous-time signal. The data samples (in the direction of flow) are associated with **time stamps** that can increase continuously by a **constant time step**. The inverse of the time step represents the **sampling rate** of the continuous-time signal on the particular connection. TSDF is able to pre-compute the sampling rates associated with graph connections (see *Introduction to Data Flow Simulation* (sim)).

SystemVue data flow simulation technology supports SDF and TSDF simultaneously. The cooperation of SDF and TSDF gives SystemVue an unique advantage of providing two domains for models, **numeric (untimed)** and **timed**, representing two distinct sets of timing behaviors. In general, SystemVue models are implemented either in the **numeric (untimed)** domain or the in **timed** domain.

Individual model documentation page specifies the associated domain information of the model.

SystemVue **numeric (untimed) models** primarily represent pure numeric processing for algorithm design, for example, mathematical operations (e.g., see *Math Scalar Category* (algorithm)), discrete Fourier transform (*FFT_Cx* (algorithm)), encoders (e.g., *GrayEncoder* (algorithm)) and decoders (e.g., *ViterbiDecoder* (algorithm)) , Math Language processing model (*MathLang* (algorithm)), etc. SystemVue numeric models decouple pure numeric processing from various implementation approaches that may impose certain timing constraints based on implementation details.

In contrast, SystemVue **timed models** generally represent discrete-time sampled version of analog/RF circuits (e.g., *Mixer* (algorithm) or see *Analog RF Category* (algorithm) library), envelope signal processing (see *Envelope Signal* (sim) introduction) that operates on notion of time (e.g., *MpyEnv* (algorithm)), or digital components that model specific implementations with timing constraints (e.g., fixed-point models in *Hardware Design Library* (hardware)).

Numeric and timed models *can* be mixed together in SystemVue Data Flow simulation. The order of model executions is determined by SDF model of computation. Data samples are **timed** (associated with **time stamps**) in such a way that the timing for the overall system is **causal**, and the time stamps match numeric semantics for numeric models and timed semantics for timed models.

Timing Method for Numeric (Untimed) Models

A numeric (untimed) model represents pure numeric computation and operates on flows of data samples. However, in order to make numeric and timed models work together, numeric models must propagate timing information in a causal way without imposing additional timing constraints. In SystemVue, the time stamps associated with the output samples of a numeric model should represent the earliest possible time those samples are available, assuming that the numeric model can only execute after all the input samples are available, and once the input samples are available, it can immediately produce output samples. Based on this definition, the timing rules for numeric models are stated as follow:

1. The output samples of a numeric model in a particular firing all have the same time stamp equal to the maximum time stamp of the input samples consumed in the same firing. This rule ensures the causality for numeric models and does not add any latency.
2. An untimed (numeric) source (i.e., a source model with "UnTimed" option) always generates data samples with time stamp = 0. In SystemVue, we assume sources always start at time 0, and because there is no timing requirement for numeric sources, all the samples generated by an untimed source can be treated available at time 0.

Since numeric semantics has no concept of time, the data samples from different input ports of a numeric (untimed) model do not have to align in time. In addition, based on the definition, if the production rate *prd* of a numeric model port is larger than 1, then in each firing, *prd* output samples all have the same time stamp.

Timing Method for Timed Models

A timed model represents analog/RF circuit, discrete-time processing, or digital implementation that involve the notion of time in its behavior. In SystemVue, to meet the timing requirements for timed models, we define that:

1. The input time stamps to a timed model must increase continuously by a constant time step (1 / sampling rate) associated with the input port (connection). A connection to a timed model acts as a special buffer that **buffers** samples that may have non-evenly spaced time stamps (e.g., from numeric models). The buffer then outputs samples at a constant rate that is equal to the sampling rate associated with the connection.
2. For a multiple-input timed model, the first time stamps from different input ports must align to the minimum input time stamp causally and within the precision of a single time step. Such **time alignment** is done by **artificially inserting zero-valued samples**. For example, suppose a timed model has port *A* and *B*. Both of the ports have time step = 1. Suppose the first incoming time stamp of port *A* is 1, and the first incoming time stamp of port *B* is 3. SystemVue will automatically insert 2 samples at port *B* to make the model start at time = 1 causally. Now suppose the first incoming time stamp of port *A* is 1.5, and the first incoming time stamp of port *B*

is 4. SystemVue will automatically insert 3 samples at port *B* to make the models start at time = 1.5 causally.

> Note that in the second case, with 3 samples inserted at port *B*, the resultant first incoming sample at port *B* seems to start at time stamp 1, but the timed model will interpret it as 1.5 to align with the first time stamp at port A. SystemVue may not be able to align the time stamps at exactly the same time instance because the difference is less than one time step.

3. The output time stamps from a timed model must increase continuously by a constant time step associated with the output port (connection). In each firing, the first output sample from each output port has the time stamp equal to the **minimum input time stamp (before possible alignment) plus the phase and/or latency** of the model. For example, suppose the first incoming time stamp to a timed model is 1, and the model has latency 5, then the first output time stamp is 6. *DownSampleEnv* (algorithm) in timed domain is another example. Suppose the down sample factor is 4 and the phase is 1, and suppose the first incoming time stamp is 0, then the first output time stamp is 1 instead of the last phase 3 (which is the case for numeric *DownSample* (algorithm) regardless of the phase).
4. A timed source (i.e., a source model with "Timed" option, either "Timed from SampleRate" or "Timed from Schematic") always generates data samples with continuously increasing time stamps. The time stamps start from 0 and increase continuously by a constant time step associated with the output port (connection).

> Because of the interaction between numeric and timed models, multirate properties and latencies, a signal to a model may start at time larger than 0.

Latency

In SystemVue, **latency** refers to the timing difference between the first input sample and the first output sample of a timed model. The SystemVue Data Flow timing method incorporates the latency behavior of a timed model by using Definition #3 stated in Timing Method for Timed Models. A set of SystemVue timed models incorporates the latency behavior of the actual implementations, e.g., a fixed point model may incorporate latency in terms of number of clock cycles. Please refer to individual model documentation for details.

Timing Behavior for Mixed Numeric and Timed Systems

SystemVue allows users to construct designs with both numeric (untimed) and timed models. For a system containing both numeric and timed models, the overall timing behavior is determined by the interaction of timing methods in individual models.

As described above, a connection in SystemVue acts as a special **buffer** that holds the samples to satisfy different timing requirements for numeric and timed models. The following table illustrates what time stamps the downstream model will see when different domains of models are connected. (In this table, time step is 1, and notation "a@0" represents a data sample with value "a" at time stamp "0".)

Connection	Samples from output port	Samples to input port
Timed to Timed	Output from a timed model --> ... c@2 b@1 a@0	... c@2 b@1 a@0 --> Input to a timed model
Timed to Numeric	Output from a timed model --> ... c@2 b@1 a@0	... c@2 b@1 a@0 --> Input to a numeric model
Numeric to Timed	Output from a numeric model --> ... c@0 b@0 a@0	... c@2 b@1 a@0 --> Input to a timed model
Numeric to Numeric	Output from a numeric model --> ... c@0 b@0 a@0	... c@0 b@0 a@0 --> Input to a numeric model

Special Models in Timing Method

Delay (OutputTiming = EqualToInput)

SystemVue *Delay* (algorithm) represents an untimed **sample delay** model when its OutputTiming option is set to *EqualToInput*. Under this option, a *Delay* (algorithm) model with delay size *N* simply "delays" the incoming signal **in samples** by *N* number of zero-valued samples. The *N* delay samples are inserted before the incoming signal. When OutputTiming is *EqualToInput*, the *Delay* (algorithm) model retains the first time stamp of the incoming signal. Suppose the incoming signal starts at time *t0*. If the downstream is a timed model, then the *Delay* (algorithm) will delay the incoming waveform by *N* time steps, and the signal still arives at the downstream model at time *t0* with initial *N* zero values. If the downsteam is a numeric model, then the *Delay* (algorithm) simply provides *N* samples that are available all at time *t0* (i.e., the first time stamp of the incoming signal).

The following table illustrates what time stamps the downstream model will see when different domains of models are connected with a *Delay* (algorithm) (OutputTiming = *EqualToInput*) in between. (In this table, time step is 1, and notation "a@3" represents a data sample with value "a" at time stamp "3"; and notation "0@3" means a data sample with value "0" at time stamp "3".)

Connection	Samples from output port	*Delay* (algorithm)	Samples to input port
Timed to *Delay* (algorithm) to Timed	Output from a timed model --> ... c@5 b@4 a@3	--> 3 sample delay -->	... c@8 b@7 a@6 0@5 0@4 0@3 --> Input to a timed model
Timed to *Delay* (algorithm) to Numeric	Output from a timed model --> ... c@5 b@4 a@3	--> 3 sample delay -->	... c@5 b@4 a@3 0@3 0@3 0@3 --> Input to a numeric model
Numeric to *Delay* (algorithm) to Timed	Output from a numeric model --> ... c@3 b@3 a@3	--> 3 sample delay -->	... c@8 b@7 a@6 0@5 0@4 0@3 --> Input to a timed model
Numeric to *Delay* (algorithm) to Numeric	Output from a numeric model --> ... c@3 b@3 a@3	--> 3 sample delay -->	... c@3 b@3 a@3 0@3 0@3 0@3 --> Input to a numeric model

Delay (OutputTiming = BeforeInput)

SystemVue *Delay* (algorithm) represents an untimed **initial sample** model when its OutputTiming option is set to *BeforeInput*. Under this option, a *Delay* (algorithm) model simply represents *N* number of zero-valued samples initially queued in the buffer before the incoming signal. When OutputTiming option is *BeforeInput*, the initial *N* samples are timed before (if possible) or equal to the first incoming sample. Suppose the incoming signal starts at time *t0* and the time step is *dt*. If the downstream is a timed model, then the first initial sample arives at the downstream model at time *max(t0 - N*dt, 0)*, followed by the remaining *N-1* initial samples, then the incoming samples. If the downsteam is a numeric model, then the *N* initial samples that are all available at time

*max(t0 - N*dt, 0).*

> The *Delay* (algorithm) with *BeforeInput* option is essential for a timed feedback loop where signal is delayed in time when propagates through the loop due to multirate properties or latencies of the components along the loop. For such feedback loop, *BeforeInput* option helps to solve timing convergence by using "initial samples" (*-N*dt*) to compensate the latency along the loop.

The following table illustrates what time stamps the downstream model will see when different domains of models are connected with a *Delay* (algorithm) (OutputTiming = *BeforeInput*) in between. (In this table, time step is 1, and notation "a@3" represents a data sample with value "a" at time stamp "3"; and notation "0@3" means a data sample with value "0" at time stamp "3".)

Connection	Samples from output port	*Delay* (algorithm)	Samples to input port
Timed to *Delay* (algorithm) to Timed	Output from a timed model --> ... c@5 b@4 a@3	--> 2 sample delay -->	... c@5 b@4 a@3 0@2 0@1 --> Input to a timed model
Timed to *Delay* (algorithm) to Timed	Output from a timed model --> ... c@5 b@4 a@3	--> 4 sample delay -->	... c@6 b@5 a@4 0@3 0@2 0@1 0@0 --> Input to a timed model
Timed to *Delay* (algorithm) to Numeric	Output from a timed model --> ... c@5 b@4 a@3	--> 2 sample delay -->	... c@5 b@4 a@3 0@1 0@1 --> Input to a numeric model
Timed to *Delay* (algorithm) to Numeric	Output from a timed model --> ... c@5 b@4 a@3	--> 4 sample delay -->	... c@5 b@4 a@3 0@0 0@0 0@0 0@0 --> Input to a numeric model
Numeric to *Delay* (algorithm) to Timed	Output from a numeric model --> ... c@3 b@3 a@3	--> 2 sample delay -->	... c@5 b@4 a@3 0@2 0@1 --> Input to a timed model
Numeric to *Delay* (algorithm) to Timed	Output from a numeric model --> ... c@3 b@3 a@3	--> 4 sample delay -->	... c@6 b@5 a@4 0@3 0@2 0@1 0@0 --> Input to a timed model
Numeric to *Delay* (algorithm) to Numeric	Output from a numeric model --> ... c@3 b@3 a@3	--> 2 sample delay -->	... c@3 b@3 a@3 0@1 0@1 --> Input to a numeric model
Numeric to *Delay* (algorithm) to Numeric	Output from a numeric model --> ... c@3 b@3 a@3	--> 3 sample delay -->	... c@3 b@3 a@3 0@0 0@0 0@0 0@0 --> Input to a numeric model

Time Delay

SystemVue *TimeDelay* (algorithm) is a timed model that delays signal **in time**, in contrast to an untimed *Delay* (algorithm) that delays signals in samples. A *TimeDelay* (algorithm) model with delay time *T* simply increases the incoming time stamps by *T* without inserting any samples. The net effect is equivalent to delaying the incoming signal by *T* in time. Note that because *TimeDelay* (algorithm) is a timed model, non-evenly spaced incoming time stamps will be received as evenly spaced time stamps by Definition #1 stated in Timing Method for Timed Models. In addition, time delay *T* can be any non-negative number.

The following table illustrates what time stamps the downstream model will see when different domains of models are connected with a *TimeDelay* (algorithm) in between. (In this table, time step is 1, and notation "a@3" represents a data sample with value "a" at time stamp "3".)

Connection	Samples from output port	*TimeDelay* (algorithm)	Samples to input port
Timed to *TimeDelay* (algorithm) to Timed	Output from a timed model --> ... c@5 b@4 a@3	--> 3.6 time delay -->	... c@8.6 b@7.6 a@6.6 --> Input to a timed model
Timed to *TimeDelay* (algorithm) to Numeric	Output from a timed model --> ... c@5 b@4 a@3	--> 3.6 time delay -->	... c@8.6 b@7.6 a@6.6 --> Input to a numeric model
Numeric to *TimeDelay* (algorithm) to Timed	Output from a numeric model --> ... c@3 b@3 a@3	--> 3.6 time delay -->	... c@8.6 b@7.6 a@6.6 --> Input to a timed model
Numeric to *TimeDelay* (algorithm) to Numeric	Output from a numeric model --> ... c@3 b@3 a@3	--> 3.6 time delay -->	... c@8.6 b@7.6 a@6.6 --> Input to a numeric model

Time Synchronizer

SystemVue *TimeSynchronizer* (algorithm) is a multi-input/multi-output timed model that aligns multiple signals with different initial time stamps. It provides two modes for time alignment: "ZeroPadding" and "TimeDelay". ZeroPadding mode is equivalent to Definition #2 stated in Timing Method for Timed Models. In other words, zero-valued samples are inserted to later incoming signals such that all signals are aligned in time to the earliest incoming signal (earliest first time stamp). In contrast, TimeDelay mode is equivalent to forcing *TimeDelay* (algorithm)s for earlier signals such that they are delayed in time to align with the latest incoming signal (latest first time stamp).

The following table illustrates the operation of the two modes.

Samples to input ports	*TimeSynchronizer* (algorithm) Mode	Samples from output ports
... c@2 b@1 a@0 --> input port 0 ... f@4 e@3 d@2 --> input port 1	ZeroPadding	output port 0 --> ..... c@2 b@1 a@0 output port 1 --> ... f@4 e@3 d@2 0@1 0@0
... c@2 b@1 a@0 --> input port 0 ... f@4 e@3 d@2 --> input port 1	TimeDelay	output port 0 --> ... c@4 b@3 a@2 output port 1 --> ... f@4 e@3 d@2

Sources

Many of the SystemVue *Sources Category* (algorithm) allow users to choose different "SampleRateOption": "UnTimed", "Timed from SampleRate", and "Timed from Schematic". As described above, if the SampleRateOption is "UnTimed", the source is treated as a numeric (untimed) model and always generates 0 time stamps (see Definition #2 in Timing Method for Numeric (Untimed) Models). On the other hand, if the SampleRateOption is either "Timed from SampleRate" or "Timed from Schematic", the source acts as a timed model and generates evenly spaced, continuously increasing time stamps starting from 0 (see Definition #4 in Timing Method for Timed Models).

Sink and BER_FER

SystemVue *Sink* (algorithm) and *BER_FER* (algorithm) can collect either samples or timed signals. If "StartStopOption" is set to "Samples" (or "Auto" in untimed system, see Pure Numeric Simulation), the *Sink* (algorithm) (or *BER_FER* (algorithm)) is treated as a numeric (untimed) model and collect only samples in indices (index always start at 0). In addition, even if multiple input signals have different incoming times, time alignment will NOT be performed because it only collect samples in indices. If "StartStopOption" is set to "Time" (or "Auto" in timed system), the *Sink* (algorithm) (or *BER_FER* (algorithm)) is

treated as a timed model, and time alignment will be performed if necessary.

Non-evenly spaced time stamps are only used for internal timing computation. *Sink* (algorithm) cannot collect non-evenly spaced timed samples.

Pure Numeric Simulation

SystemVue sources are timed by default.

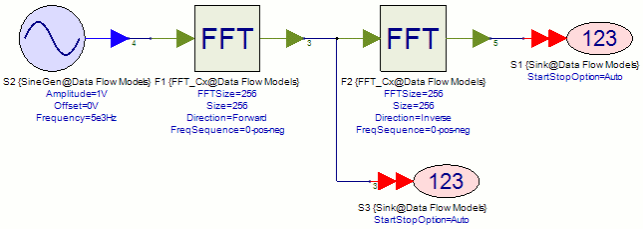
To create a pure numeric simulation, you should use only numeric models and set all the sources to be "UnTimed".

Examples

The following examples illustrate systems that are mixed of numeric (untimed) and timed models.

Example 1

Suppose the *SineGen* (algorithm) source "S2" has "SampleRateOption" parameter set to "Timed from SampleRate", and suppose the "SampleRate" is at 1M hz. In this case, the source "S2" behaves as a timed source and generates data samples with continuously increasing time stamps, 0 us, 1 us, 2 us, 3 us, ... The *FFT* (algorithm) model "F1" and the *IFFT* (algorithm) model "F2" are both numeric (untimed) models with consumption rate = 256 and production rate = 256 (this is because their "FFTSize" and "Size" parameters are set to 256). The first firing of "F1" consumes 256 samples with time stamps from 0 us to 255 us (see Definition #4 in Timing Method for Timed Models) and produces 256 samples all with the same time stamp 255 us (see Definition #1 in Timing Method for Numeric (Untimed) Models). Then the first firing of "F2" consumes 256 samples all with time stamp 255 us and produces 256 samples all with time stamp 255 us (see Definition #1 in Timing Method for Numeric (Untimed) Models). Finally, suppose the *Sink* (algorithm) "S3" has parameter "StartStopOption" to be "Auto" or "Time". In this case, "S3" behaves as a timed model and collect data samples starting from 255 us, followed by 256 us, 257 us, ... – this is because the first output sample from "F1" is at 255 us and the following samples are buffered and re-timed according to the sample rate at the input of "S3" (see Definition #1 in Timing Method for Timed Models). Similarly, suppose the *Sink* (algorithm) "S1" has parameter "StartStopOption" to be "Auto" or "Time". In this case, "S1" acts as a timed model and collect data samples starting from 255 us, followed by 256 us, 257 us, ... – this is because the first output sample from "F2" is at 255 us and the following samples are buffered and re-timed according to the sample rate at the input of "S1" (see Definition #1 in Timing Method for Timed Models).



S3_Time	re(S3)	im(S3)
255e-6	37.301	0
256e-6	96.505	-62.693
257e-6	-26.716	33.89
258e-6	-8.906	16.304
259e-6	-4.806	11.139
260e-6	-3.145	8.556

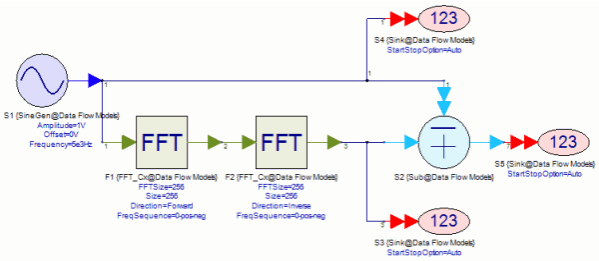
Screenshot of S3 in dataset.

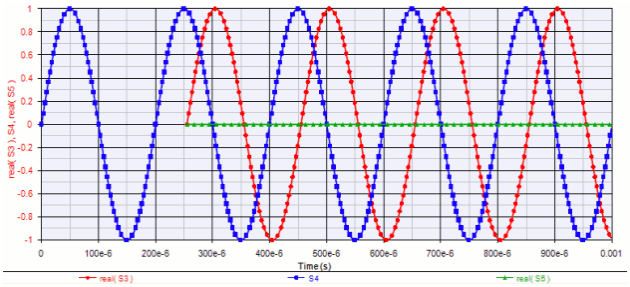
S1_Time	re(S1)	im(S1)
255e-6	-55.51e-18	-118e-18
256e-6	0.031	159.6e-18
257e-6	0.063	183e-18
258e-6	0.094	66.79e-18
259e-6	0.125	-79.8e-18
260e-6	0.156	54.64e-18

Screenshot of S1 in dataset.

Example 2

Example 2 continues from Example 1. Here, *SineGen* (algorithm) and *Sink* (algorithm)s are timed models, and *FFT* (algorithm), *IFFT* (algorithm), and *Subtractor* (algorithm) are numeric (untimed) models. We plot the signals collected by S3, S4, and S5 in the same graph. As described in Example 1, S3 (red signal) starts at time 255 us due to multirate behavior of the FFT and IFFT. S4 (blue signal) is directly connected from source *SineGen* (algorithm), so it starts at time 0 us. The *Subtractor* (algorithm) is a pure numeric (untimed) model, so it performs subtraction sample by sample (without time alignment) and outputs first sample starting at time 255 us.




real(S3), S4, real(S5)
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
100e-6
200e-6
300e-6
400e-6
500e-6
600e-6
700e-6
800e-6
900e-6
0.001
Time (s)
real(S3)
S4
real(S5)

Introduction to Dynamic Data Flow Simulation

Overview

As introduced in *Introduction to Data Flow Simulation* (sim), most SystemVue blocks are modeled based on *Synchronous Data Flow* (sim) (SDF) semantics. In other words, the data flow production and consumption rates for these blocks are restricted to be constant integers and must be known at compile-time (before simulation). With such restriction, SystemVue is able to perform compile-time optimizations such as deadlock detection, sample-rate (timing) resolution, buffer allocation, and static scheduling that jointly minimize runtime overheads.

However, such constant integer restriction also limits the expressive power of SDF. In fact, real world systems may involve dynamic behavior that cannot be modeled under SDF semantics. For example, modern wireless communication systems often apply adaptive modulation to adjust modulation schemes (e.g., QPSK, 16 QAM, 64 QAM [1]) dynamically based on channel condition. The *Mapper* (algorithm) block that maps a fixed number of bits to a symbol based on the pre-specified modulation scheme is definitely not enough to model such system.

Beyond SDF, dynamic data flow (DDF) is the most general data flow model of computation. In dynamic data flow, the number of samples consumed and produced for each execution of a DDF block can change dynamically at runtime. Such flexibility gives DDF sufficient expressive power to model various dynamic behavior, but at the expense of losing compile-time scheduling capabilities. In general, dynamic data flow requires significant amount of runtime overheads to determine execution order, detect deadlock, and allocate/re-allocate buffers.

SystemVue's Dynamic Data Flow Approach

Modern wireless communication systems often involve dynamic behavior, e.g., adaptive modulation [1,3], HARQ (hybrid automatic repeat request) [2, 3], and dynamic resource allocation. In such systems, even though the number of data to be processed and transmitted can change dynamically at runtime, standards are defined in such a way that within a transmission unit (e.g., subframe in LTE [1]), the numbers of data produced and consumed between adjacent blocks are properly matched.

To efficiently model such dynamic but matched rate changes, SystemVue uses **variable-size vectors (matrices)** to encapsulate variable numbers of samples for dynamic data flow processing. In this approach, blocks process a vector (matrix) of data at a time, and vector (matrix) size can change dynamically at runtime. Two special blocks (*DynamicPack_M* (algorithm) and *DynamicUnpack_M* (algorithm)) are provided that use dynamic data flow technology to enable the interactions between variable-size, vector (matrix)-based blocks and static, sample-based blocks.

The following sections describe the approach in more details.

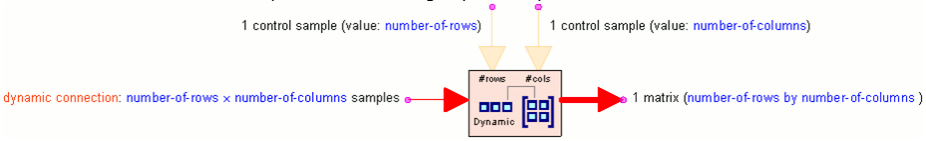
Dynamic Data Flow Models

SystemVue provides two dynamic data flow models, *DynamicPack_M* (algorithm) and *DynamicUnpack_M* (algorithm), to dynamically pack and unpack variable numbers of samples to and from matrices.

Vector (row vector or column vector) in SystemVue is represented as one-dimensional matrix.

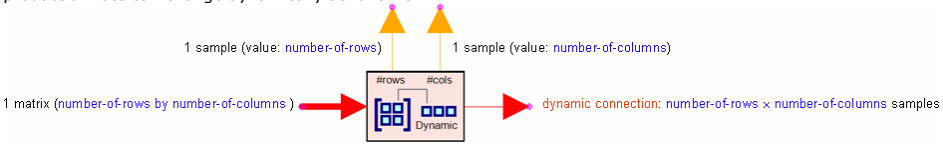
DynamicPack_M

DynamicPack_M (algorithm) packs variable numbers of samples into matrices based on control signals that control the *number-of-rows* and *number-of-columns* of the matrix to be packed for each execution (see to *DynamicPack_M* (algorithm) for details). The following picture shows the data flow rates of *DynamicPack_M* (algorithm) for a complete execution. The connection to the *input* port of *DynamicPack_M* (algorithm) is a **dynamic connection** because the consumption rate can change dynamically at runtime.



DynamicUnpack_M

DynamicUnpack_M (algorithm) unpacks variable size matrices into samples (see *DynamicUnpack_M* (algorithm) for details). The following picture summaries the data flow rates of *DynamicUnpack_M* (algorithm) for a complete execution. The connection from the *output* port of *DynamicUnpack_M* (algorithm) is a **dynamic connection** because the production rate can change dynamically at runtime.



Whenever there is a dynamic block (currently *DynamicPack_M* (algorithm) and *DynamicUnpack_M* (algorithm)) on schematic, SystemVue will turn on dynamic data flow simulation.

Variable-Size Matrix Processing

Modeling signal processing blocks in dynamic data flow in order to process variable numbers of samples and configure rates dynamically at runtime, in general, causes significant amount of scheduling overheads and results in poor usability when developing and using such blocks.

For wireless communication types of applications, it is very efficient to represent variable numbers of samples into variable-size vectors (matrices) and model such block to process a matrix at a time. This modeling approach fits well with how DSP algorithm designers develop the blocks. In addition, because most of the processing blocks remain in SDF domain (fixed data flow rates at the granularity of matrices), SystemVue is able to apply various optimization and scheduling techniques that gives much better runtime performance.

In variable-size matrix-based processing, a block's operation can adjust dynamically to the input matrix size. If the input matrix size conflicts with block's settings (e.g., parameters, other inputs, or control signals), the block may error it out, ignore the unused elements, or perform whatever appropriate based on the defined operation. When inconsistency occurs, SystemVue does not handle individual samples inside matrices.

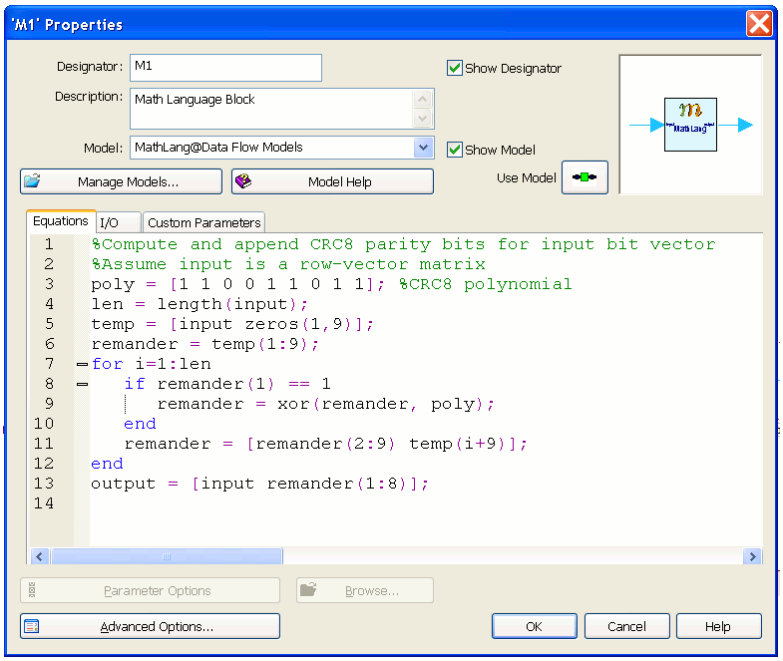
How to Use SystemVue's Dynamic Data Flow

For processing variable numbers of samples, including 0 number of sample that holds incoming stream, users can use *DynamicPack_M* (algorithm) coupled with controller(s) that control the numbers of samples to be packed. The resulting variable-size matrices can then be processed by *matrix* (sim)-type blocks, see *Math Matrix Category* (algorithm) and *3GPP LTE Baseband Verification Library* (ltebasever), or user defined models with matrix I/O, see *C++ Models* (users) and *MathLang* (algorithm). To convert variable-size matrices back to samples, users can use *DynamicUnpack_M* (algorithm) for further sampled-based processing. SystemVue *Sink* (algorithm) collects variable-size matrices into *DataSet* (users) as **cell array**, and users can choose each individual matrix for display.

In SystemVue libraries, only the matrix-type blocks in *Math Matrix Category* (algorithm) and *3GPP LTE Baseband Verification Library* (ltebasever) that handle variable-size matrices can work with SystemVue's dynamic data flow approach.

User-Defined Models for Variable-Size Matrix Processing

Users can create custom *MathLang* (algorithm) block for variable-size matrix processing in dynamic data flow. When *port rates* (algorithm) are one, input and output matrices can be simply accessed in the *Equations* (algorithm) tab by using the port's "*Name in Equations* (algorithm)". The following screenshot shows an example using *MathLang* (algorithm) to create a CRC encoder block that computes CRC8 parity bits for input bit vector and append the parity bits in output vector, regardless of vector size (number of bits).



Users can also create custom *C++ Model* (users) for variable-size matrix processing. To access input and output matrices in C++ models, users have to use matrix-type *circular buffer* (users) (see \SystemVue2009.08\ModelBuilder\include\SystemVue\MatrixCircularBuffer.h) and use SystemVue *matrix* (users) data type (see \SystemVue2009.08\ModelBuilder\include\SystemVue\Matrix.h). The following code shows the C++ model header file for *Mapper_M* (algorithm) that maps an input bit vector (boolean matrix) into a complex constellation point. The modulation type is dynamically determined by the number of bits in the input matrix (see *Mapper_M* (algorithm)).

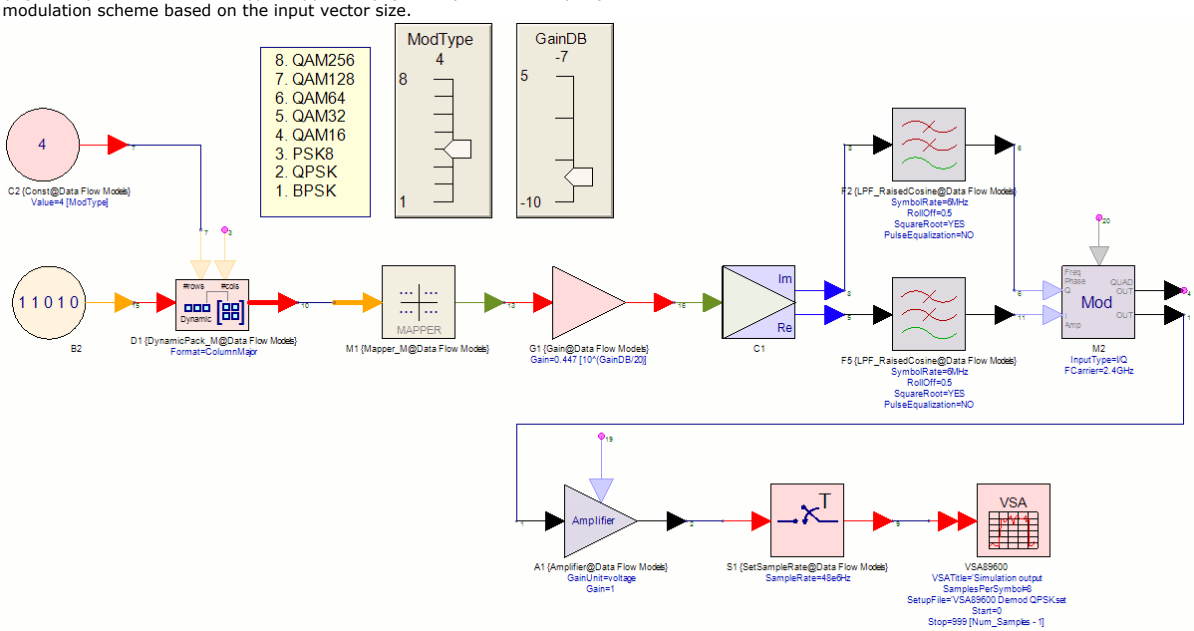
```
class Mapper_M : public AgilentEEsof::DFModel
{
public:
   DECLARE_MODEL_INTERFACE ( Mapper_M )
   bool Run();
   bool Initialize();
   //input bit vector
   AgilentEEsof::CircularBufferE<AgilentEEsof::Matrix<bool>> m_input;

   //output complex symbol
   AgilentEEsof::CircularBuffer<std::complex<double>> m_output;
   //...
};
```

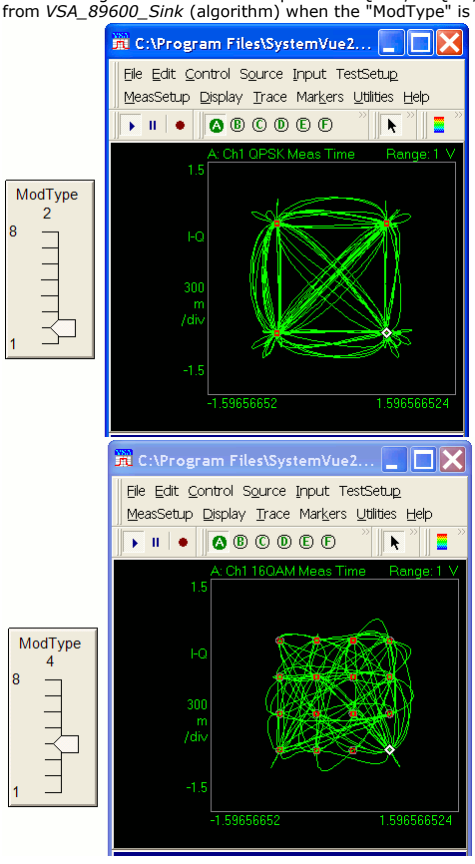
Example: Dynamic Mapper

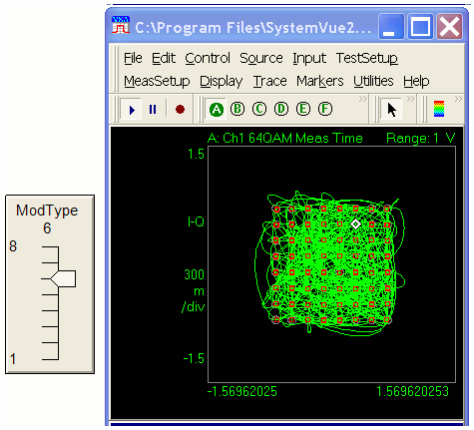
The following screenshot shows the dynamic mapper example in "QPSKTuning" schematic

in \SystemVue2009.08\Examples\Instruments\VSA89600Sink\VSA89600 Demod QPSK.wsv. The "ModType" slide bar allows users to control the number of bits to be packed by the *DynamicPack_M* (algorithm) "D1" through changing the value of *Const* (algorithm) "C2". The matrix-type *Mapper_M* (algorithm) automatically adjusts the modulation scheme based on the input vector size.



The following three screenshots capture the QPSK, 16QAM, and 64QAM constellation plots from *VSA_89600_Sink* (algorithm) when the "ModType" is 2, 4, and 6 respectively.

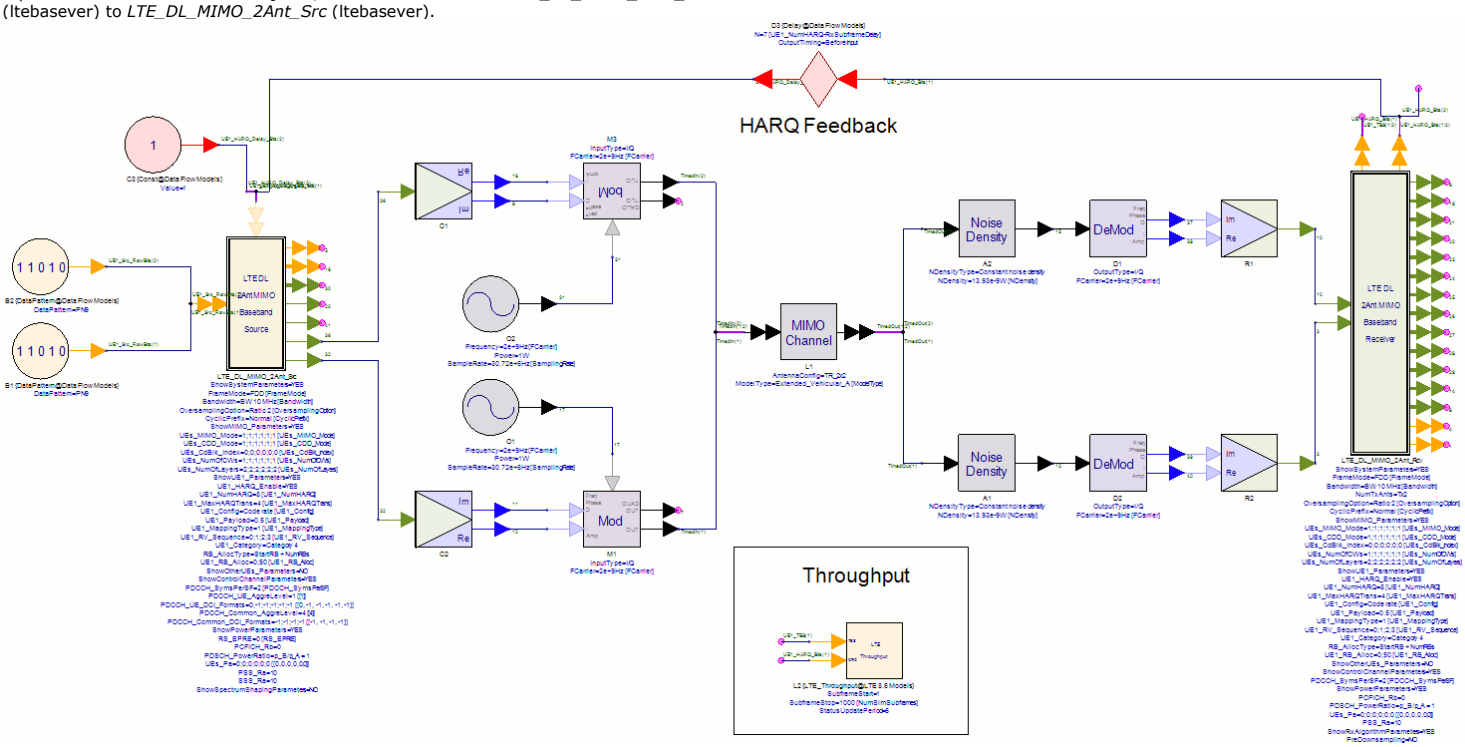




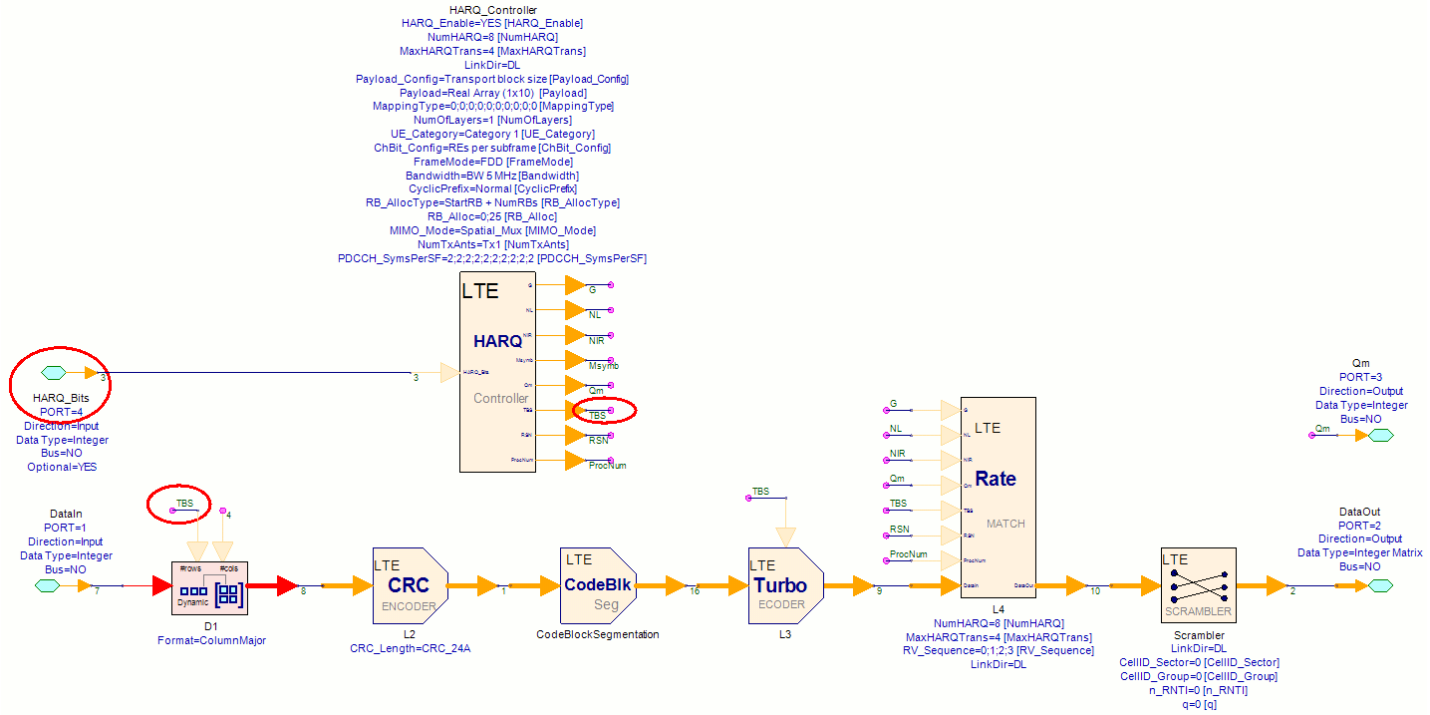
Example: LTE HARQ

This example illustrates how SystemVue LTE workspaces use *DynamicPack_M* (algorithm), matrix-based processing, and dynamic data flow to model various dynamic behavior in LTE systems. It assumes that the readers are familiar with LTE standards [1,2,3].

The following screenshot captures the top-level schematic for LTE downlink 2x2 MIMO fading channel throughput measurement. This example can be found in \SystemVue2009.08\Examples\Baseband Verification\LTE\3GPP_LTE_DL_MIMO_Throughput.wsv. The upper path feedback loop represents the feedback for HARQ ACK/NACK information from *LTE_DL_MIMO_2Ant_Rcv* (ltebasever) to *LTE_DL_MIMO_2Ant_Src* (ltebasever).



The following picture shows the *LTE_DL_ChannelCoder* (ltebasever) sub-network schematic. The HARQ feedback path in the top-level schematic actually goes into *LTE_HARQ_Controller* (ltebasever) inside *LTE_DL_ChannelCoder* (ltebasever) inside *LTE_DL_MIMO_2Ant_Src* (ltebasever). If the HARQ signal is ACK for a particular HARQ process, *LTE_HARQ_Controller* (ltebasever) then computes the transport block size (TBS) (i.e., the number of bits to be transmitted from upper layer) for the particular subframe based on various LTE settings. Note that the *TBS* value varies subframe by subframe. Based on the *TBS* signal, the *DynamicPack_M* (algorithm) packs *TBS* number of bits into vector (transport block [2]). The transport block is then processed by *LTE_CRCEncoder* (ltebasever), *LTE_CodeBlkSeg* (ltebasever), *LTE_TurboCoder* (ltebasever), and *LTE_RateMatch* (ltebasever) blocks. If the HARQ signal is NACK for a particular HARQ process, *LTE_HARQ_Controller* (ltebasever) then sets *TBS* to 0 to force *DynamicPack_M* (algorithm) to hold the incoming bits and output "empty" vector (matrix) that stops the processings for *LTE_CRCEncoder* (ltebasever), *LTE_CodeBlkSeg* (ltebasever), and *LTE_TurboCoder* (ltebasever). The retransmission starts from the rate matching buffer inside the *LTE_RateMatch* (ltebasever) block. The following block diagram and the operations match exactly as defined in 3GPP TS 36.212 [2].



The variable-size vector (matrix)-based processing inside *LTE_DL_MIMO_2Ant_Src* (ltebasever) sub-network ends at *LTE_DL_MuxOFDMSym* (ltebasever). After *LTE_DL_MuxOFDMSym* (ltebasever), signal is represented sample by sample again. This is because *LTE_DL_MuxOFDMSym* (ltebasever) maps all physical channels and physical signals into resource elements in one subframe, and the number of resource elements in a subframe (number of OFDM symbols * number of sub-carriers) is fixed based on the LTE configuration. Back to the top-level LTE downlink MIMO fading channel schematic, *LTE_DL_MIMO_2Ant_Src* (ltebasever) generates 1 ms subframe in samples, which then up converted to carrier frequency and go through fading and noisy channel using SystemVue *Analog RF Category* (algorithm) blocks.

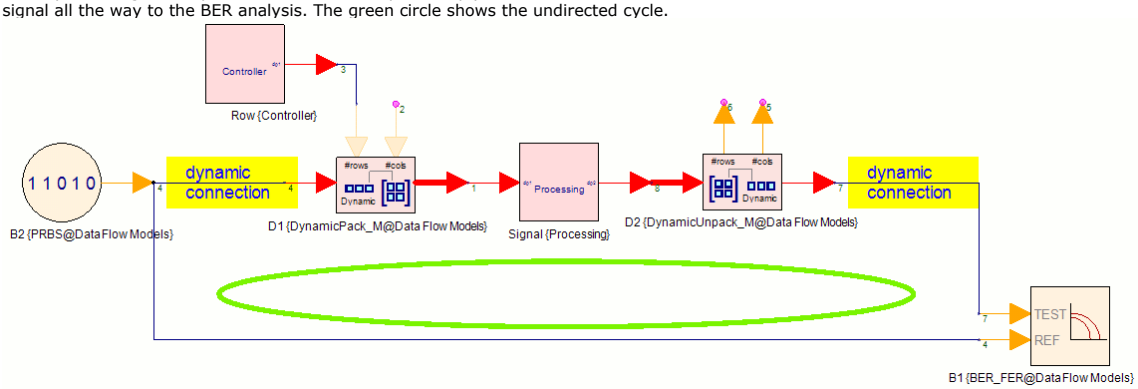
Limitations

1. SystemVue currently supports only a single connected graph in dynamic data flow simulation (i.e., there must an undirected path between any pair of blocks). In other words, when there is a dynamic block on schematic that triggers dynamic data flow simulation, SystemVue will error it out if there are multiple isolated graphs.

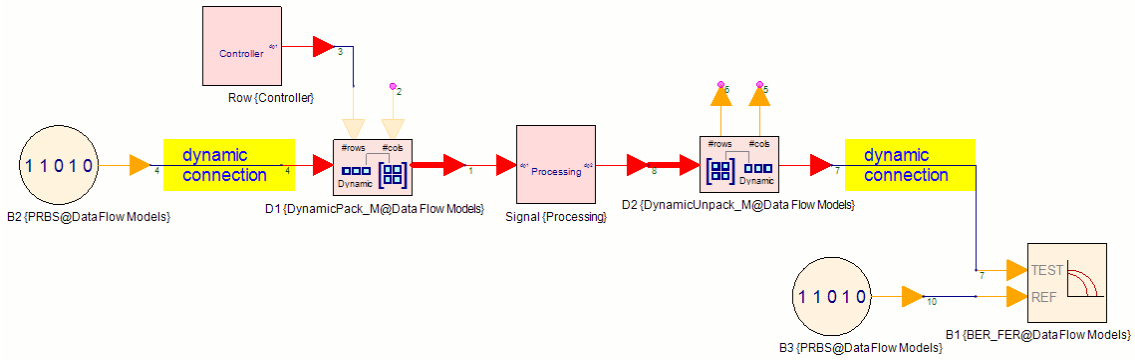
   Users can always move an isolated graph to another schematic for simulation.

   The only exceptions are:

   - The range-check blocks (e.g., *LTE_DL_Src_RangeCheck* (ltebasever), *LTE_UL_Src_RangeCheck* (ltebasever)) in *3GPP LTE Baseband Verification Library* (ltebasever) and *ControllerFxp* (hardware) in *Hardware Design Library* (hardware). These blocks have no I/O, thus, do not involve in data flow.
   - Isolated graphs without sinks that demand the simulation (see *simulation control* (sim)).
2. For dynamic data flow simulation, SystemVue currently requires that any pair of **dynamic connections** must **not lie in any undirected cycle**. In other words, removing any dynamic connection must disconnect the graph. As described in Dynamic Data Flow Blocks, a dynamic connection only refers to a connection to *DynamicPack_M* (algorithm)'s *input* port or a connection from *DynamicUnpack_M* (algorithm)'s *output* port. The most common scenario to get into this restriction is illustrated in the following schematic. Here, the upper path represents a common use model that dynamically packs variable numbers of samples for variable-size, matrix-based processing and unpacks them. The lower path simply connects the source signal all the way to the BER analysis. The green circle shows the undirected cycle.



   The proper way to construct this type of design is to provide a repeatable signal source (e.g., pseudo random bit source) feed separately into the BER, as shown in the following schematic.

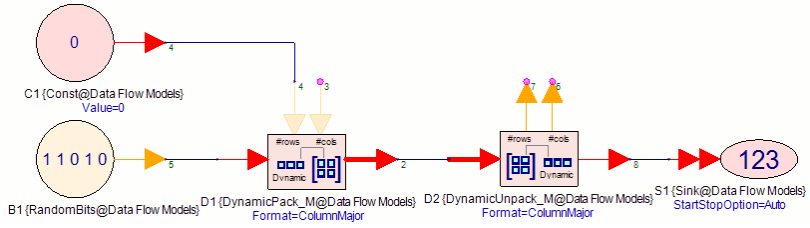


3. Dynamic connections partition the graph into separated **static regions** — i.e., by removing dynamic connections, the graph is disconnected into isolated regions, and each region is in fact an SDF/TSDF graph with only static data flow rates. SystemVue currently **times each static region separately** because the constant-sample-rate requirement in *Timing Method* (sim) cannot be preserved across dynamic connections.

> Comparing timed signal across different static regions is not recommended.

> In dynamic data flow simulation, the *Data Flow Information Table* (sim) will separate each static region with a - - - line to help users to identify different static regions.

4. It is user's responsibility to make sure the system does not deadlock or run forever "by design". The following screenshot shows an example where *DynamicPack_M* (algorithm) always packs empty matrices (because the control signal is always 0), and *DynamicUnpack_M* (algorithm) has nothing to output (because empty matrices). This example will run forever until it is forced to stop because the *Sink* (algorithm) cannot collect enough samples to stop the simulation (see *Simulation Control* (sim)).



References

1. 3GPP TS 36.211 v8.6.0, "Physical Channels and Modulation", March 2009.
2. 3GPP TS 36.212 v8.6.0, "Multiplexing and Channel Coding", March 2009.
3. 3GPP TS 36.213 v8.6.0, "Physical layer procedures", March 2009.

Using Data Types

As was discussed in the *Nets, Connection Lines and Buses* (users) page, the data flow port colors and thickness represent the data type that is passed between the connected parts. In addition to port colors and thickness, part and model names are suffixed to indicate the data type supported. The following table lists that how the different data types are represented.

Data Type	Port Color	Port Thickness	Model Suffix
Scalar Integer	**Orange**	Thin	Int
Scalar Floating Point (Real)	**Blue**	Thin	
Scalar Complex	**Green**	Thin	Cx
Scalar Fixed Point	**Magenta**	Thin	Fxp
Integer Matrix	**Orange**	**Thick**	Int_M
Floating Point (Real) Matrix	**Blue**	**Thick**	_M
Complex Matrix	**Green**	**Thick**	Cx_M
Envelope Signal	**Black**	Thin	Env
Any Type	**Red**	Thin	
Variant	**Cyan**	Thin	

Note: In the current release of SystemVue, there are no parts that process integer matrices.

To learn more about models and parts refer to *Parts, Models, and Symbols* (users) page.

Data Types Defined

Scalar Data

Scalar data is defined as follows:

- integer value (signed value defined with a 32-bit value)
- fixed point value
- floating point (real) double precision floating-point (real) number
- complex pair of double precision floating-point (real) number for real and imaginary parts

Fixed Point Data

SystemVue FixedPoint Data Type (users) is similar in computational behavior of SystemC™ 2.2 fixed point type based on IEEE Std. 1666™ Language Reference Manual (LRM) .

For detailed description of SystemC™ 2.2 fixed-point type, consult section 7.10 of SystemC ™ IEEE Std. 1666™ Language Reference Manual (LRM)

A fixed-point type is characterized by whether it is signed or unsigned, its word length (wl), integer word length (iwl), quantization mode, overflow mode and the number of saturation bits (only applicable in certain overflow modes).

The range of values for a signed fixed-point number is $[2^{iwl-1}, 2^{iwl-1} - 2^{-(wl-iwl)}]$.

The range of values for an unsigned fixed-point number is $[0, 2^{iwl} - 2^{-(wl-iwl)}]$.

The available quantization modes are:

- RND - rounding to plus infinity
- RND_ZERO - rounding to zero
- RND_MIN_INF - rounding to minus infinity
- RND_INF - rounding to infinity
- RND_CONV - convergent rounding
- TRN - truncation (default)
- TRN_ZERO - truncation to zero

The available overflow modes are:

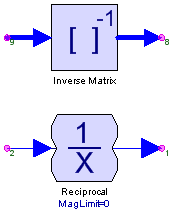
- SAT - saturation
- SAT_ZERO - saturation to zero
- SAT_SYM - symmetrical saturation
- WRAP - wrap-around (default)
- WRAP_SM - sign magnitude wrap-around (undefined for unsigned types)

Saturation bits are only applicable in the two wrap-around overflow modes.

For more information please see *SystemVue FixedPoint Data Type* (users).

Matrix Data

All matrix data is defined as a two-dimensional array (rows, columns) of either int, fixed, floating-point (real), or complex values. All matrix data types are indicated by thick lines, in contrast with the thin lines used for scalar data types.



Matrix Data (Thick Lines) and Scalar Data (Thin Lines)

Envelope Signal

SystemVue uses an efficient envelope signal to represent modulated signals on RF carriers. In this representation, the sample rate needed to represent the complex envelope signal is on the order of the information (envelope) bandwidth. This is in contrast to using a to direct real signal representation where the sample rate needs to be on the order of the RF carrier. To learn more about this format, refer to the *Envelope Signal* (sim) section.

Data Type Polymorphism

Data type polymorphism enables you to use a part with data types.

Variant

Variant is a container data type. It holds data together with the tag identifying the data type. The data can be of any data flow type except fixed-point and envelope signals.

Variant accepts user entry as is, assigning it the tag that best matches the data.

For instance, the *MathLang* (algorithm) model enables processing of data that can assigned an integer, real or complex number, or an integer, real or complex matrix and it will be stored in the variant directly.

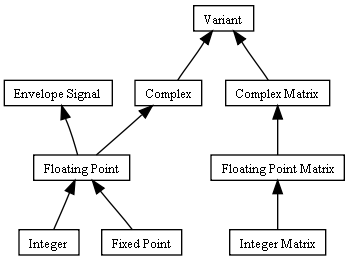
Conversions between variant and other data types follow the same rules as for the data types without the container (see below).

Any type

An any type port has a type that is resolved during the initialization stage of the simulation and remains the same through the simulation run.

For instance, the *Const* (algorithm) source has an any type output. The output data type is determined by the type of the Value parameter setting.

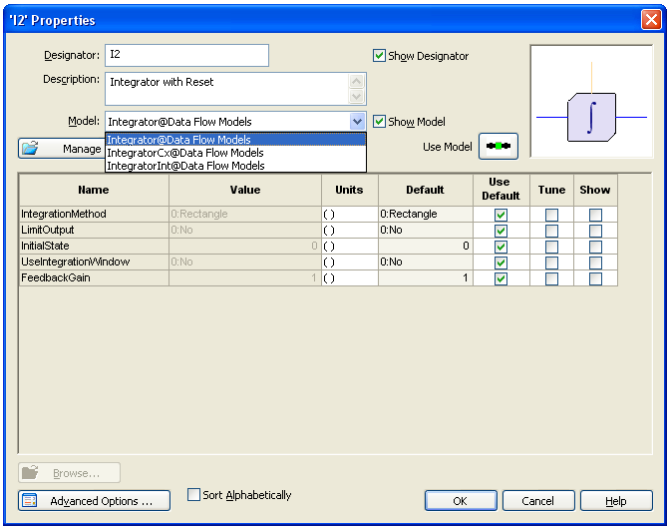
In other places, such as in *Add* (algorithm) and *Mpy* (algorithm), the type is resolved based on the input data types. For example, if a *Add* (algorithm) has both integer and a floating point input data, the output data type will be resolved to floating point. The figure below summarizes the type resolution in cases such as these.



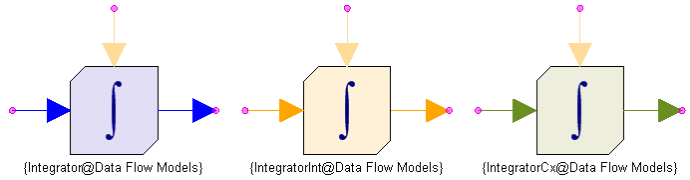
When the data type is resolved to either envelope signal or fixed point data types, the any type model will use the default parameter settings of the associated strongly typed model. For example, for the *Mpy* (algorithm) part, a *MpyFxp* (hardware) would be used if the data type is resolved to fixed point and a *MpyEnv* (algorithm) if the data type is resolved to envelope signal. If you want to use non-default parameters in these cases, you should use the method described below in the Polymorphic Parts section below.

Polymorphic Parts

A part may have multiple models associated to enable you to switch the data type processed by the block. An example of this is the *Integrator* available in the *Algorithm Design* library. To switch the data type, double click on the part to edit the part properties. Then click on the *Model* menu as shown below:



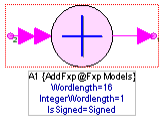
As you switch between the models, the port colors change. Here are the different models available for the *Integrator* part:

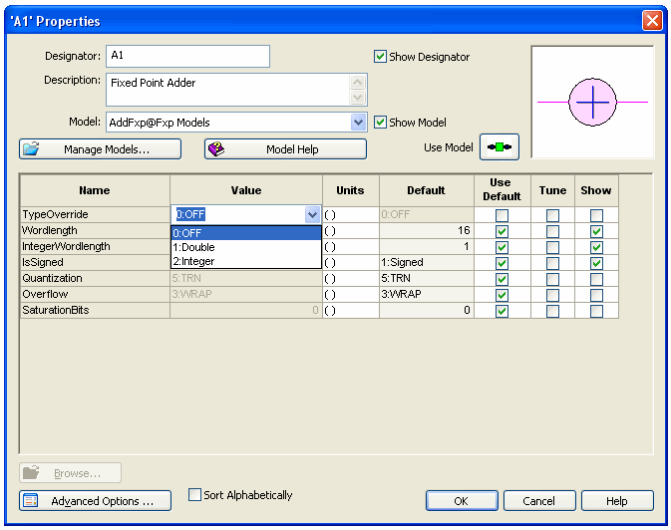


Polymorphic Models

A model may also natively support multiple types. All of the parts in the *Hardware Design* library support a data type override. By default, these models support fixed-point (TypeOverride=OFF), in addition they support floating-point (TypeOverride=Double) and integer (TypeOverride=Integer).

Below is an instance of the AddFxp part showing the TypeOverride parameter:





Conversion of Data Types

Most type conversions do what you expect. If the conversion from A to B requires more information (integer to floating-point (real), floating-point (real) to complex, etc.) the obvious happens. For example, conversion from floating-point (real) to complex is done by setting the imaginary part of the complex number equal to 0.0. If there is loss of information automatic conversion is not supported, the only exception to this is from floating or fixed point to integer. For matrix type conversions, the conversions are done element-by-element following the same rules.

Below is a summary of the conversion rules:

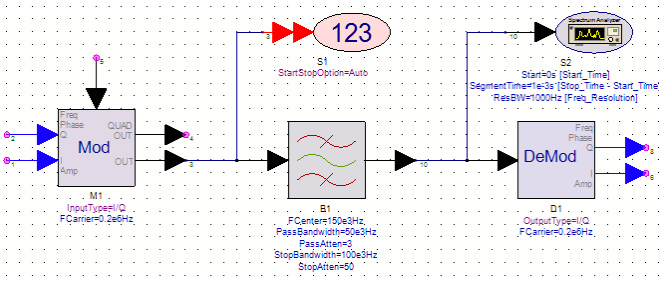
From	To	Rule
Scalar	Matrix	A 1x1 matrix of the scalar is created.
Floating or Fixed Point Number	Integer	The value is rounded to the closest integral value.
Non-complex	Complex	The real part is set to the value, the imaginary part is set to 0.
Envelope signal	Floating point or variant	Real base-band equivalent of the signal is computed.
Variant	All data types except envelope signal	Converted using rules above.

Envelope Signal

How To Use Envelope Signal

Most SystemVue RF blocks operate on an envelope signal. Please refer to *Analog/RF* (algorithm) for the list of RF blocks. In SystemVue, a modulated passband signal is usually represented as an **analytic signal**, $x_a(t) = (x_i(t) + j\, x_q(t)) \exp(j\, 2\, \pi\, f_c\, t)$, in the **envelope data type**. Envelope data carries the characterization frequency f_c of the modulated passband signal, and RF signal processing is performed at the complex envelope level. You can view it as time varying **complex envelope** data, $x_c(t) = x_i(t) + j\, x_q(t)$, flow through envelope (black pin to black pin) connection.

A user can use a *Modulator* (algorithm) to modulate a baseband signal into a passband signal in analytic (envelope) format. The modulated signal can then be processed by the SystemVue *Analog/RF* (algorithm) parts. A user can demodulate the RF (passband) signal by using a *Demodulator* (algorithm). At any point, a user can use a *Sink* (algorithm) to collect envelope data or use a *SpectrumAnalyzerEnv* (algorithm) to view the spectrum. The following image illustrates a simple RF design.



The benefit of using SystemVue envelope signal to represent modulated signals compared to direct real signal representation is that the sample rate needed to fully represent a complex envelope signal can be in the order of the information bandwidth, which is in general orders of magnitude smaller than the sample rate required for direct real signal representation.

Please note that the characterization frequency is not the same as the signal carrier frequency.

Even though the characterization frequency of an envelope-type port remains constant (as discussed above), a frequency modulated signal can still be represented because the in-phase $x_i(t)$ and quadrature $x_q(t)$ components as well as the signal "carrier" frequency can change over time.

Definition

Let x(t) represents a real-valued signal. An **analytic signal** $x_a(t)$ is defined as $x_a(t) = x(t) + j\, x_h(t)$, where $x_h(t)$ is the Hilbert transform of x(t).

An analytic signal can be expressed in the form $x_a(t) = x_c(t) \exp(j\, 2\, \pi\, f_c\, t)$. In this representation, $x_c(t)$ is defined as the **complex envelope** of x(t), and f_c is the **characterization frequency** associated with the complex envelope. Complex envelope $x_c(t)$ is a complex-valued signal. It can be expressed as $x_c(t) = x_i(t) + j\, x_q(t)$, where $x_i(t)$ and $x_q(t)$ are both real-valued signals and are referred to as the **in-phase component** the **quadrature component** of x(t).

Using complex envelope form, the real signal can be expressed as $x(t) = \text{Real}\{x_a(t)\} = x_i(t) \cos(2\, \pi\, f_c\, t) - x_q(t) \sin(2\, \pi\, f_c\, t)$.

Based on the above equations, an analytic signal $x_a(t)$ (as well as the associated real signal $x(t) = \text{Real}\{x_a(t)\}$) can be characterized equivalently by various different characterization frequencies f_c. Different characterization frequencies result in different complex envelopes $x_c(t) = x_i(t) + j\, x_q(t)$. Regardless of different characterization frequencies, the analytic signal $x_a(t)$ (as well as the associated real signal $x(t) = \text{Real}\{x_a(t)\}$) remains the same. For example, both $x_a(t)$ and x(t) below are represented with two sets of complex envelopes and characterization frequencies:

$x_a(t) = (x_{i1}(t) + j\, x_{q1}(t),) \exp(j\, 2\, \pi\, f_{c1}\, t) = (x_{i2}(t) + j\, x_{q2}(t),) \exp(j\, 2\, \pi\, f_{c2}\, t)$

$x(t) = x_{i1}(t) \cos(2\, \pi\, f_{c1}\, t) - x_{q1}(t) \sin(2\, \pi\, f_{c1}\, t) = x_{i2}(t) \cos(2\, \pi\, f_{c2}\, t) - x_{q2}(t) \sin(2\, \pi\, f_{c2}\, t)$.

SystemVue Envelope Data Type

In SystemVue, an **envelope signal** v(t) 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 $f_c > 0$.

As discussed in *Using Data Types* (sim), SystemVue associates **envelope** data type with the color **black**. A black-colored port operates on envelope signal. SystemVue data flow technology restricts such 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.

Delay for Envelope Signal

SystemVue *Delay* (algorithm) block represents z^{-N} operator, where N is the size of delay.

When a Delay block is placed before an envelope-typed (black-colored) port, SystemVue data flow technology is able to set the initial delay value for such envelope signal based on its representation and characterization frequency. If the envelope data sample represents a real signal, the initial delay value is set to real 0. On the other hand, if the envelope data sample represents an analytic signal, the initial delay value is then set to 0-valued complex envelope (x_i=0 and x_q=0) at the associated characterization frequency.

The alternate envelope signal delay block *DelayEnv* (algorithm) is also available for delaying an envelope signal with additional features such as interpolation in the delay process, characterization frequency phase shift, and it also supports voltage controlled delay as well. Please refer to *DelayEnv* (algorithm) for details.

Collecting Envelope Signal into Dataset

SystemVue *Sink* (algorithm) block can collects envelope data samples into a dataset. Suppose "S1" is the name of the Sink, and let *i* denote the index of samples in the order of receiving.

When the input envelope data represent a real signal, the Sink block collects real signal values into the variable "S1" in dataset, along with time stamps into the variable "S1_Time".

When the input envelope data represents an analytic signal, the Sink block collects complex envelope values ($x_i(i)$, $x_q(i)$) into the complex variable "S1", and also collects real characterization frequency values $f_c(i)$ into the variable "S1_Fc", along with time stamps into the variable "S1_Time".

The sampled version of the analytic signal can be expressed as x_a(S1_Time(*i*)) = S1(*i*) exp(j 2π S1_Fc(*i*) S1_Time(*i*)), and the sampled version of the real signal can be expressed as x(S1_Time(*i*)) = real(S1(*i*)) cos(2π S1_Fc(*i*) S1_Time(*i*)) - imag(S1(*i*)) sin(2π S1_Fc(*i*) S1_Time(*i*)).

Here is a complex envelope sample in a dataset. To see the complex format, right click on "S1", choose "Real+Imaginary" in Complex Format.

DF1_Data

Variable	(s:)	S1_Time	re(S1)	im(S1)
LogOutput="Execution time: 0.494 sec...	1	0	0	0
S1	2	1e-6	0.588	0.618
S1_Fc	3	2e-6	0.951	1.176
S1_Time	4	3e-6	0.951	1.618
	5	4e-6	0.588	1.902
	6	5e-6	122.5e-18	2
	7	6e-6	-0.588	1.902
	8	7e-6	-0.951	1.618
	9	8e-6	-0.951	1.176
	10	9e-6	-0.588	0.618

Type Conversion

This section introduces type conversion to and from **envelope** type.

Floating Point, Integer, and Fixed Point to Envelope

This category of type conversion is performed by SystemVue data flow technology. The resulting envelope will represent a real signal. The value of the real signal (in double floating point precision) is converted directly from the value of floating point, integer, or fixed point.

Complex to Envelope

SystemVue does NOT support automatic type conversion from complex to envelope, and will report error message for such conversion.

Users need to explicitly place a *CxToEnv* (algorithm) block in-between such connections to specify the characterization frequency of the complex data in order to convert it to an analytic signal. Please refer to *CxToEnv* (algorithm) for details.

Let cx(t) = re(t) + j im(t) represents the input complex signal to CxToEnv. Let f_c denote the characterization frequency of the input "fc" pin of CxToEnv, or the value of the "Fc" parameter when "fc" pin connection does not exist. When f_c is 0, the output envelope represents a real signal re(t). When $f_c > 0$, the output envelope represents an analytic signal (re(t) + j im(t)) exp(j 2 π f_c t).

Variant to Envelope

SystemVue data flow technology will automatically insert a converter block in between variant to envelope connections.

If a variant is a scalar type, the resulting envelope will represent a real signal. The value of the real signal is converted directly from the value of the variant.

If a variant is a complex type, SystemVue will ask users to place a *CxToEnv* (algorithm) block for such connection.

If variant is another type other than scalar or complex, SystemVue will not support the

conversion and will reports error messages.

Envelope to Floating Point, Integer, and Variant

SystemVue Data Flow technology will automatically insert a converter block in between these connections.

If an envelope represents a real signal, the converter will converts the real signal value (in double floating point precision) directly to destination data type (floating point, integer, or variant). Possible loss of information may occur due to numeric precision.

If an envelope represents an analytic signal, the converter will first convert the analytic signal to a real signal, $x(t) = x_i(t) \cos(2 \pi f_c t) - x_q(t) \sin(2 \pi f_c t)$, then convert the real signal value to the destination data type.

In analytic to real signal conversion, SystemVue will report warning messages if the sample rate is not enough ($< 4*f_c$) to characterize the resulting real signal.

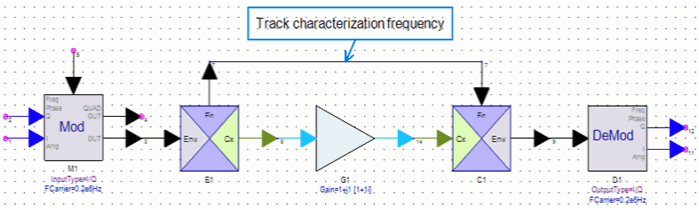
Envelope To Complex

SystemVue does NOT support automatic type conversion from envelope to complex, and will ask users to explicitly place an *EnvToCx* (algorithm) block to track the characterization frequency. Please refer to *EnvToCx* (algorithm) for details.

If an envelope represents a real signal, the output of *EnvToCx* (algorithm) is simply a complex signal with the real part equal to the input real signal and imaginary part equal to 0. The **fc** output pin will carry the characterization frequency information ($f_c = 0$).

If an envelope represents an analytic signal, the output of *EnvToCx* (algorithm) is simply the complex envelope signal, $x_c(t) = x_i(t) + j\, x_q(t)$. The **fc** output pin will carry the characterization frequency information.

The following image illustrates a typical usage of *EnvToCx* (algorithm) and *CxToEnv* (algorithm). The "fc" output of EnvToCx can be used to carry the characterization frequency of the envelope signal for later conversions, and the "fc" input of CxToEnv can be used to obtain the characterization frequency from another envelope signal. In the case, the connection from the "fc" output of EnvToCx "E1" to the "fc" input of CxToEnv "C1" tracks the characterization frequency of the envelope signal, and users can apply computations purely on the complex envelope signal in between "E1" and "C1".



Other Types

Except the data types described above, SystemVue does not support other type conversions to and from envelope.

Filtering Envelope Signal

Many SystemVue filter blocks operate on envelope signal. This section introduces the filtering process for envelope signal. Please refer to *IIR Filter Design* (users) and *FIR Filter Design* (users) for basic filter design methods and refer to *Filter Part* (algorithm) for SystemVue filter models that support envelope signal.

Filtering Real Signal

Suppose an envelope input to a filter block represents a real signal x(t), and suppose h(t) is the impulse response of the designed filter based on user specification. Then the output envelope represents a real signal y(t). y(t) is the output of the filter and can be expressed as x(t) --> h(t) --> y(t) or y(t) = x(t) * h(t), (where * represents convolution operator). In other words, when an envelope represents a real signal, the conventional filtering process is applied.

Please note that SystemVue data flow filter blocks are actually implemented as either FIR or IIR digital filters.

Bandpass and Bandstop Filtering for Analytic Signals

Suppose an envelope input to a **bandpass** filter block represents an analytic signal $x_a(t) = (x_i(t) + j\, x_q(t)) \exp(j\, 2\, \pi\, f_c\, t)$. Suppose the bandpass filter is specified by center frequency *FCenter*, and/or passband bandwidth *PassBandwidth*, and/or stopband bandwidth *StopBandwidth*, as well as other parameters. Then the SystemVue bandpass filter block filters the analytic signal in the following steps:

1. Design a **lowpass** filter $h_l(t)$ with passband frequency *PassFrequency* = *PassBandwidth*/2 and/or stopband frequency *StopFrequency* = *StopBandwidth*/2, along with the other parameters.
2. Convert the input complex envelope $x_c(t) = x_i(t) + j\, x_q(t)$ with characterization frequency f_c to an intermediate complex envelope $x'_c(t) = x'_i(t) + j\, x'_q(t)$ with characterization frequency *FCenter*. This can be done by $x'_c(t) = x'_i(t) + j\, x'_q(t) = (x_i(t) + j\, x_q(t)) \exp(j\, 2\, \pi\, (f_c - FCenter)\, t)$
3. Filter the intermediate I and Q signals separately by the same lowpass filter $h_l(t)$. $x'_i(t)$ --> $h_l(t)$ --> $y'_i(t)$ and $x'_q(t)$ --> $h_l(t)$ --> $y'_q(t)$

4. Convert the intermediate output complex envelope $y'_c(t) = y'_i(t) + j\, y'_q(t)$ with characterization frequency *FCenter* back to the output complex envelope $y_c(t) = y_i(t) + j\, y_q(t)$ with the same characterization frequency as input f_c. This can be done by $y_c(t) = y_i(t) + j\, y_q(t) = (y'_i(t) + j\, y'_q(t))\, \exp(j\, 2\, \pi\, (\, FCenter - f_c\,)\, t)$
5. Finally, the output envelope represents the analytic signal $y_a(t) = (\, y_i(t) + j\, y_q(t)\,)\, \exp(j\, 2\, \pi\, f_c\, t)$.

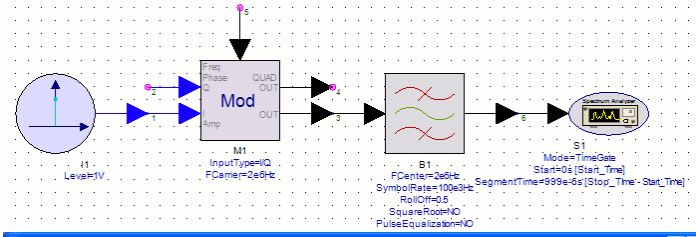
The **bandstop** filter blocks work in the similar way.

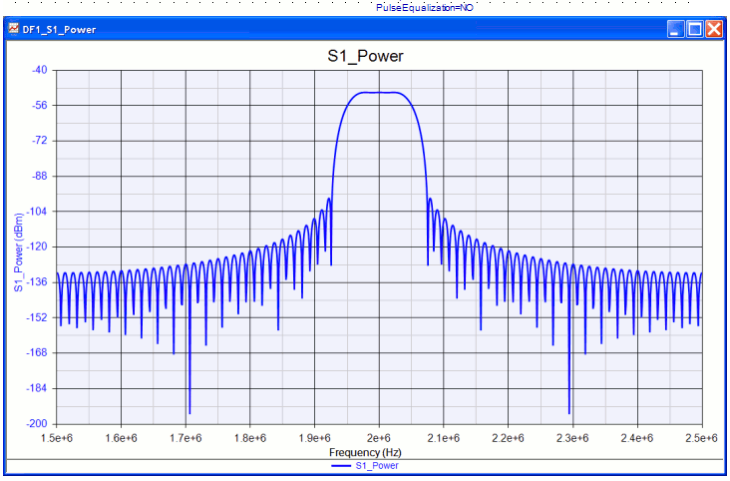
1. Design a **highpass** filter $h_h(t)$ with passband frequency *PassFrequency* = *PassBandwidth*/2 and/or stopband frequency *StopFrequency* = *StopBandwidth*/2, along with the other parameters.
2. Convert the input complex envelope $x_c(t) = x_i(t) + j\, x_q(t)$ with characterization frequency f_c to an intermediate complex envelope $x'_c(t) = x'_i(t) + j\, x'_q(t)$ with characterization frequency *FCenter*. This can be done by $x'_c(t) = x'_i(t) + j\, x'_q(t) = (x_i(t) + j\, x_q(t))\, \exp(j\, 2\, \pi\, (\, f_c - FCenter\,)\, t)$
3. Filter the intermediate I and Q signals separately by the same highpass filter $h_h(t)$. $x'_i(t) \dashrightarrow h_h(t) \dashrightarrow y'_i(t)$ and $x'_q(t) \dashrightarrow h_h(t) \dashrightarrow y'_q(t)$
4. Convert the intermediate output complex envelope $y'_c(t) = y'_i(t) + j\, y'_q(t)$ with characterization frequency *FCenter* back to the output complex envelope $y_c(t) = y_i(t) + j\, y_q(t)$ with the same characterization frequency as input f_c. This can be done by $y_c(t) = y_i(t) + j\, y_q(t) = (y'_i(t) + j\, y'_q(t))\, \exp(j\, 2\, \pi\, (\, FCenter - f_c\,)\, t)$
5. Finally, the output envelope represents the analytic signal $y_a(t) = (\, y_i(t) + j\, y_q(t)\,)\, \exp(j\, 2\, \pi\, f_c\, t)$.

Let *FUpper* denote *FCenter* + *PassBandwidth*/2 and represent upper passband edge frequency, and let *FLower* denote *FCenter* - *PassBandwidth*/2 and represent lower passband edge frequency. The bandpass/bandstop filter block with analytic input signal has its frequency response linearly symmetrical about FCenter. However, the bandpass/bandstop filter block with real input signal has its frequency response geometrically symmetrical about the square root of *FUpper*FLower*.

The benefit of this envelope filtering technology is that we can use smaller sampling rate to perform bandpass and bandstop filtering of analytic signals by lowpass and highpass filtering of complex envelopes.

The following images show the schematic and the spectrum of an impulse source that is modulated to 2M hz in analytic signal format and then filtered by a bandpass raised cosine filter centered at 2M hz.





Lowpass and Highpass Filtering for Analytic Signal

Suppose an envelope input to a **lowpass** filter block represents an analytic signal $x_a(t) = (\, x_i(t) + j\, x_q(t)\,)\, \exp(j\, 2\, \pi\, f_c\, t)$. Then the SystemVue lowpass filter block filters the analytic signal in the following steps:

1. Design a **lowpass** filter $h_l(t)$ based on user specification.
2. Convert the input complex envelope $x_c(t) = x_i(t) + j\, x_q(t)$ with characterization frequency f_c to an intermediate complex envelope $x'_c(t) = x'_i(t) + j\, x'_q(t)$ with characterization frequency 0. This can be done by $x'_c(t) = x'_i(t) + j\, x'_q(t) = (x_i(t) + j\, x_q(t))\, \exp(j\, 2\, \pi\, (\, f_c - 0\,)\, t)$

3. Filter the intermediate I and Q signals separately by the same lowpass filter $h_l(t)$. $x'_i(t)$ --> $h_l(t)$ --> $y'_i(t)$ and $x'_q(t)$ --> $h_l(t)$ --> $y'_q(t)$
4. Convert the intermediate output complex envelope $y'_c(t) = y'_i(t) + j\ y'_q(t)$ with characterization frequency 0 back to the output complex envelope $y_c(t) = y_i(t) + j\ y_q(t)$ with the same characterization frequency as input f_c. This can be done by $y_c(t) = y_i(t) + j\ y_q(t) = (y'_i(t) + j\ y'_q(t))\ \exp(j\ 2\ \pi\ (\ 0 - f_c\ )\ t)$
5. Finally, the output envelope represents the analytic signal $y_a(t) = (\ y_i(t) + j\ y_q(t)\ )\ \exp(j\ 2\ \pi\ f_c\ t)$.

The highpass filter blocks work in the similar way.

1. Design a **highpass** filter $h_h(t)$ based on user specification.
2. Convert the input complex envelope $x_c(t) = x_i(t) + j\ x_q(t)$ with characterization frequency f_c to an intermediate complex envelope $x'_c(t) = x'_i(t) + j\ x'_q(t)$ with characterization frequency 0. This can be done by $x'_c(t) = x'_i(t) + j\ x'_q(t) = (x_i(t) + j\ x_q(t))\ \exp(j\ 2\ \pi\ (\ f_c - 0\ )\ t)$
3. Filter the intermediate I and Q signals separately by the same highpass filter $h_h(t)$. $x'_i(t)$ --> $h_h(t)$ --> $y'_i(t)$ and $x'_q(t)$ --> $h_h(t)$ --> $y'_q(t)$
4. Convert the intermediate output complex envelope $y'_c(t) = y'_i(t) + j\ y'_q(t)$ with characterization frequency 0 back to the output complex envelope $y_c(t) = y_i(t) + j\ y_q(t)$ with the same characterization frequency as input f_c. This can be done by $y_c(t) = y_i(t) + j\ y_q(t) = (y'_i(t) + j\ y'_q(t))\ \exp(j\ 2\ \pi\ (\ 0 - f_c\ )\ t)$
5. Finally, the output envelope represents the analytic signal $y_a(t) = (\ y_i(t) + j\ y_q(t)\ )\ \exp(j\ 2\ \pi\ f_c\ t)$.

Reference

1. S. Haykin, *Communication Systems*, 3rd edition, Wiley, 1994.

Fixed Point Simulation

SystemVue provides an extensive set of Fixed Point parts/models in its *Hardware Design Library* (hardware). *SystemVue FixedPoint Data Type* (users) is similar in computational behavior of SystemC ™ 2.2 fixed point type based on IEEE Std. 1666 ™ Language Reference Manual (LRM) .

For detailed description of SystemC ™ 2.2 fixed-point type, consult section 7.10 of SystemC ™ IEEE Std. 1666 ™ Language Reference Manual (LRM)

Hardware Design Parts in *Hardware Design Library* (hardware) can be used to build, simulate and analyze fixed point systems. A range of functions from low level logic elements to more advanced signal processing parts such as *filters* (hardware) and *fast Fourier transforms (FFTs)* (hardware) are available.

The fixed-to-float and float-to-fixed conversion parts provide a means of interfacing fixed point components with other SystemVue blocks. Hardware Design parts can also be configured to automatically collect information on dynamic range, overflows and underflows and be shown in Fixed Point Analysis Table in order to help with system optimization.

When used in conjunction with the *HDL Code Generation* (sim), Verilog/VHDL RTL code describing the fixed point system can be automatically generated. See *HDL Code Generation, in the Simulation > Data Flow* (sim) section for more information.

Using Fixed-Point Parts

Conversion between Fixed-Point and Non-Fixed-Point Data

SystemVue does not support automatic conversion from non-fixed-point to fixed-point data. To convert floating-point to fixed-point data use *FloatToFxp* (hardware) part. You could also convert from fixed-point to floating-point data using *FxpToFloat* (hardware) part.

Fixed-Point Precision at the Input Port

SystemVue resolves the fixed-point precision at the input port automatically based on the output precision of the previous component in the data flow schematic. A user does not have to specify the precision for the input port.

Fixed-Point Precision at the Output Port

For most of the parts you need to specify the fixed-point precision at the output port explicitly. There are cases where output port precision is tightly coupled with fixed-point precision at the input ports, for example, *MuxFxp* (hardware), and *DelayFxp* (hardware), in such cases it may not be required to specify output precision. The fixed-point models, which require a user to specify output port precision, share the following common parameters.

Wordlength

This parameter is used to specify the total number of bits in the fixed-point, including fractional and integer bits.

IntegerWordlength

This parameter is used to specify the number of integer bits in the fixed-point including the sign bit. Based on the values of Wordlength and IntegerWordlength the fixed point precision could have one of the following three scenarios

- **Wordlength < IntegerWordlength:** There will be (IntegerWordlength - Wordlength) zeros added between the Least Significant Bit (LSB) and the binary point.
- **0 <= IntegerWordlength <= Wordlength:** The fixed point will have IntegerWordlength integer bits and (Wordlength - IntegerWordlength) fractional bits. The binary point will be fully containted in the precision specified.
- **IntegerWordlength < 0:** There are (-IntegerWordlength) signed-extended bits added between binary point and Most Significant Bit (MSB).

Please consult section 7.10.1 of SystemC ™ IEEE Std. 1666 ™ Language Reference Manual (LRM) for more details.

IsSigned

This parameter is used to specify that either the fixed-point is signed or un-signed type.

Quantization

This method is used to specify the quantization mode for the fixed-point. There are seven possible modes

- **RND:** Rounding to plus infinity. The corresponding SystemC ™ mode is SC_RND.
- **RND_ZERO:** Rounding to zero. The corresponding SystemC ™ mode is SC_RND_ZERO.
- **RND_MIN_INF:** Rounding to minus infinity. The corresponding SystemC ™ mode is SC_RND_MIN_INF.
- **RND_INF:** Rounding to infinity. The corresponding SystemC ™ mode is SC_RND_INF.
- **RND_CONV:** Convergent rounding. The corresponding SystemC ™ mode is SC_RND_CONV.
- **TRN:** Truncation. The corresponding SystemC ™ mode is SC_TRN.
- **TRN_ZERO:** Truncation to zero. The corresponding SystemC ™ mode is SC_TRN_ZERO.

Please consult section 7.10.9 of SystemC ™ IEEE Std. 1666 ™ Language Reference Manual (LRM) for more details.

Overflow

This parameter is used to specify overflow mode. The overflow occurs when specified precision is not enough to hold the value. There are five possible modes

- **SAT:** Saturation. The corresponding SystemC ™ mode is SC_SAT.
- **SAT_ZERO:** Saturation to zero. The corresponding SystemC ™ mode is SC_SAT_ZERO.
- **SAT_SYM:** Symmetrical saturation. The corresponding SystemC ™ mode is SC_SAT_SYM.
- **WRAP:** Wrap-around. The corresponding SystemC ™ mode is SC_WRAP.
- **WRAP_SM:** Sign magnitude wrap-around. The corresponding SystemC ™ mode is SC_WRAP_SM.

Please consult section 7.10.9 of SystemC ™ IEEE Std. 1666 ™ Language Reference Manual (LRM) for more details about overflow mode.

SaturationBits

This parameter is used to provide number of saturation bits for WRAP and WRAP_SM Overflow modes.

Please consult Table 39 and Table 40 in section 7.10.9 of SystemC ™ IEEE Std. 1666 ™ Language Reference Manual (LRM) for more details about saturation bits.

TypeOverride

All fixed point models share a common parameter **TypeOverride**. This parameter is used to override the numeric format of fixed point block. There are 3 possible modes.

- **OFF:** The Fixed point model will be simulated with fixed-point precision. This is the default mode of operation.
- **Double:** The Fixed point model will be simulated with double floating-point precision.
- **Integer:** The Fixed point model will be simulated with integer precision.

> **Note**
> SystemVue does not support automatic conversion to fixed-point, you may need to consider this when switching the TypeOverride parameter.

HDL Code Generation and Automatic HDL Cosimulation.

SystemVue allows its users to generate VHDL or Verilog using the synthesizeable parts in its *Hardware Design Library* (hardware). Please consult *HDL Code Generation* (sim) document for further details. SystemVue also provides its users the ability to automatically *Cosimulate SystemVue generated HDL* (sim) with ModelSim ™ SE HDL simulator.

Cosimulating with User HDL

While simulating fixed-point design, it is possible in SystemVue to cosimulate with user written HDL using ModelSim ™ SE HDL simulator by using *HdlCosim* (sim) part. However, *HDL Code Generation* (sim) cannot synthesize a fixed-point model that includes user written HDL code, because *HdlCosim* (sim) part is not synthesizeable. Also automatic HDL cosimulation is not possible with user written code. Please consult *HDL Cosimulation* (sim), and *HdlCosim* (sim) part document for more details.

Fixed Point Analysis Table

The fixed point analysis table is created automatically after simulating a design that contains fixed point parts and when *data flow analysis options* (sim) are set properly to collect fixed point analysis data.

The fixed point analysis table contains 8 columns:

1. Part- the instance name of the design block.
2. Model - the model used in the Part.
3. Signal - the name of the fixed point signal.
4. Precision - the fixed point number representation (± indicates signed, the first number in the angle brackets is the register word length, the second number in the angle brackets is the length of the integer part).
5. Overflows - shows the total number and percentage of overflows on the signal.
6. Underflows - shows the total number and percentage of underflows on the signal.
7. Max - the maximum value of the signal.
8. Min - the minimum value of the signal.

As overflows and underflows are generally undesirable, they are shown in red.

Fixed Point Examples

SystemVue is shipped with examples using fixed point analysis with HDL code generation. These examples are located in the directory <SystemVue installation directory>\Examples\Hardware Design.

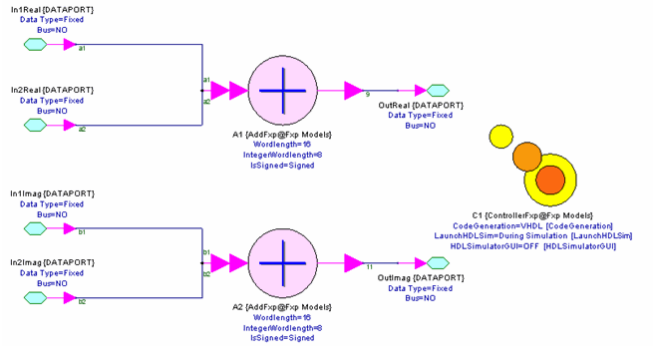
HDL Code Generation

SystemVue provides its users with an easy path from schematic design to the hardware. This could be done by using HDL Code Generation capability of SystemVue. A user created SystemVue *sub-network model* (users), using only synthesizeable Fixed Point parts from *Hardware Design Library* (hardware) can be used to generate VHDL/Verilog for the sub-network.

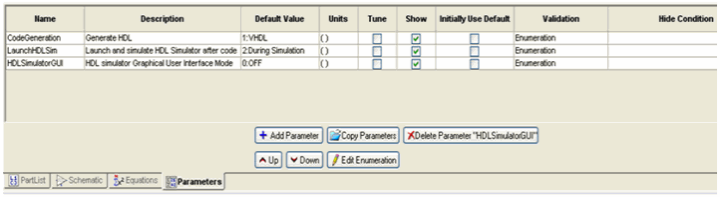
In this tutorial we will go through a simple example to understand the design flow to generate HDL Code. The same design flow can be used for more complex designs. We will create a Fixed point design for a Complex Adder, generate VHDL for the Complex Adder and performs functional verification of the generated VHDL.

Generating Fixed Point Sub-Network Model

- If you have note done so then read and understand *Sub-Network Models* (users) documentation.
- Create a sub-network model as shown below.



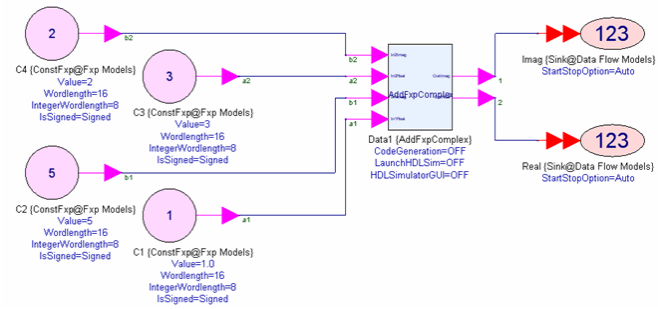
- Note that ControllerFxp component is included in the design. Also note that the values of parameters CodeGeneration, LaunchHDLSim and HDLSimulatorGUI are assigned the model parameter names.
- Set the Sub-Network parameters as follows



- For the *sub-network model* (users) parameters, edit the enumeration type for *CodeGeneration* to be *ControllerFxp_CodeGeneration*, for *LaunchHDLSim* to be *ControllerFxp_LaunchHDLSim* and for *HDLSimulatorGUI* to be *ConstrollerFxp_HDLSimulatorGUI* from the library *Fxp Enums*.

> Parameterizing the sub-network model to control ControllerFxp is not required but helps if you would like to use the same sub-network model for Fixed point simulation and automatic HDL Cosimulation.

- Create a top level design and simulate the sub-network model to make sure its correct functionality. Make sure that parameter *CodeGeneration=OFF*. An example top-level design is shown below



> **Warning**
> The fixed point sub-network model used to generate HDL must contains only synthesizeable Fixed Point parts from *Hardware Design Library* (hardware). For example *FloatToFxp* (hardware) part is not synthesizeable. To ensure that a Fixed Point part is synthesizeable, consult its documentation.

Generating the HDL and HDL Simulation

- Change the value of model parameter *CodeGeneration* to *VHDL*. If you have not parameterized your sub-network model then select *CodeGeneration=VHDL* for ControllerFxp component inside the sub-network.
- Simulate the top level design, it will perform the fixed point simulation and generates the corresponding *VHDL* for the sub-network model instance in the sub-directory ***<schematic design name>_<sub-network model instance name>_HDL\hdl*** under the same directory where the workspace containing the design is located. The name of the file containing the top level VHDL will be ***<sub-network model name>.vhd***.

If you have installed ModelSim SE and it is configured to run from command line i.e. your operating system's PATH variable points to ModelSim SE, then you can also invoke HDL simulation **After Simulation** using the test vectors generated by SystemVue simulation. Alternatively you can use automatic *HDL Cosimulation* (sim) **During Simulation** to make sure that generated VHDL is functionally correct. The automatic *HDL Cosimulation* (sim), performs inter process communication between SystemVue and ModelSim to process data by HDL simulator in real time while SystemVue simulation is running. The Hdl simulation **During Simulation** first generate the HDL for the sub-network and then runs the HDL portion of the design in ModelSim and rest in SystemVue using inter process communication.

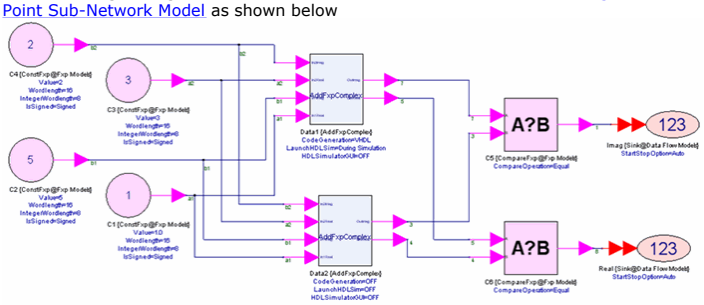
- Change the value of model parameter *LaunchHDLSim=After Simulation* and run the simulation. This will simulate the design in SystemVue and at the end of simulation starts ModelSim and run ModelSim simulation using the test vectors generated by the SystemVue simulation.
- To perform HDL simulation **During Simulation** change the value of model parameter *LaunchHDLSim=During Simulation* and run the simulation. This will perform the HDL Cosimulation in the back ground if *HDLSimulatorGUI=OFF*.
- Now change the value of model parameter *HDLSimulatorGUI=ON* and run the simulation. This will bring up ModelSim GUI and halts the SystemVue simulation which is waiting for data from ModelSim. This interactive mode can be used for debugging HDL code. To resume the simulation, either issue the command **run** inside ModelSim to run for a single step or **run -all** to run the complete simulation.

> - Currently only VHDL is supported with *LaunchHDLSim=During Simulation*.
> - If *HDLSimulatorGUI=ON* then close the ModelSim before starting simulation again. Otherwise you will require additional license for new instance of ModelSim.

Testing for Functional Equivalency

This section covers that how to perform functional equivalency test to prove that SystemVue generated HDL is functionally equivalent to the fixed point model for which HDL was generated. You must have ModelSim SE installed properly, see section Generating the HDL and HDL Simulation for more details.

- Create a design using the sub-network model created in section Generating Fixed Point Sub-Network Model as shown below



- This design includes two instances of the sub-network model. One instance is configured to generate VHDL and LaunchHDLSimulation=During Simulation, the other instance is configured not to generate HDL and perform only fixed point simulation.
- Both instances are fed with the same inputs.
- The output of both instances are compared using ***CompareFxp* (hardware)** part with CompareOperation=Equal. The output of ***CompareFxp* (hardware)** is '1' to indicate true and is '0' to indicate false.
- Run the simulation, and observe the results. If the output of all *CompareFxp* (hardware) instances is always '1' that means the generated HDL is functionally equivalent to original fixed point model.

> **Warning**
> - Currently only VHDL can be used with *LaunchHDLSimulation=During Simulation* therefore it is not possible to perform functional equivalency test for SystemVue generated Verilog.
> - If you are using any delay or changing the sample rate in the sub-network model then you cannot use any downstream component that is backward reachable simultaneous to a sub-network model instances using *LaunchHDLSimulation=During Simulation* and sub-network model instances **not** using *LaunchHDLSimulation=During Simulation*. The functional verification can still be performed by connecting the output of sub-network model instances to two different sinks and comparing the data in the dataset. In the above example it means not using the *CompareFxp* (hardware).

The reason that outputs of sub-network model instances (one with HDL Cosimulation and the other with fixed-point simulation) cannot be combined at the input of a downstream component is because sub-network model with LaunchHDLSimulation=During Simulation is replaced by a single *HdlCosim* (sim) model which is a uni-rate model, where as the other instance uses the fixed-point models in the simulation.

Understanding the Generated HDL

The generated HDL will be located inside the sub-directory ***<schematic design name>_<sub-network model instance name>_HDL\hdl*** under the same directory where the workspace containing the design is located. The name of the file containing the top level HDL will be ***<sub-network model name>.vhd*** for VHDL and ***<sub-network model name>.v*** for verilog.
The number of input/output ports will be same if there is no sequential component is used, for example the Complex Adder design above. However, in case of using sequential components such as *DelayFxp* (hardware), *RegisterFxp* (hardware) etc, the resulting HDL will have extra **CLK** and **RESET** input ports, it may also have a **DataInEnable** input port as well, depending upon the design. This **DataInEnable** control input port must be used to indicate when the input data is valid ('1') and when it is not ('0'). There may be a **DataOutEnable** output port which may be used to detect when the output of the HDL is valid ('1') and when it is not valid ('1').
Other than the top level HDL file other HDL files are also included. Most of these HDL files contains HDL for sub-components used to create the top level HDL and must be included in any synthesis/simulation tool along with the top-level HDL.

SystemVue Examples

SystemVue ships with hardware design examples which are configured or can be configured to generate HDL. These examples are installed under **<SystemVue install directory>\Examples\Hardware Design**.

HDL Cosimulation

About HDL Cosimulation

With the SystemVue HDL Cosimulation feature you can simulate components represented in a hardware description language (HDL) in the same schematic with other SystemVue components. This integrated capability provides complete design flexibility, and complements other SystemVue features, including HDL generation.

The ability to design all portions of a communications product in one integrated environment can eliminate design errors resulting from disconnects among design teams. By cosimulating with HDL designs, you can easily incorporate your existing HDL intellectual property into new designs, or you could cosimulate with SystemVue generated HDL.

With HDL cosimulation, you can test hardware defined in HDL with a DSP algorithm, or use an algorithm written in HDL within an existing SystemVue design. This cosimulation capability in one design environment makes it easy to test HDL components along with other SystemVue design components and see the effect on the entire system.

Note

Currently only VHDL is supported with the ModelSim ™ HDL simulator from Mentor Graphics as HDL simulator.

Requirements for HDL Cosimulation

To run HDL Cosimulation, you must have ModelSim ™ SE PLUS 6.3g or later installed on your system. The ModelSim ™ must be setup to run from command line prompt. This means that your PATH environment settings and ModelSim ™ license must be setup appropriately prior to running HDL Cosimulation in SystemVue. If PATH has not been setup properly during ModelSim installation, consult help and support of your Operating System to learn how to setup environment variables.

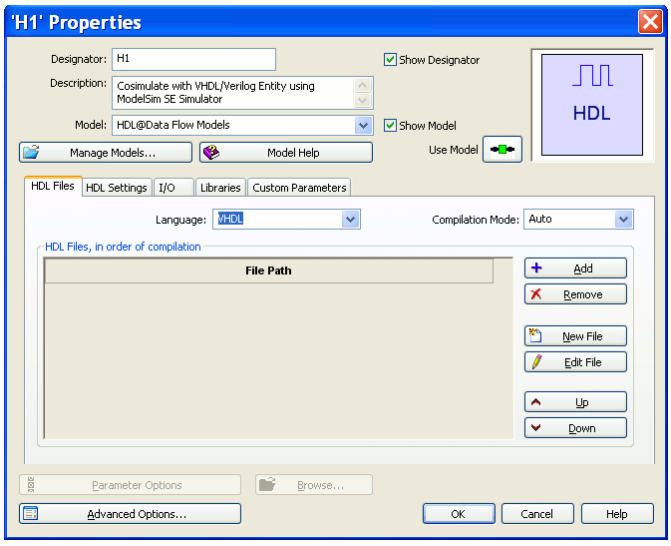
Prerequisite

Before continuing with this document please make sure that you have read and understood the following two sections:

1. *Create Your First Data Flow Simulation* (sim)
2. *Nets, Connection Lines and Buses* (users)

Quick Start Guide

- If you have not already done it, please read and understand *Create Your First Data Flow Simulation* (sim) first.
- Create a New Design with Schematic and name it *HDLCosimTest.*
- Select the library *Hardware Design* and category *HDL Cosimulation*, select *HDL* part and place its instance in the design.
- Double click on *HDL Part* (hardware) to open its properties as shown below

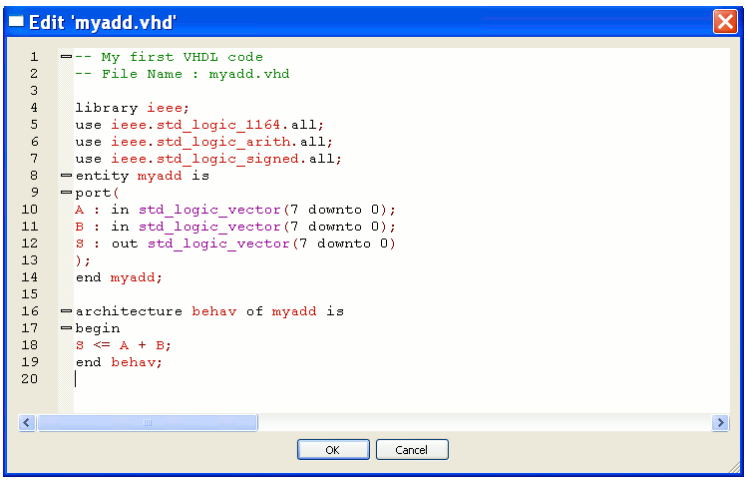


- Click on *New File* button and browse to the place where you would like to create file. In the File Name field write **myadd.vhd** and click *Open*. A VHDL file editor will open, write the following VHDL code in the editor.

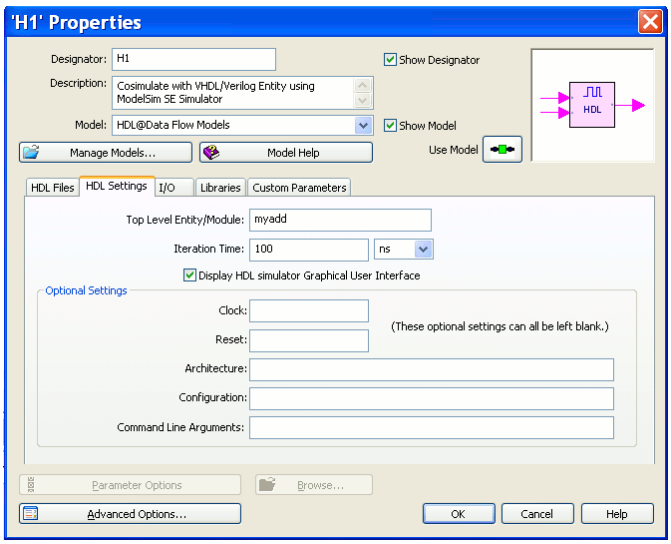
```
-- My first VHDL code
-- File Name : myadd.vhd
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
entity myadd is
port(
A : in std_logic_vector(7 downto 0);
B : in std_logic_vector(7 downto 0);
S : out std_logic_vector(7 downto 0)
);
end myadd;
architecture behav of myadd is
begin
```

```
S <= A + B;
end behav;
```

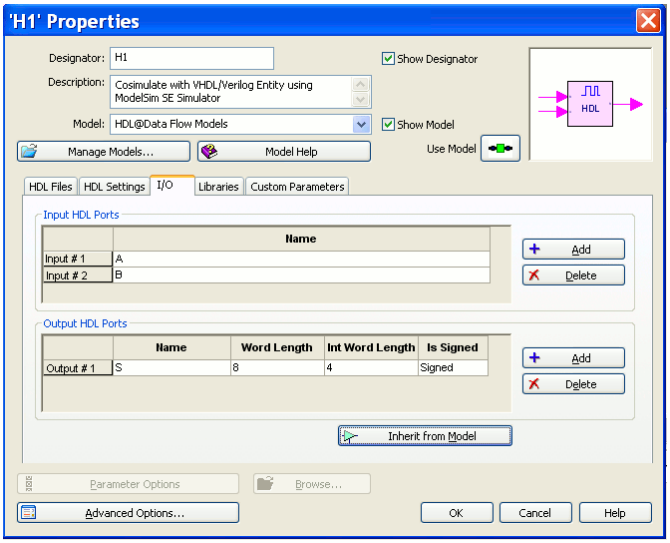
- Note the VHDL syntax highlighting and line numbers in the VHDL file editor as shown below



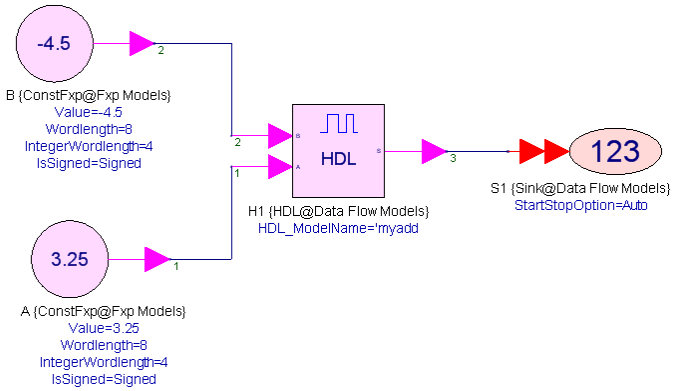
- Click *OK* to save the file, note the VHDL **entity** name *myadd* and VHDL **port** names *A* , *B*, and *S* and width of output port (8) before clicking OK.
- Click on **HDL Settings** tab and enter **myadd** in the **Top Level Entity/Module** field, also select **Display HDL Simulator Graphical User Interface** as shown below



- Under I/O tab add two input ports *A* and *B*. The port names must match the **in** type port names in the above VHDL code.
- Under I/O tab add once output port *S*. The port name must match the **out** type port name in the above VHDL code.
- Specify the fixed point formatting you like for each output port. For now use **Word Length** equals to 8 and for **Int Word Length** use 4 and select *Signed* for **Is Signed** . The I/O tab should now look like as follows



- Click *OK*. This will create a custom Symbol for your HDL model with appropriately named ports matching the port names in VHDL code.
- Select the library *Algorithm Design* and category *Sinks*, select *Sink* part and place its instance in the design. Connect the output of *HdlCosim* part to *Sink* input.
- Select the library *Hardware Design* and category *Fixed Point*, select *ConstFxp* and place 2 instances in the design. Connect each instance of *ConstFxp* to one input port of your customized *HDL* instance.
  - Rename the source labels to **A** and **B**.
- Create following VHDL code using any text editor of your choice and save it as **myadd.vhd** in any directory that you remember.
- Since our VHDL code has two inputs of 8-bits each, we need to modify our sources accordingly to generate proper fixed point values.
  - For both sources **A** and **B** on the schematic select *WordLength*=8 to match std_logic_vector(7 down to 0) of VHDL ports **A** and **B** respectively.
  - For both sources **A** and **B** on the schematic select *IntegerWordLength*=4
  - For source **A** select *value*=3.25 and for source **B** select *value*=-4.5
- The schematic should look like as follows.



- Add a *Data Flow Analysis*, if you don't know how to do it then read section *Add the Data Flow Analysis* (sim)
- Run Simulation, it will compile the VHDL code and open ModelSim UI for you, because we have selected **Display HDL Simulator Graphical User Interface**.
- All the signals in your top level entity will also be added to ModelSim Wave as well.
- In ModelSim issue the command '*run -all*' to start simulating the design.
- Watch for the signal values in ModelSim.
- Go back to SystemVue and examine the data in SystemVue. If you do not know how to do it then please follow the instructions in *Examine the Data* (sim).
- Unselect **Display HDL Simulator Graphical User Interface** and simulate the design again. This time ModelSim will run in the background. You could examine the results in SystemVue once the simulation is complete.
- Double click on HDL part to open its properties and make sure **Compilation Mode** is *Auto* and simulate the design again. This time your VHDL code will not be re-compiled, instead already compiled VHDL code in *work* library from previous simulation will be used. This is convenient if it takes longer to re-compile your HDL code.
- Now under HDL Files tab select myadd.vhd and click **Edit File**, slightly modify the file by adding a space character and then click *OK*. Make sure **Compilation Mode** is *Auto* and simulate the design again, this time VHDL code will be compiled because we have modified it. You can choose **Compilation Mode** to be **Auto**, **Always** or **Never** based on your need.

Understanding HDL Cosimulation

With the HDL cosimulation feature, SystemVue has been configured to cosimulate with ModelSim HDL simulator. In this use model, you first create the HDL design, use HDL model HDL editor to write your HDL code or use SystemVue to generate HDL design. If you have written your own HDL design then it must be error free. This means that you

must be able to compile and simulate the HDL design with ModelSim before cosimulation.

If your HDL code is not compiled, you can use SystemVue to compile the code before cosimulation. Cosimulation requires information regarding the VHDL entity or Verilog module that you want to cosimulate with. This is used to generate HDL wrappers that incorporate your HDL code and SystemVue specific C-interface code to create an inter-process communication (IPC) link between SystemVue and the HDL simulator.

You can run HDL cosimulation in graphical user interface mode to monitor the HDL simulation or debug you HDL code in ModelSim environment, please read ModelSim documentation for HDL debugging. The cosimulation can also be run in the background processing mode.

HDL cosimulation uses the SystemVue Data Flow Analysis, in which numeric signals are consumed and produced by the HDL model. There is no timing information communicated between SystemVue and the HDL simulator. SystemVue sends data into the HDL simulator and receives data without any knowledge of the HDL timing. Furthermore HDL cosimulation component is a uni-rate component.

The HDL model is a numeric model. Because the HDL simulation itself is time driven, it is run for user specified amount of time whenever the HDL model reads data at its input ports prior to collecting data from HDL simulator. The time scale used by the HDL simulator is independent of the SystemVue simulation.

Each time the HDL model receives input values from other SystemVue components, it sends these input values to HDL simulator which uses them to perform the HDL simulation. Once the HDL simulator is finished with its processing, it passes the simulation results back to SystemVue. These passed values are then the outputs of the HDL model, and thus the simulation cycle continues. This cycle repeats as many times as the SystemVue Data Flow Analysis simulation requires. Each time the HDL model is invoked, the HDL simulation duration is determined by the value of the IterationTime parameter in the HDL cosimulation component. You must determine how long the HDL simulator should run before its outputs are sent back to the HDL model. This timing information should not be confused with the timing used in other SystemVue models.

From the HDL simulator engine's point of view, the SystemVue input interface is viewed as forcing values onto the ports. At the output interface of the HDL model, the results are converted back into SystemVue format and sent to the other connecting SystemVue components.

Please note that, input ports of the HDL model can only be connected to the output of a fixed point model. Automatic conversion from other type of ports to fixed point is not supported.

Supported HDL Port Types

HDL cosimulation currently supports various bit and bit-vector type HDL ports; they are all mapped to the SystemVue fixed point data type port.

In the case of Verilog HDL, which supports only bit and bit-vector type ports, HDL cosimulation will support any type of Verilog port. In the case of VHDL, which has a large set of data types, HDL cosimulation will only support the ports that are of bit, bit-vector, signed, unsigned, std_logic, std_logic_vector types described in the IEEE std_logic_1164 library.

Bidirectional HDL Ports

Bidirectional ports are not supported in HDL model.

Automatic Clock and Reset Signals

Using HDL Cosimulation it is possible to automatically generate Clock and Reset signals without connecting those to the input of *HDL* (hardware) model. This features enable the user to generate one clock cycle per *HDL* (hardware) model iteration which is not possible by connecting a Clock at the input port of *HDL* (hardware) model because of its unirate properties.

To generate an automatic Clock set **Clock:** field under *HDL Settings* tab to the name of the Clock port in your HDL code.

To generate an automatic Reset set **Reset:** field under *HDL Settings* tab to the name of the Reset port in your HDL code.

The autogenerated Clock is of a 50% duty cycle and has a period equal to the HDL iteration time. The positive clock edge occurs at IterationTime/2. The default value for the Reset during the first iteration is a logic low from time 0 to 1/4 times the HDL iteration time, a logic high from time 1/4 to 3/4 times the HDL Iteration time, a logic low for remaining HDL iteration time, and a logic low for iterations after that. This means that Reset signal is high during first rising edge of the Clock signal and low after.

Time-Specified Signals in User HDL Code

When HDL code has internal clocks or time-specified signals (for example, wait statements in VHDL code) the HDL cosimulation may keep running until all the events in the user HDL code are processed. The number of events generated in user HDL code can be infinite (for example, when you have an internal clock).

You can avoid using an internal clock and use the SystemVue *Clock* instead (refer to the section *Automatic Clock and Reset Signals* above). If this is not possible, then infinite event processing can be avoided if you know how long the HDL simulation needs to run to complete the cosimulation, with all of the SystemVue iterations. Different simulators have different mechanisms to break a simulation after a certain simulation time. Here is an example using ModelSim:

1. Use the ModelSim simulator to create a file called *test.do*. An example *test.do* may look like this:

```
run 11000
quit -f
```

2. Set **Command Line Arguments** field under HDL Settings to *-do test.do*. This stops the simulation after 11000 nsec. (Refer to CmdArgs on the HdlCosim component block.)

The total run time can be calculated as equal to:

The number of SystemVue iterations (depends on the Data Flow Analysis setup and the different sinks used in the design) multiplied by the IterationTime specified on the HdlCosim block.

Alternatively, you can also open the ModelSim UI mode and use multiple *run 100* commands to see how long it takes before the message *VHDL Cosimulation has completed..* appears in the ModelSim UI. This time can then be used to create the *test.do* file.

Do not use the *run-all* command, which will process all the events in the HDL simulation.

HDL Part and Model

HDL cosimulation uses *HDL Part* (hardware) and the corresponding *HDL* (hardware) model which is available under *Hardware Design* under the category *HDL Cosimulation*. The *HDL* (hardware) model provides cosimulation with ModelSim ™ SE Plus 6.3g or later. If the user code has not been compiled, *HDL* (hardware) model can compile the user code before cosimulation or use existing compiled *HDL* (hardware) code depending on the *HDL* (hardware) model settings. Please read *HDL model documentation* (hardware) for more details about using this model.

MATLAB Cosimulation

The MATLAB_Cosim part provides an interface between SystemVue and MATLAB®, a numeric computation and visualization environment from The MathWorks, Inc.

SystemVue handles the conversion of data to and from MATLAB. Since MATLAB cosimulation involves multiple environments and associated inter-process communication, the installation and pre-simulation configuration must be precise and correct for the infrastructure to work as expected. To ensure proper operation, the instructions provided in Setting Up MATLAB Cosimulation must be followed.

MATLAB cosimulation using multiple MATLAB_Cosim parts does not work in multithreaded simulations. If you are using multiple MATLAB_Cosim parts, the multithreaded simulation option will be automatically overridden and the simulation will run in a single thread.

Supported MATLAB Versions

MATLAB Cosimulation requires MATLAB and supports MATLAB version 7.7 (Release 2008b).

There is also a good chance that other MATLAB versions would work in SystemVue as well.

Setting Up MATLAB Cosimulation

MATLAB must be configured correctly before using cosimulation. If MATLAB_Cosim is run and MATLAB is not configured correctly, SystemVue will report an error.

You should be able to manually launch MATLAB from the Windows Command Prompt. If MATLAB does not start that way, MATLAB_Cosim will not work either. This may be due to the fact that your PATH variable exceeds the Windows limitation, in which case you may want to reduce it or create a batch script with a reduced PATH variable to launch SystemVue. Alternatively you may want to just copy all of the MATLAB DLLs (libeng.dll, libut.dll, etc.) from its *bin\win32* folder to the SystemVue *bin* folder (but remember to replace them after an upgrade).

In addition, MATLAB COM server must be properly registered.

For most Windows users, SystemVue MATLAB cosimulation will work as expected when MATLAB and SystemVue are installed.
Typically, the MATLAB installer registers the COM components in the Windows registry.

To manually register COM components run

```
matlab /regserver
```

This may be necessary if you have multiple versions of MATLAB on your system and SystemVue MATLAB cosimulation fails with an error *"Matlab could not be invoked"*.

Simulating with MATLAB

The MATLAB interpreter's working directory is set to the *ScriptDirectory* parameter, if it is given. Any custom MATLAB models will be searched there, and any output files will be written there. If you leave the *ScriptDirectory* parameter blank, MATLAB_Cosim will use the SystemVue workspace folder as its working directory.

MATLAB will search for scripts in the folders in your MATLAB path. Normally your MATLAB scripts should be placed either in the folder with the workspace file using them, or in your personal *MATLAB* folder (usually found under *My Documents*). You can also add more folders to your MATLAB path using *Set Path...* dialog in MATLAB.

Writing Functions for MATLAB_Cosim

There are several ways in which MATLAB commands can be specified in the MATLAB_Cosim in the *MatlabFunction* parameter.

If only a MATLAB function name is given for this parameter, the function is applied to the inputs in order. The function's outputs are sent to the model's outputs.

For example, specifying *eig* means to perform the eigendecomposition of the input. The function will be called to produce one or two outputs, according to how many output ports there are. If there is a mismatch in the number of inputs and outputs between the MATLAB_Cosim part and the MATLAB function, then an error will be reported by MATLAB.

You may also explicitly specify how the inputs are to be passed to a MATLAB function and how the outputs are taken from the MATLAB function. For example, consider a two-input, two-output MATLAB_Cosim part to perform a generalized eigendecomposition. The command

```
[output#1, output#2] = eig( input#1, input#2 );
```

says to perform the generalized eigendecomposition on the two input matrices, place the generalized eigenvectors on output#2, and the eigenvalues (as a diagonal matrix) on output#1. Before this command is sent to MATLAB, all "#" characters are replaced with the underscore character "_" because "#" is illegal in a MATLAB variable name.

The MATLAB_Cosim part also allows a sequence of commands to be evaluated. Continuing with the previous example, we can plot the eigenvalues on a graph after taking the generalized eigendecomposition:

```
[output#1, output#2] = eig( input#1, input#2 ); plot( output#1 );
```

When entering such a collection of commands in SystemVue, both commands appear on the same line without a new line after the semicolon. In this way, very complicated MATLAB commands can be built up.

The *MatlabSetup* and *MatlabWrapUp* parameters are called during the model's begin and wrap-up procedures. During each of these procedures, data is not passed into or out of the model.

Because the same MATLAB interpreter is used for the entire simulation, variables are preserved from iteration to iteration. For example, the output of a MATLAB_Cosim part with settings:

```
MatlabSetUp = 'x=ones(2,1);
MatlabFunction = 'output#1=x(2)/x(1); x=[x(2),sum(x)];
```

will converge on the golden mean.

Using MATLAB_Cosim as a Source or Sink

The input to MATLAB_Cosim is optional, so it can be directly used as a simulation source.

The output is not optional, it must be connected to a simulation sink. If the MATLAB code produces no output, a dummy output needs to be created. Just add "ouput#1=0" at the end of your MATLAB command, e.g.

```
MatlabSetUp = 'hold on;
MatlabFunction = 'plot(input#1,input#2); output#1=0;
MatlabWrapUp = 'pause(20);
```

Passing Parameters to MATLAB_Cosim

MATLAB commands are executed in the MATLAB environment and thus are not directly aware of the parameters existing in your SystemVue workspace. Therefore, something like *MatlabSetUp = 'm_script_param=SystemVue_param;* will not work. To pass a parameter to MATLAB command you need to use equations to create a string containing the parameter value. For instance, in the Equations tab create

```
matlab_setup_str = ['m_script_param=' num2str(SystemVue_param)];
```

and then assign that string variable directly to MatlabSetUp:

```
MatlabSetUp = matlab_setup_str
```

See *Equations* (users) for more information about SystemVue equations.

Hiding MATLAB Code

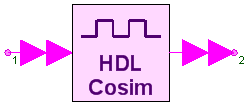
If you don't want to share your MATLAB IP with other users, you can generate MATLAB p-code files from your m-code by using the MATLAB command *pcode*.

For instance, if your MATLAB code is in the file mycode.m in the directory of your SystemVue workspace, you need to open MATLAB and in the MATLAB Command Window use *cd* command to go to that directory and execute

```
pcode mycode.m
```

You will find mycode.p in the same directory. When you run the simulation, mycode.p will be executed instead of mycode.m, and you can remove mycode.m from your workspace directory, leaving only mycode.p. The format is a non-readable binary, so your m-code is not visible to other users.

HdlCosim



Description: Cosimulates with VHDL Entity using ModelSim SE Simulator
Categories: Hdl Cosimulation

Model Parameters

Name	Description	Default	Units	Type
HdlSrcFile	Top Level HDL Source file; or a text file containing the list of HDL files, one per line, in the order of compilation.			Filename
HdlInputs	Input port names in HDL code to communicate with. It is an String Array.			None
InputPhases	Delay for updating inputs in HDL simulator within each IterationTime step. It is an integer Array.			Integer array
InputWordlengths	Number of bits in each HDL input port in the same order as in the HdlInputs parameter. It is an integer Array.			Integer array
AutoGenerateClock	Generate Clock automatically with one complete cycle per iteration: NO, YES	NO		Enumeration
HdlClockName	Name of Clock port in HDL Code			Text
AutoGenerateReset	Generate Reset automatically during first iteration: NO, YES	NO		Enumeration
HdlResetName	Name of Reset port in HDL Code			Text
HdlOutputs	Output port names in HDL code to communicate with. It is an String Array.			None
OutputWordlengths	Number of bits in each HDL output port in the same order as in the HdlOutputs parameter. It is an integer Array.			Integer array
OutputIntegerWordlengths	Formats the fixed point outputs in the same order as in the HdlOutputs parameter. It is an integer Array.			Integer array
AreOutputsSigned	Are outputs signed 's' or unsigned 'u' in the same order as in the HdlOutputs parameter. It is an String Array.			None
HdlModelName	VHDL entity[.architecture|+configuration] or Verilog module name to cosimulate with			Text
ShowAdvancedParams	Show advanced parameters: NO, YES	NO		Enumeration
HdlLibrary	HDL library that user model depends on (to map them explicitly use name=path syntax), if set to none then all the code will be compiled in work library. To avoid recompiling use "work" (remember that code must be compiled atleast once before cosimulation).			Text
HdlSimulatorGUI	HDL simulator Graphical User Interface Mode: OFF, ON	OFF		Enumeration
CmdArgs	HDL Simulator command invocation arguments, if any			Text
IterationTime	Time to run the HDL simulator before collecting the outputs	100		Integer
TimeUnit	Time resolution limit to be passed to HDL simulator: fs, ps, ns, us, ms, sec	ns		Enumeration

Input Ports

Port	Name	Signal Type	Optional
1	Input	multiple fix	YES

Output Ports

Port	Name	Signal Type	Optional
2	Output	multiple fix	YES

Warning

HdlCosim model is obsolete and should not be used anymore in any new design. Please use a better *HDL model* (hardware) instead using *HDL Part* (hardware).

Prerequisite

Before continuing with this document please make sure that you have read and understood the following two sections:

1. *Create Your First Data Flow Simulation* (sim)
2. *Nets, Connection Lines and Buses* (users)

Understanding HDL Cosimulation

With the HDL cosimulation feature, SystemVue has been configured to cosimulate with ModelSim HDL simulator. In this use model, you first create the HDL design or use SystemVue to generate HDL design. If you have written your own HDL design then it must be error free. This means that you must be able to compile and simulate the HDL design with ModelSim before cosimulation.

If your HDL code is not compiled, you can use SystemVue to compile the code before cosimulation. Cosimulation requires information regarding the VHDL entity or Verilog module that you want to cosimulate with. This is used to generate HDL wrappers that incorporate your HDL code and SystemVue specific C-interface code to create an inter-process communication (IPC) link between SystemVue and the HDL simulator.

You can run HDL cosimulation in graphical user interface mode to monitor the HDL simulation or debug you HDL code in ModelSim environment, please read ModelSim documentation for HDL debugging. The cosimulation can also be run in the background processing mode.

HDL cosimulation uses the SystemVue Data Flow Analysis, in which numeric signals are consumed and produced by the HDL cosimulation component. There is no timing information communicated between SystemVue and the HDL simulator. SystemVue sends data into the HDL simulator and receives data without any knowledge of the HDL timing. Furthermore HDL cosimulation component is a uni-rate component.

The HDL cosimulation component is a numeric component. Because the HDL simulation itself is time driven, it is run for user specified amount of time whenever the HDL cosimulation component reads data at its input ports prior to collecting data from HDL simulator. The time scale used by the HDL simulator is independent of the SystemVue simulation.

Each time the HDL cosimulation component receives input values from other SystemVue components, it sends these input values to HDL simulator which uses them to perform the HDL simulation. Once the HDL simulator is finished with its processing, it passes the simulation results back to SystemVue. These passed values are then the outputs of the

HDL cosimulation component, and thus the simulation cycle continues. This cycle repeats as many times as the SystemVue Data Flow Analysis simulation requires. Each time the HDL cosimulation component is invoked, the HDL simulation duration is determined by the value of the IterationTime parameter in the HDL cosimulation component. You must determine how long the HDL simulator should run before its outputs are sent back to the HDL component. This timing information should not be confused with the timing used in other SystemVue components.

From the HDL simulator engine's point of view, the SystemVue input interface is viewed as forcing values onto the ports. At the output interface of the HDL cosimulation component, the results are converted back into SystemVue format and sent to the other connecting SystemVue components.

Please note that, input ports of the HDL Cosimulation component can only be connected to the output of a fixed point component. Automatic conversion from other type of ports to fixed point is not supported.

You can specify the HDL simulation to run until the HDL simulator has no more events to process by specifying a negative iteration time. Using this method, the outputs are guaranteed to be stable since there are no more events left in the simulator that might change them. This method is less efficient than the fixed positive iteration time method, as the HDL simulator must be monitored to determine when all events have been processed. Also, it will not work for certain HDL models where some designs never run out of events, such as those with internal clock signals.

Important

When using negative iteration time, be careful about signals generated internally by your HDL code to avoid infinite event loop in HDL simulator.

Supported HDL Port Types

HDL cosimulation currently supports various bit and bit-vector type HDL ports; they are all mapped to the SystemVue fixed point data type port.

In the case of Verilog HDL, which supports only bit and bit-vector type ports, HDL cosimulation will support any type of Verilog port. In the case of VHDL, which has a large set of data types, HDL cosimulation will only support the ports that are of bit, bit-vector, std_logic, and std_logic_vector types described in the IEEE std_logic_1164 library.

Bidirectional HDL Ports

For a bidirectional VHDL port, two ports are created on the cosimulation model. One port is an input port named <VHDL portname>In, while the other port is an output port named <VHDL portname>Out. The input data on the inout type port is applied by SystemVue for the first half of the HDL iteration time; the signal value is then changed to a tri-state condition. You can drive the output data on an inout type port only during the second half of the HDL iteration time, when the value has been changed to a tri-state condition by SystemVue. You must set the inout port to a tri-state condition during the first half of the HDL iteration time period, so that SystemVue can drive the new input data value on the inout port.

Bidirectional ports are not supported in Verilog cosimulation.

Automatic Clock and Reset Signals

Using HDL Cosimulation it is possible to automatically generate Clock and Reset signals without connecting those to the input of HdlCosim component. This features enable the user to generate one clock cycle per HDLCosim iteration which is not possible by connecting a Clock at the input port of HdlCosim component because of its unirate properties.

To generate an automatic Clock select "YES" for the parameter *AutoGeneratedClock*, another parameter *HdlClockName* will be included set it to the name of the Clock port in your HDL code.

To generate an automatic Clock select "YES" for the parameter *AutoGeneratedReset*, another parameter *HdlResetName* will be included set it to the name of the Reset port in your HDL code.

The autogenerated Clock is of a 50% duty cycle and has a period equal to the HDL iteration time. The positive clock edge occurs at IterationTime/2. The default value for the Reset during the first iteration is a logic low from time 0 to 1/4 times the HDL iteration time, a logic high from time 1/4 to 3/4 times the HDL Iteration time, a logic low for remaining HDL iteration time, and a logic low for iterations after that. This means that Reset signal is high during first rising edge of the Clock signal and low after.

The timing of application of inputs for the HDL code generated for a mixed logic is crucial. For example consider a multiplexer (non-clocked, or in general any combinational logic) followed by a latch (clocked, or any sequential logic). The input multiplexer would be triggered the moment input is applied and would produce results with zero delay. If the following component is a clocked component (for example, sequential logic like a latch), then it will be triggered during the same iteration cycle at the positive edge of the clock. So, in the above example, the multiplexer and the latch will be triggered in the same clock cycle. In the corresponding fixed-point design, the multiplexer followed by a latch would fire the multiplexer in one cycle and the latch in the next, producing a delay of one cycle. The HDL cosimulation results will appear one cycle earlier when compared to the equivalent Fixed point component simulation results.

To match the results, the inputs to the HDL cosimulation block must be delayed until after the positive edge of the clock or IterationTime/2. The inputs will be applied to multiplexer or combinational logic after the positive clock edge. The latch will latch this result only in the next firing of the HDL cosimulation block or the next positive clock edge (the automatic clock has one positive clock edge per firing).

The delay for each input ports is specified in the parameter InputPhases.

Time-Specified Signals in User HDL Code

When HDL code has internal clocks or time-specified signals (for example, wait statements

in VHDL code) the HDL cosimulation may keep running until all the events in the user HDL code are processed. The number of events generated in user HDL code can be infinite (for example, when you have an internal clock).

You can avoid using an internal clock and use the SystemVue *Clock* instead (refer to the section *Automatic Clock and Reset Signals* above). If this is not possible, then infinite event processing can be avoided if you know how long the HDL simulation needs to run to complete the cosimulation, with all of the SystemVue iterations. Different simulators have different mechanisms to break a simulation after a certain simulation time. Here is an example using ModelSim:

1. Use the ModelSim simulator to create a file called *test.do*. An example *test.do* may look like this:

```
run 11000
quit -f
```

Set `CmdArgs="-do test.do"`. This stops the simulation after 11000 nsec. (Refer to CmdArgs on the HdlCosim component block.)
The total run time can be calculated as equal to:

   The number of SystemVue iterations (depends on the Data Flow Analysis setup and the different sinks used in the design) multiplied by the IterationTime specified on the HdlCosim block.

Alternatively, you can also open the ModelSim UI mode and use multiple *run 100* commands to see how long it takes before the message *VHDL Cosimulation has completed.*. appears in the ModelSim UI. This time can then be used to create the *test.do* file.

Do not use the *run-all* command, which will process all the events in the HDL simulation.

HDL Cosimulation Component and Parameters

HDL cosimulation component *HdlCosim* is available under *Hardware Design* under the category *HDL Cosimulation*. The *HdlCosim* provides cosimulation with ModelSim ™ SE Plus 6.3g or later. If the user code has not been compiled, HDL cosimulation can compile the user code before cosimulation or use existing compiled HDL code depending on the *HdlSrcFile* and *HdlLibrary* settings.

The component has one multi-input and one multi-output fixed-data type ports.

HdlSrcFile

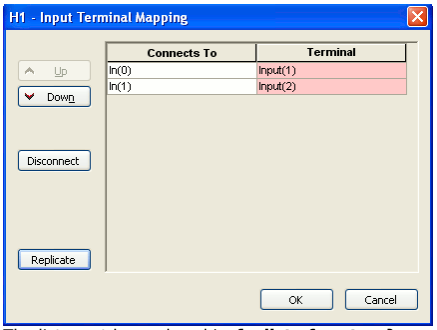
The *HdlSrcFile* parameter could be specified as follows

- If the HDL design is completely contained in one HDL file then Browse to the file. In case of VHDL this file must contain the top level *entity* name specified in *HdlModelName* parameter. In case of Verilog this file must contain the top level *module* name specified in *HdlModelName* parameter. The space character is allowed in the path to HDL files.
- If the HDL design contains multiple files then you must specify all the HDL files. In such case *HdlSrcFile* parameter will point to a file containing list of HDL Files, one per line, in the order in which these needs to be compiled. Each HDL file in every line must be specified using complete path to the file. The space character is allowed in the path to HDL files.

  Warning
  - Do not enclose any HDL file with complete path by double quotes "".
  - Do not include multiple HDL Files in a single line when *HdlSrcFile* points to the list of HDL files.
- The HdlCosim will automatically detects that if *HdlSrcFile* points to an HDL file or a file containing the list of HDL files.

HdlInputs

The *HdlInputs* parameter lists the names of the input ports of the HDL entity/module specified in *HdlModelName* parameter. In case of multiple input ports, all the HDL input port names that need to be updated from SystemVue must be specified in the same order in which these are connected to the multiple-inputs port. To find the order of connection right click on the *HdlCosim* component input port and *Edit Terminal Mapping* to open its *Input Terminal Mapping*, the *Net* connected to ***Input(1)*** should correspond to the first input in the list of *HdlInputs* and so on. Please see the picture below.



The list must be enclosed in **{ <list of ports> }**, each port name in the list must be enclosed by single quotes **'<port Name>'** and separated by space character. For example to specify input ports **Ain**, and **Bin** use the following exact format,

```
{'Ain' 'Bin' }
```

This list is used to make the input connections between the SystemVue ports and the HDL ports. The number of inputs connected to *HdlCosim* component's multiple-inputs port must be equal to the number of strings specified in the *Inputs* string array.

Note

- Even if you have only single input, you must use { } and ' ' to specify it. For example {'A'}
- Please read *Nets, Connection Lines and Buses* (users) to use the Buses to connect multiple inputs to multi-input port

InputPhases

The *InputPhases* parameter delays the application of the input to the HDL model. It is an array of integers. The time unit is the same as specified by the *TimeUnit* parameter, described later. The *InputPhases* parameter specifies the delay for the application inputs during an iteration, as explained in the section Automatic Clock and Reset Signals.

The order of the *InputPhases* specification must be the same as the order of the input names specified in the *HdlInputs* parameter. If no *InputPhases* are specified then all the input values are assigned to the corresponding HDL input ports at the start of each iteration without any delay. For example to delay the input **Ain** and **Bin** by 2ns each, the following exact format could be used

```
[2 2]
```

InputWordLengths

The *InputWordLengths* parameter specifies the number of bits in the corresponding ports specified in VHDL/Verilog description of the entity/module. The order of the *InputWordLengths* specification must be the same as the order of the input names specified in the *Inputs* parameter. This parameter must be specified properly to declare the number of bits in each port specified in *Inputs*. Use 1 for a single bit port. For example, if **Ain** and **Bin** are of 16 and 8 bits respectively, the following exact format could be used

```
[16 8]
```

- If wordlength of input fixed point is larger than what is specified in *InputWordLengths* for that port then only *n* least significant bits are sent to HDL entity/module where *n* is the number of bits specified in *InputWordLengths*.
- If the wordlength of input fixed point is less than what is specified in *InputWordLengths* for that port then remaining most significant bits are appended with 0's.

AutoGenerateClock

Instead of connecting a Clock signal to the input, it is possible to automatically generate a clock signal for your HDL code using HdlCosim. Selecting YES for this parameter will generate automatic Clock in HDL simulator for the HDL port whose name is specified in *HdlClockName* parameter explained next. The generated clock is of 50% duty cycle with period equals to the value of parameter *IterationTime*. The rising edge of the clock will appear in the middle of each iteration cycle. With this feature it is possible to generate a complete clock cycle for each iteration cycle of HdlCosim block, which is not possible by connecting a manual clock signal as one of the input because of unirate nature of HdlCosim.

HdlClockName

Name of the port in HDL code for which automatic clock will be generated.

Warning

HdlClockName must not be same as one of the entries in HdlInputs parameter.

AutoGenerateReset

Instead of connecting a Reset signal to the input, it is possible to automatically generate a reset signal for your HDL code using HdlCosim. Selecting YES for this parameter will generate automatic Reset in HDL simulateor for the HDL port whose name is specified in HdlResetName parameter explained next. The default value for the Reset during the first iteration is a logic low from time 0 to 1/4 times the HDL iteration time, a logic high from time 1/4 to 3/4 times the HDL Iteration time, a logic low for remaining HDL iteration time, and a logic low for iterations after that. This means that Reset signal is high during first rising edge of the Clock signal if AutoGenerateClock is selected and low after.

HdlResetName

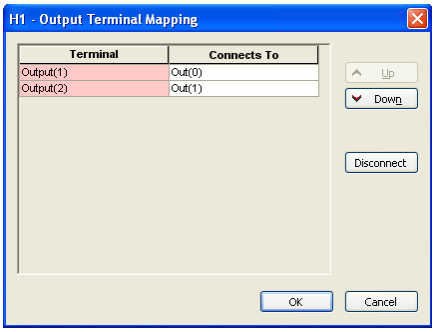
Name of the port in HDL code for which automatic reset will be generated.

Warning

HdlResetName must not be same as one of the entries in HdlInputs parameter.

HdlOutputs

The *HdlOutputs* parameter lists the names of the output ports of the HDL entity/module specified in *HdlModelName* parameter. In case of multiple output ports, all the HDL output port names that need to be updated in SystemVue must be specified in the same order in which these are connected to the multiple-outputs port. To find the order of connection right click on the *HdlCosim* component output port and select *Edit Terminal Mapping* to open its *Output Terminal Mapping*, the *Net* connected to ***Output(1)*** should correspond to the first output in the list of *Outputs* and so on. Please see the picture below:



The list must be enclosed in **{ <list of ports> }**, each port name in the list must be enclosed by single quotes '**<port Name>**' and separated by space character. For example to specify output ports **Aout**, and **Bout** use the following exact format,

```
{'Aout' 'Bout' }
```

This list is used to make the output connections between the SystemVue ports and the HDL ports. The number of outputs connected to *HdlCosim* component's multiple-outputs port must be equal to the number of strings specified in the *Outputs* string array.

Note

- Even if you have only single output, you must use { } and ' ' to specify it. For example {'Z'}
- Please read *Nets, Connection Lines and Buses* (users) to use the Buses to connect multiple outputs to multi-output port

OutputWordLengths

The *OutputWordLengths* parameter specifies the number of bits in the corresponding ports specified in VHDL/Verilog description of the entity/module. The order of the *OutputWordLengths* specification must be the same as the order of the output names specified in the *HdlOutputs* parameter. This parameter must be specified properly to declare the number of bits in each port specified in *HdlOutputs*. Use 1 for a single bit port. For example, if **Aout** and **Bout** are of 16 and 8 bits respectively, the following exact format could be used

```
[16 8]
```

OutputIntegerWordLengths

The *OutputIntegerWordLengths* could be used to format the fixed point precision of the output data to be used with other fixed point components in SystemVue. The SystemVue obtains raw bits from HDL simulator. You could use this parameter to format those raw bits to the precision you want at the corresponding output ports. This is a mandatory parameter and value for each port must be specified. This parameter specified the number of bits on the left of fixed point (integer bits) for each output port. For example, if **Aout** and **Bout** have 8 and 4 integer bits respectively, the following exact format could be used

```
[8 4]
```

AreOutputsSigned

The *AreOutputsSigned* can be used to specify that the output of HDL Cosim should be formatted as either *signed* or *unsigned* fixed point data type to be used with other SystemVue fixed point components. This is a string array type parameter. The **'s'** represents signed and **'u'** represents unsigned fixed point data type.

The list must be enclosed in {}, each entry in the list must be enclosed by single quotes **'<sign of port>'** and separated by space character. For example to specify output port **Aout** to be *signed* port, and **Bout** to be *unsigned* use the following exact format,

```
{'s' 'u' }
```

HdlModelName

HdlModelName is the name of the HDL entity or module to cosimulate with.

For a Verilog module, specify the module name to cosimulate with. For a VHDL entity you can specify this parameter in the following ways:

- To select an entity that has only one architecture, the syntax is **<entity name>** e.g. **myAdder**
- To select an entity along with a particular architecture when more than one is available, the syntax is **<entity name>.<architecture name>** e.g. **myAdder.behavior**
- To select an entity along with its configuration specification, the syntax is **<entity>+<configuration>** e.g. **myAdder+myConfig**.

ShowAdvancedParams

Selecting "YES" for this parameters will show non-mandatory advanced parameters discussed below.

HdlLibrary

The *HdlLibrary* parameter specifies the library from which the compiled HDL module or entity must be loaded. This parameter can control the compilation as follows:

- If the code needs to be compiled, *HdlLibrary* parameter must be empty. This will compile the code under work library. For any subsequent re-simulation of the same design, the HDL code need not be recompiled. To turn off compilation, specify *HdlLibrary*=**work**.
- If you have already compiled the code in another library (for example, *hdllib*) then only the file that has the entity or module specification needs to be specified for *HdlSrcFile*, and *HdlLibrary* should be set to *hdllib*.
  - Before starting SystemVue, the MODELSIM environment variable must be set to a modelsim.ini file that has the mapping information for hdllib.
  - If MODELSIM is not set, you can specify the mapping for the library using =, for example HdlLibrary=**hdllib="c:\user\xyz space\hdllib"**. You can specify more than one library by separating them using spaces, and can specify mappings for any of the libraries using =. If path to the library contains spaces then enclosed that path under double quotes "".

HdlSimulatorGUI

The *HdlSimulatorGUI* parameter determines the user interface mode of the HDL simulator. If the *HdlSimulatorGUI* is ON, the HDL simulator is started with its graphical user interface on. You can view the progress of the simulation, graph signals, and edit values while the simulation is running. You could also debug your HDL design in HDL simulator environment when *HdlSimulatorGUI* is ON.

The ModelSim command Restart is not supported during cosimulation. To restart HDL cosimulation, quit ModelSim and restart the SystemVue simulation.

Note

If the HdlSimulatorGUI is ON and IterationTime is negative, use run -all in ModelSim to perform cosimulation. The other run commands will only increment the HDL simulation time and may not cosimulate properly.

If the HdlSimulatorGUI is OFF, the simulator is run in the background. SystemVue will start the HDL simulator, run the simulation, and close the simulator at the end of simulation without user interaction and without bringing ModelSim GUI.

CmdArgs

The *CmdArgs* parameter specifies special simulator command invocation arguments required for simulation of the HDL model.

IterationTime

The *IterationTime* is the time that the HDL simulation is run during each invocation of the HDL cosimulation component. If the integer value provided is positive, the HDL simulator will simulate for the specified number of time units (where the time units are specified by the parameter *TimeUnit*) then send data to SystemVue. This does not check to see if there are any events still to be processed in the simulator. This feature is useful if you are running a model whose output data is to be sampled periodically at a predetermined time.

Note

The value can never be specified as 0 because the simulation will stop with a range error.

Negative iteration time is valid only for ModelSim VHDL cosimulation. If the value is negative, the HDL simulator is run until all the events are processed. The magnitude of the value specifies the minimum amount of time to run before checking to see if there are any events still to be processed. The output data is read after the event queue becomes empty. This facility can slow down the simulation due to the overhead of monitoring the simulation event queue. The lower the magnitude, the slower the execution because the event queue must be polled more often. This facility is useful when the time the model takes to provide stable/correct data output varies. This will not work for certain models that never run out of events, such as those with internal clock signals.

TimeUnit

The *TimeUnit* parameter specifies the HDL simulation time resolution unit: fs, ps, ns, us, ms or sec.

For VHDL simulation using ModelSim, TimeUnit will control the VHDL simulation time resolution.

For Verilog simulation, TimeUnit will add timescale directives to the top-level cosimulation wrapper. Users can have timescale directives for different modules in their code. If any user module does not have a timescale specified, TimeUnit will be used to generate a default timescale. The smallest of the different timescale specifications will control the Verilog simulation time resolution.

Spectral Propagation and Root Cause Analysis (SPARCA)

A new simulation technique has been created to simulate RF architecture. This technique is called Spectral Propagation and Root Cause Analysis. Every spectrum at each node propagates both forward and backward to every node in the schematic. Along the way noise, intermods, harmonics, and phase noise spectrums are created and propagate to every node in the schematic. These spectrums contain spectral density information so the effects of bandwidth are automatically accounted for. As spectrums propagate through the system spectral genealogy is maintained providing users with the ability to identify the propagation path of every spectrum. Furthermore, this parentage information also includes coherency identification, desired or undesired status, and the frequency equation associated with the given spectrum. For more information see *spectral identification*. This simulation technique is extremely fast compared to traditional non-linear simulation techniques such as harmonic balance that requires convergences criteria and mathematical inversions of large matrices to achieve simulation solutions.

Users specify arbitrary paths through a single block diagram to gather cascaded information along a given path. Each path contains several types of paths such as desired, total, noise, phase noise, etc. Each spectrum along the designated path will be placed in the appropriate path category. Measurements operate on specific path types to create desired effects. For example, the channel noise power measurement excludes all signal, intermods and harmonics, and phase noise spectrums from its path spectrums giving the user only noise within the channel regardless of whether or not a much stronger signal is located at the same frequency. This is a huge advantage allowing the user to see and measure true in-channel signal to noise ratio.

SPARCA Simulation advantages:

- *Fast simulation speed* (sim) - This new technique is much faster than traditional non-linear simulation techniques
- *Identification of every spectrum* - Parentage information is retained for each spectrum and is displayed in graph tooltips
- Signals can be seen underneath other signals - i.e. Harmonics of an amplifier can be seen underneath the noise floor
- True in-channel signal to noise ratio measurements - All spectral components are retained individually and are segregated according to their type giving users a view of desired versus spurious signals even at the same frequency
- Spectral directionality - Spectrums propagate both forward and backward. Some path measurements may only operate in the forward direction of the path
- Bandwidths for all spectrums - All spectrums have bandwidth and spectral density. i.e. A 2nd harmonic with had twice the bandwidth of its fundamental
- *Broadband noise* - Noise is simulated from the lowest to highest frequencies (generally from DC to 5 times the highest signal source frequency)
- *Phase noise* (sim) - Behavioral phase noise is propagated through the system and measurements can operate on just this type of spectrum
- Path VSWR effects - Stage mismatch effects are included in all simulation results
- *Multiple path analysis* for single block diagram - Path analysis is NOT restricted to the traditional 2 port topologies
- Restrictive assumptions from traditional cascaded equations (i.e. *noise* (sim) and *intermods*) are removed - Spectrums are integrated and measurements operate on these spectrum to determine results
- Flexibility for future growth - New spectrum types can be defined and new propagation methods can be added to support changing needs

Getting Started with Spectrasys

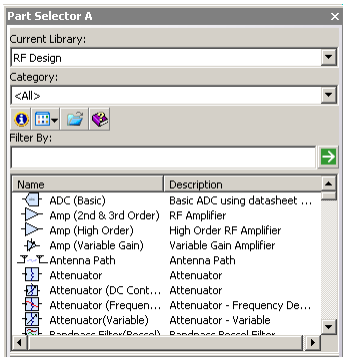
Spectrasys uses a new simulation technique called SPARCA that brings RF architecture design to a whole new level. This walk through will help you design a simple RF chain and measure the architectures noise and gain performance.

The **basic steps** for analyzing an RF system are:

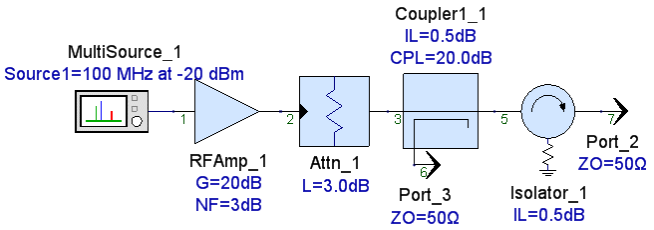
1. Create a System Schematic
2. Adding a System Analysis
3. Run the Simulation
4. Add a Graph or Table

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.



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



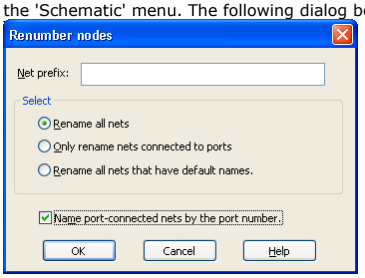
1. Select the '**Amp (2nd & 3rd Order)**' from the system toolbar or part selector.
2. Move the cursor and click inside the schematic window to place the part.
3. Use the prior steps to place a fixed **Attenuator**, **Coupler (Single Dir)**, and **Isolator**.
4. Place a **Source (Multi)** at the input. 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 a **Output Port** on the output of the isolator and the coupler.

> **Hint**
> 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.

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

7. The node numbers seen on your schematic may vary due to the order of the parts placed on the schematic.
   To '**Renumber Nodes..**' select the schematic then select 'Renumber Nodes...' from the 'Schematic' menu. The following dialog box will appear:



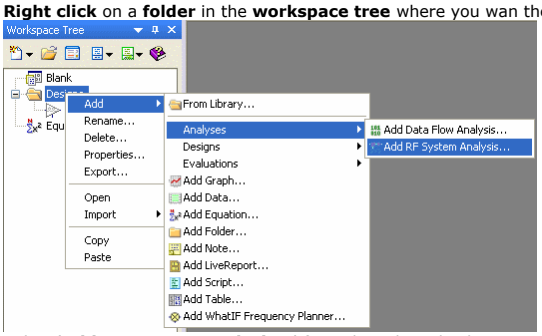
8. Select the desired options and click 'OK'.

Adding a System Analysis

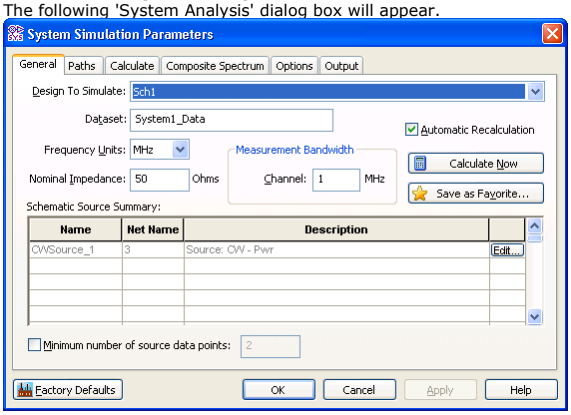
After creating a schematic a system analysis must be created. There a several ways to accomplish this. Only one way will be shown here. For additional information on *adding analyses* (users) click here.

To add a system analysis:

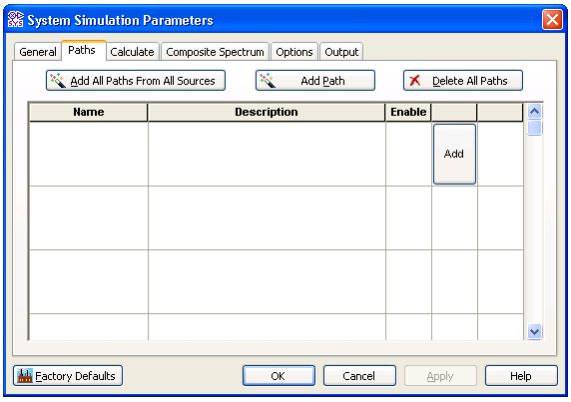
1. **Right click** on a **folder** in the **workspace tree** where you wan the analysis located.



2. Select '**Add RF System Analysis...**' from the selected sub menus as shown above.
3. The following 'System Analysis' dialog box will appear.



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



5. Click the '**Add All Paths From All Sources**' button.

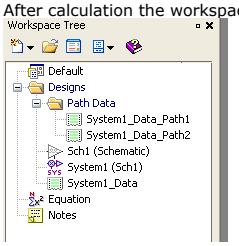
> **Note**
> Node numbers may be different than shown above depending on the node numbers in your schematic. For additional information on *specifying paths* click here.

6. 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 manual calculation is needed the calculate button (!red_calculate_button.gif!) will appear red and so will other items on 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:

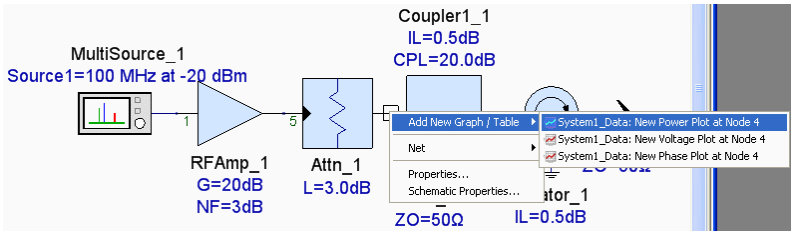


For more information on *datasets* (users) click here.

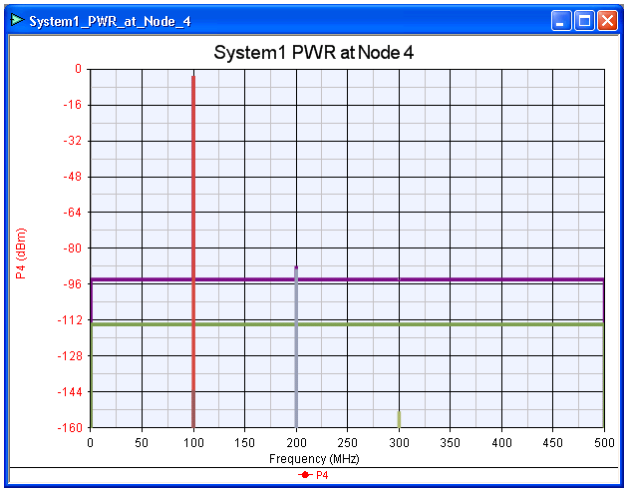
Add a Graph or Table

There are several ways to display data in Genesys. Only one way will be demonstrated here. For additional information on *graphs* (users), click here.

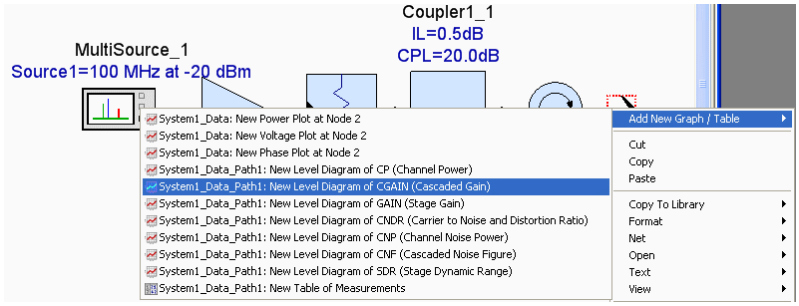
The easiest way to add a **spectral power, phase, or voltage plot** in Spectrasys is by right clicking the node to be viewed and selecting '**System1_Data: New Power Plot at Node x**' from the submenu '**Add New Graph/Table**'. (The output of the attenuator was selected in the following figure)



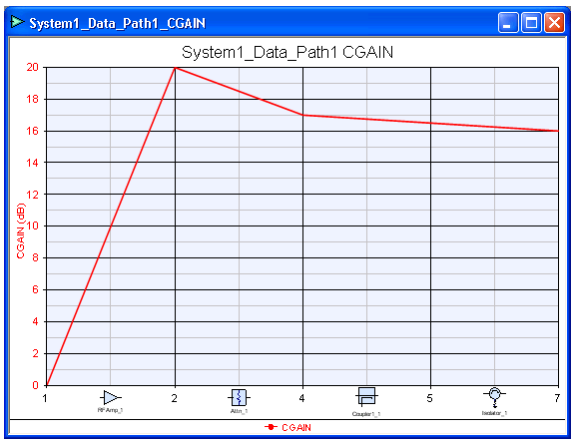
The following graph will appear:



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.

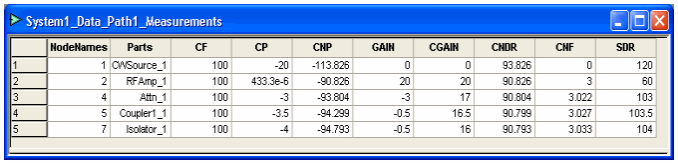


The following level diagram will appear:



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* (sim) information click here.

The default table will look like:



	NodeNames	Parts	CF	CP	CNP	GAIN	CGAIN	CNDR	CNF	SDR
1	1	CWSource_1	100	-20	-113.826	0	0	93.826	0	120
2	2	RFAmp_1	100	433.3e-6	-90.826	20	20	90.826	3	60
3	4	Attn_1	100	-3	-93.804	-3	17	90.804	3.022	103
4	5	Coupler1_1	100	-3.5	-94.299	-0.5	16.5	90.799	3.027	103.5
5	7	Isolator_1	100	-4	-94.793	-0.5	16	90.793	3.033	104

Hint
Right click on the table data to see additional table options.

Dialog Box Reference

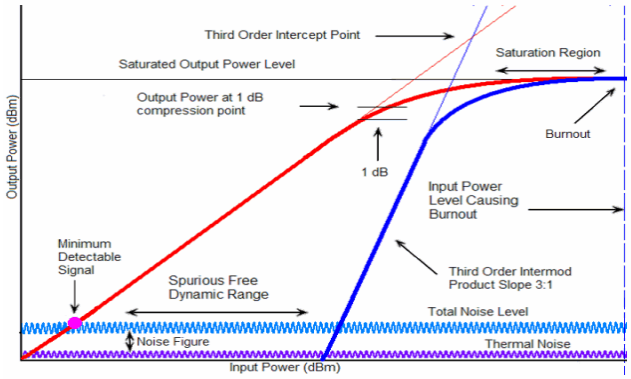
- *General Tab* (sim)
- *Paths Tab* (sim)
- *Add/Edit Path* (sim)
- *Calculate Tab* (sim)
- *Composite Spectrum Tab* (sim)

- *Options Tab* (sim)
- *Output Tab* (sim)

Intermods

Non Linear Model Behavior

In the real world components and stages exhibit non linear distortion such as gain compression and power output saturation. To characterize non linear behavior compression points, saturation, intercept points, and spurious free dynamic ranges are defined according to the following diagram.



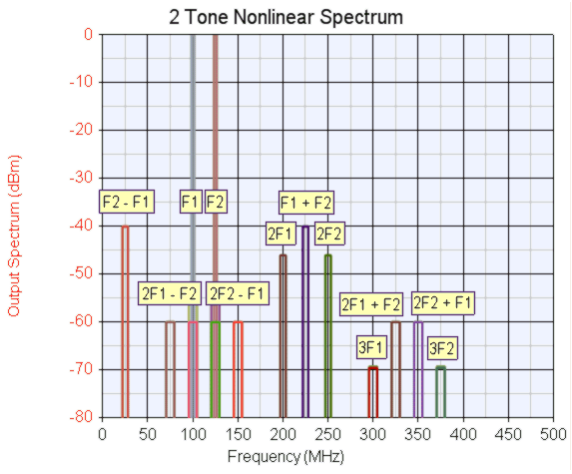
The above figure illustrates 3rd order intermod and noise performance boundaries. As shown a higher intercept point yields larger dynamic range. Consequently, intercept points are commonly used as a performance characteristic of RF systems. In general the higher the intercept point the more tolerant the system is to interference.

In a cascade of RF behavioral models a diagram similar to that above can be derived to determine the overall system performance. The cascaded intercept point is generally referred to the input or output for convenience. Transmitters and amplifiers generally have their intercept points referred to the output and receivers have them referred to their input.

Intermod and Harmonic Basics

This section will help the user understand fundamental relationships between intermods, harmonics, and intercept points. When '*Calculate Intermods* (sim)' and '*Calculate Harmonics* (sim)' are enabled intermods and harmonics will always be created by nonlinear behavioral models. The '*Maximum Order* (sim)' parameter on the 'Calculate' tab of the system analysis determines the maximum intermod and harmonic order used in the simulation.

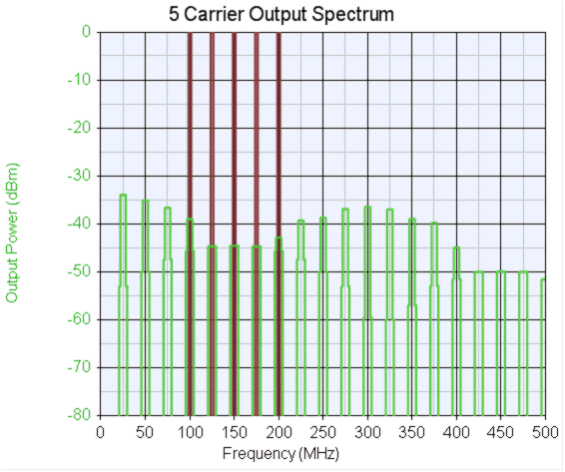
Here is an example of the output spectrum of an amplifier with a two tone input:



The 2 tones are located at 100 and 125 MHz. Notice that the bandwidth of 2nd order products is twice that of the fundamentals and the 3rd order products are 3 times the bandwidth of the fundamentals. The amplifier OIP3 is +30 dBm and the OIP2 is +40 dBm.

Note
The *channel measurement bandwidth* (sim) must be set to at least 3 times the bandwidth of the fundamental tones in order to view the full intermod power of 3rd order products. This bandwidth must be increased accordingly for higher order products.

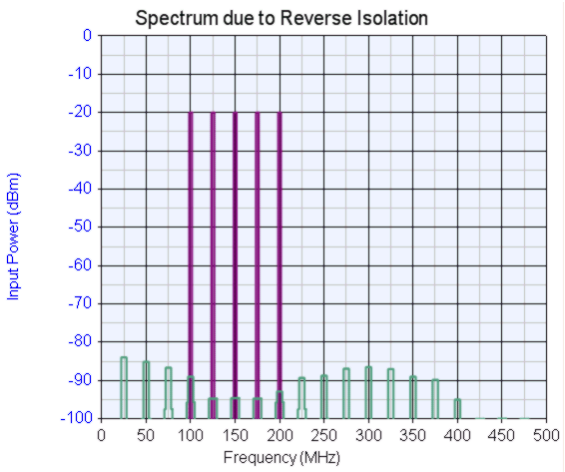
Here is an example of the intermod spectrum due to 5 carriers through the same amplifier. Spectrum groups are being displayed instead of individual spectrums. Signals are shown in one color and all intermods are grouped together and shown in another color.



Notice the peaking effects of the intermods around 300 and 25 MHz as well as the in-channel effects associated with carrier triple beats and 2nd order products.

Reverse Isolation

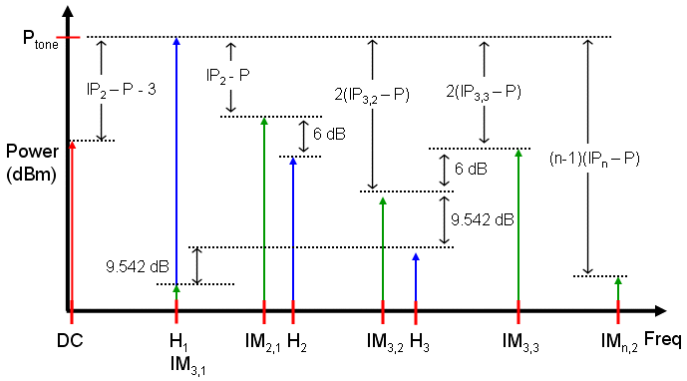
Intermods can and do appear at the input to a nonlinear stage due to the reverse isolation of the device as shown in the following figure:



The 5 carriers are displayed in one color and the reverse intermods in another.

Calculated Products

The following figure shows the nonlinear second and third order products created for two input signals F1 and F2 where F2 is greater in frequency than F1.



The relative levels of spectral components for the small signal regime and equal amplitudes of the signal's tones is shown above.

Definitions of symbols

- **P** - Fundamental Tone Power
- **DC** - DC Value
- **IP $_n$** - Nth Order Intercept Point
- **H $_1$** - Fundamental Tone
- **H $_2$** - 2nd Harmonic

- **H_3** - 3rd Harmonic
- **IM_n** - Nth Order Intermods
- **$IM_{n,m}$** - Nth Order Intermods due to M tones

DC Products

Even order intermods produce DC components. For 2nd order products the DC value is 3 dB higher than the corresponding 2nd harmonic because the DC value is peak power and the 2nd harmonic is average power.

2nd Order Intermod Products

The amplitude of the second order intermod products ($F_2 - F_1$ and $F_1 + F_2$) are equal to the tone power level minus IP2 or in other words $IM_2 = P_{tone}$ - IP2.

2nd Harmonics

The amplitude of the second harmonics are calculated as follows. The amplitude of the second harmonic is equal to the tone power level minus the difference between IP2 (second order intercept) and the tone power level of the device.

3rd Order 2 Tone Products

The amplitude of the third order products ($2F_1 - F_2$, $2F_2 - F_1$, $2F_1 + F_2$, and $2F_2 + F_1$) are equal to 2 times the quantity of the tone power level minus IP3 below the tone level or in other words $IM3 = P_{tone} - 2 (IP3 - P_{tone})$.

Carrier Triple Beats (3rd Order 3 Tone Products)

When more than two carriers are present in a channel, certain 3rd order intermod products are created by the multiplication of three carriers. These intermods are called carrier triple beats. These triple beats have different amplitudes than the more common $2F_1 \pm F_2$ 3rd order intermods. Spectrasys automatically creates triple beats for all combinations of 3 or more carriers. Working out the math, carrier triple beats will be 6 dB higher that the 3rd order 2 tone products. This calculation of the triple beat level assumes that the amplitude of all input signals is the same. The frequency combinations of the carrier triple beats are as follows:

$F_1 - F_2 + F_3$

$F_1 - F_2 - F_3$

$F_1 + F_2 + F_3$

$F_1 + F_2 - F_3$

3rd Harmonics

The amplitude of the third harmonics are 9.542 dB (the non-linear polynomial coefficient of the harmonic is 1/3 that of the 2 tone intermods or 20 log(1/3))below the 3rd order 2 tone products.

Higher Orders

The '*Maximum Order* (sim)' parameter on the 'Calculate' tab of the system analysis determines the maximum intermod and harmonic order used in the simulation. The intermod levels and frequencies are calculated based on a complicated mathematical process. This process description is beyond the scope of this text. Please see other resources for additional information.1

Tone Dissimilar Amplitude

Spectrasys automatically accounts for the amplitude of all input signals that create a given intermod. This yields accurate intermod results as opposed to cascaded intermod equations which ignore the effects of unequal amplitudes.

Number of Intermods

The total number of intermod frequencies is calculated according to the following formula:

Number of Frequencies = (2 x Number of Carriers)^(Harmonic Order + 1)

NOTE: The harmonic order referred to above is the **harmonic order** of each carrier and not the mixing order of the intermod.

The **Maximum Mixing Order** = 2 x Number of Carriers x Harmonic Order.

If the desired mixing order is less than the maximum mixing order then the number of generated intermods is less.

For example, the number of intermod frequencies for 5th order harmonics of each of the 10 carriers is:

(2 x 10)^(5+1) = 20^6 = 6.4e8 intermods

The maximum mixing order would be = 2 x 10 x 5 = 100.

Each intermod will have a unique amplitude many of these intermods may be below the thermal noise floor.

Channel Bandwidth and Intermods

The bandwidth of third order products is greater than the individual bandwidth of the sources that created them. For example, if two tone, each of 1 Hz bandwidth, were used to create intermods, the resulting bandwidth would be 3 Hz. The bandwidth follows the intermod equation that determines the frequency except for the fact that bandwidth cannot be subtracted. For example, if the third order intermod equation is: Fim3 = F1 -

2*F2 then the equation for the resulting bandwidth would be: BWim3 = BW1 + 2*BW2. If BW1 = 30 kHz and BW2 = 1 MHz, then the resulting bandwidth would be 2.03 MHz. The user needs to make sure that the 'Channel Measurement Bandwidth' is set wide enough to integrate all of this energy.

1. Jose Carlos Pedro, Nuno Borges Carvalho, "Intermodulation Distortion in Microwave and Wireless Circuits", Artech House, 2003

Second-Order Intercept Differences for Mixers and Amplifiers

The definition of the IP2 for **mixers** and **amplifiers** may be different. The definition difference is 6 dB. The relationship between these two methods is as shown in the following equation:

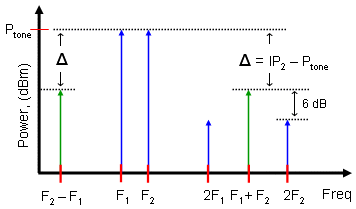
$IP2_h = IP2_i + 6$ dB

Where

$IP2_h$ - Is the second-order intercept due to harmonics

$IP2_i$ - Is the second-order intercept due to intermods

Standard Amplifier IP2 Definition
Amplifier two tone output spectrum is shown in the following graph.



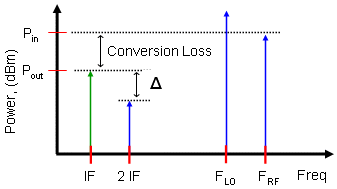
Two tones are generally used to characterize the non-linear response of an amplifier. The difference between the second-order intermod products and second harmonics is 6 dB.

IP2 is defined as:

$IP2 = P_{tone} + Delta_i$

Mixer IP2 Definition
In many cases, second-order products are generally out of band. However, mixer input spurious products exist at sub harmonics of the IF and can be converted directly to the IF through harmonics. Second harmonics are dominant since they have larger amplitudes than third harmonics.

Frequently, the mixer is terminated with a bandpass filter and the second harmonic level is measured by moving the RF input frequency slightly so its second harmonic will fall at the desired IF frequency.



IP2 is defined as:

$IP2 = P_{in} + Delta_h$

Even though the IP2 equations appear to be similar between the amplifiers and mixers the **delta** is actually different. In the amplifier case the reference point is **intermod** amplitude whereas for a mixer it is a **harmonic**. The difference between these two references is 6 dB. The **IF** output frequency can really be thought of as just an intermod ($F_{RF} + F_{LO}$ or $F_{RF} - F_{LO}$).

Note
Generally, IP2 of an **amplifier** is defined at its **output**. However, IP2 of a **mixer** is generally defined at its **input**.

Cascaded Intermod Equations

Caution
Cascaded intermod equations are NOT used by Spectrasys. There are serious drawbacks using these cascaded equations.

Background

Using basic assumptions the intercept point for a cascade can be determined. Cascaded equations come in two flavors, coherent and non-coherent. If the intermods throughout the cascade are assumed to be in phase then coherent addition should be used. This will yield a worst case intercept point. However, if the intermods are assumed to be out of phase then non-coherent addition can be used.

Cascaded Intercept (Coherent Addition)

$$1 / ITOI_{cascade} = 1 / ITOI_1 + 1 / (ITOI_2 / G_1) + \ldots + (G_1 G_2 \ldots G_{n-1} / ITOI_n)$$

Cascaded Intercept (Non-Coherent Addition)

$1 / (ITOI_{cascade})^2 = 1 / (ITOI_1)^2 + 1 / (ITOI_2 / G_1)^2 + ... + (G_1 G_2 ... G_{n-1} / ITOI_n)^2$

Where:

- ITOI - Numeric Stage Input Third Order Intercept
- G - Number Stage Gain
  See McClaning K., and T. Vito (2000). Radio Receiver Design Atlanta, GA: Noble, pp. 605-626 for additional information.

The basic assumptions are:

1. There is no concept of frequency
2. All stages have been perfectly matched
3. No consideration for filtered tones that generate the intermods
4. Multiple paths are ignored (only valid for two port lineups)
5. Gain is assumed to be independent of power level
6. Intermods never travel backwards (reverse isolation is assumed to be infinite)

The calculation of cascaded intermods is generally in a spreadsheet. Note there is no relationship between these calculations and the physical measurements of the intercept points in the lab. There is no mention of frequencies and power levels of tones that are need to make measurements in the lab.

Because of these serious restrictions new intermod measurements were created to eliminate these issues. See '*Intermod Path Measurement Basics*' for additional information.

Intercept Measurements in the Lab

Intercept measurements in the lab are broken down into two groups, in-band and out-of-band. In-band measurements are used when tones are not attenuated by filtering through the cascade. For example, intercept point for a power amplifier is generally done with 2 tones that exhibit the same power throughout the system. Out-of-band measurements are used when they are attenuated like filtering in an Intermediate Frequency (IF).

In-Band Intercept Measurements

For in-band measurements two tones, f1 and f2 are created by two signal generators and combined before entering the Device Under Test (DUT). Care needs to be taken in the setup to ensure reverse intermods will not be generated in the signal generators before appearing at the DUT input. A typical setup is as shown below using two tones.

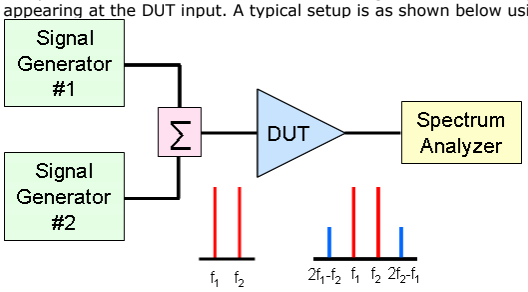


Figure 1 - In-Band Intermod Measurement Setup

The intercept point is determined from the measured power level of the two tones and the power level of the intermods themselves on a spectrum analyzer as shown in the following figure.

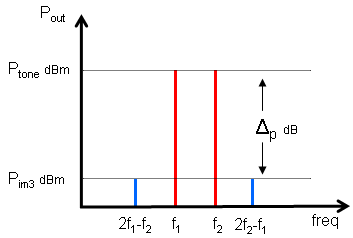


Figure 2 - In-Band DUT Output Spectrum

From this information the output third order intercept is determined as follows:
OTOIdBm = Ptone out, dBm + Δp / 2

The input third order intercept is:
ITOIdBm = OTOIdBm - GainDUT

Out-of-Band Intercept Measurements

Out-of-band measurements are more complicated since the tones have been attenuated at the IF output. This is illustrated in the following two figures.

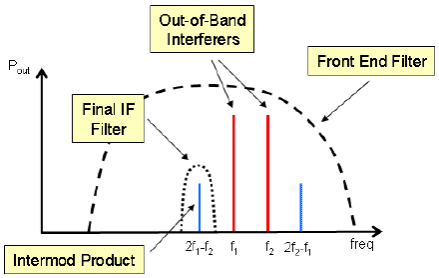


Figure 3 - Third-Order Distortion Inside the Receiver Input Filter

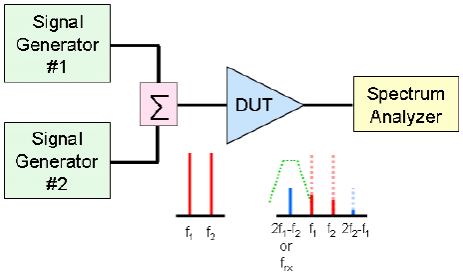


Figure 4 - Out-of-Band Intermod Measurement Setup (Won't Work)

Without knowing what the un-attenuated tone power level is the intercept point cannot be determined as shown in the following diagram.

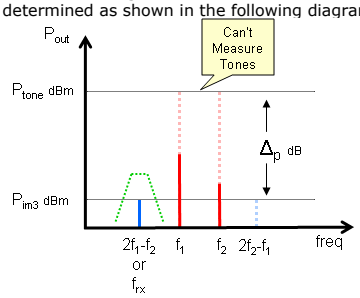


Figure 5 - Out-of-Band DUT Output Spectrum (Won't Work)

To remedy this additional steps are needed to determine an out-of-band intercept point.

To solve this problem a virtual tone can be determined which, is the power of the tone at the DUT input plus the in-channel cascaded gain. Once the virtual tone power and the in-channel gain are determined then both input and output intercept points can be found.

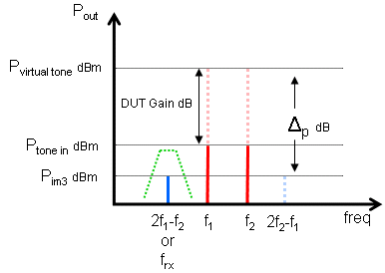


Figure 6 - Out-of-Band DUT Output Spectrum (Virtual Tone)

In the lab measuring the in-channel cascaded gain and intermod power is a two step process. First, the in-channel gain is measured using a single signal generator as shown below.

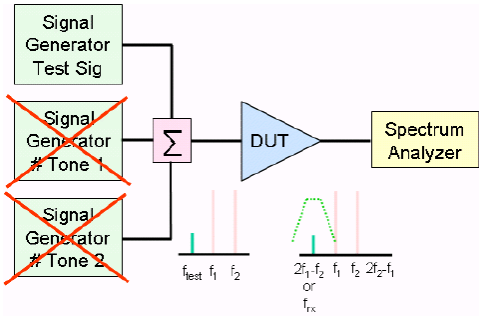


Figure 7 - Out-of-Band Intermod Measurement Setup - Step #1

Next, the in-channel signal generator is disabled and the two tone generators are enabled and the in-channel intermod power is measured as shown below.

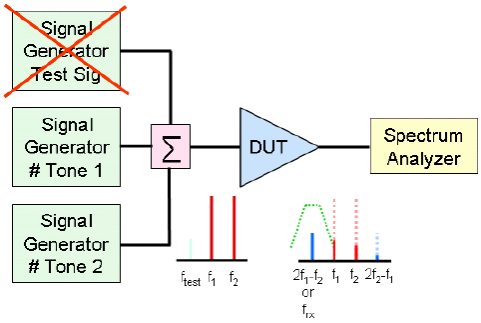


Figure 8 - Out-of-Band Intermod Measurement Setup - Step #2

Be aware that it is very difficult in practice to measure intermod power levels near the noise floor of the receiver. One common technique is to measure the S/N ratio at the receiver output given a known in-channel signal generator input power level. When the in-channel signal generator is disabled and two tones are injected into the receiver the power level of the two tones can be adjusted until the S/N ratio decreases by 3 dB. At this point we know that the power level of the intermod is equivalent to the power level of the on-channel signal generator.

Intercept Measurement Summary

The process of measuring the intercept point is very different than using cascaded equations. When determining the intercept point in the lab the user must physically create both tones at the frequencies of interest. The spacing between the tones needs to be such that the third order intermods will appear at the desired locations. The power level of the input tones needs to be sufficiently low to keep the DUT in linear operation but large enough to be seen in the dynamic range of the spectrum analyzer. When measuring the intermod and tone power levels the user must select the appropriate frequencies on the spectrum analyzer to place the markers. Out-of-band intermod measurements require an additional step to measure the in-channel cascaded gain. This is a very different process than using cascaded intermod equations that remain ignorant of frequencies, impedances, directions, and power levels.

Intercept Points Other Than 3rd Order

Thus far only 3rd order intercept points have been addressed. However, for a two tone source the general intercept equation is:
IPn = (Ptone - Pintermod, n) / (n - 1) + Ptone

Where:

- IPn is the nth order intercept point
- n is the intercept order
- Pintermod, n is the power level of the nth order intermod (created by the two tones)
- All power levels are measured in dBm

This equation only applies for intermods created with two tones. Intermods created with more than two tones have slightly different amplitudes and do not fit the above relationship directly. However, the measurement technique in the lab follows the same process except the frequency of the intermod will changed based on the order of the intermod. For instance, even order products are generally located at twice the frequency of the two tones or close to DC.

Intermod Path Measurement Basics

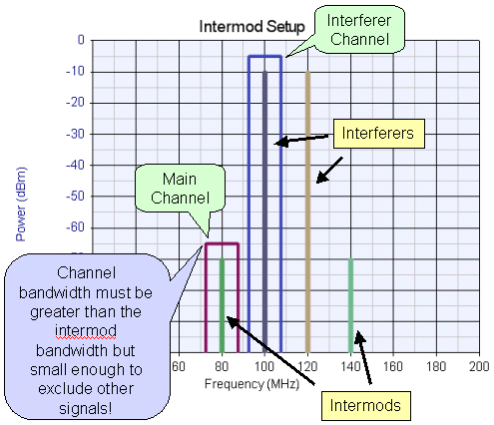
In Spectrasys, intermods are automatically created by all nonlinear behavioral models as long as *intermod and harmonic calculation* (sim) has been enabled.

Measuring intermods in Spectrasys is very similar to measuring intermods in the lab. *Cascaded intermod equations* are NOT used in Spectrasys because of their serious limitations. As such new measurements were created to removed these restrictions.

A behavioral model will always conduct intermods from its input to its output. Additionally, non-linear models will create intermods at the output based on input spectrums.

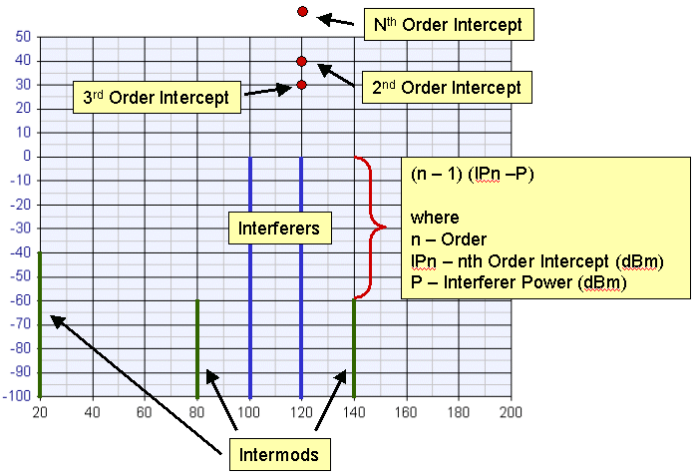
Intermod measurements can show the *generated* (sim), *conducted* (sim), and *total* (sim) intermod channel powers along a path. Furthermore, these measurements can also segregate the data based on intermod order. These channel based measurements should not be confused with a measurement called '*Total Intermod Power* (sim)' which contains the total intermod power of the entire spectrum at the given node and cannot segregate its data based on order.

Here is a simple diagram showing how to setup Spectrasys to make intermod path measurements.



Determining the Intercept Point

Intercept point measurements are assumed to be from two interfering tones. The calculations are based on what would be done in a laboratory as shown in the following figure. As can be seen two measurements are needed to determine the power of the intercept point. These measurements are the power level of the intermod and that of the one of the two tones. If the interfering tones are expected to be attenuated through the system as in a receiver IF a virtual interfering tone must be created by the simulator to correctly determine the intercept point. This is done by injecting a small test signal at the intermod frequency to measure the in-channel gain. Knowing the power level of the two interfering tones to the system plus the in-channel cascaded gain a virtual tone power can be determined at the output of the system which will be used to find the intercept point. In the laboratory this would be done in a two step process since a spectrum analyzer is unable to separate out the cascaded gain test signal and the intermod. However, in Spectrasys this presents no problems and both signal can co-exist at the same time.



Remember
Intermod bandwidth is a function of the governing intermod equation. For example, if the intermod equation is 2F1 - F2 then the intermod bandwidth would be: 2BW1 _ BW2.

Note
Bandwidths never subtract and will always add. The channel bandwidth must be set wide enough to include the entire bandwidth of the intermod to achieve the expected results. For example, CW signals have a 1 Hz bandwidth. Therefore, a third order intermod generated from CW signals will have a 3 Hz bandwidth. If the channel bandwidth is set smaller than 3 Hz not all of the third order intermod energy will appear in the intermod measurements.

Intercept points can only be determined by measuring and interferer signal in an interferer channel. The user must set the main channel frequency in Spectrasys to the frequency where the intermods are to be measured and the interferer channel must be set to the frequency of the interferer to get the correct interferer channel power.

Intercept points can be determined from an in-band or out-of-band method. Both techniques will give identical results in-band. However, if an in-band method is used in a system where the interfering signals are attenuated (like in the IF filter in a receiver) incorrect intercept points will be reported.

Caution
The method used to determine the intercept point is only valid for 2 tones with equal amplitude.

Remember
Intermods travel **BACKWARDS** as well as forward. Backward traveling intermods (like those through reverse isolation paths) will be included in channel measurements and must be considered when making comparisons to cascade intermod equation results. Please consult the specific intermod measurements of interest for details.

Intermod Path Measurement Summary

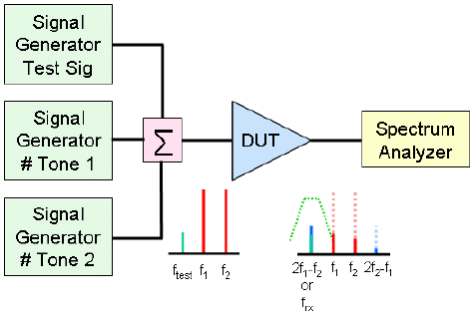
- Add a source that will create an intermod at the path frequency
- Set the '*Path Frequency* (sim)' to the frequency of the intermod.
- Set the '*Channel Measurement Bandwidth* (sim)' to the widest order of interest (not

too wide to include interfering tones).
- Set the '*Path (Interferer) Frequency* (sim)' in the Edit Path dialog box to the frequency of the (interfering) tone.
- Make sure the '*Maximum Order* (sim)' is set high enough to include the intermods of interest.
- Add intermod measurements to a level diagram or table.
- **Remember**: Intermod can and do travel backwards be careful when only expecting forward traveling results.

Cascaded Intermod Equations and Spectrasys

Spectrasys doesn't use cascaded intermod equations and is not restricted to their limitations. Measuring intercept points in Spectrasys is akin to *making measurements in the lab*. Spectrasys needs to know both the frequency of the intermod and the frequency of one of the tones. These values are specified on a path. The path frequency is the frequency of the intermod being measured and the 'Tone (Interferer) Frequency' is the frequency of the tone used to determine the intercept point. Unlike a marker on a spectrum analyzer Spectrasys has the ability to measure the power of a spectrum across a channel. Care must be given to ensure the channel bandwidth is wide enough to cover the bandwidth of the intermod yet narrow enough to exclude the power level of any tones used to create the intermods. Remember, channel measurements use brick wall filtering. Care must be given to set the frequency and bandwidth of the channel to only include the signals of interest. For example, if the user is careless about setting the bandwidth correctly it can be set so wide to include the power of both of the two tones which will yield inaccurate results.

Because Spectrasys has the ability to differentiate between signals and intermods the two step out-of-band process can be done in a single step as shown in the following figure. The out-of-band configuration is the general case and will also work for in-band intercept measurements.



The user can view measurements showing frequencies and power levels of the tones and intermods being used in intercept calculations.

The following tables show a mapping of Spectrasys measurements to the intermod test setup and results:

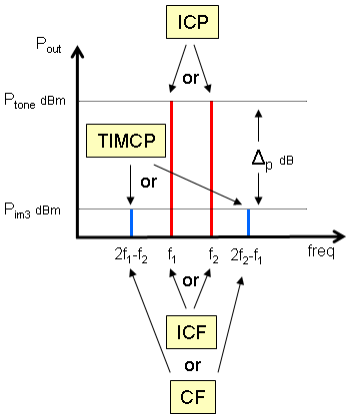
Table 1 - Spectrasys Intercept Setup Measurements

Measurement	Description	Intermod Setup
CF	Channel Frequency	Intermod Frequency
TIMCP (sim)	Total Intermod Channel Power	Intermod Channel Power for each Order up to the Maximum Order
TIMCPn (sim)	Total Intermod Channel Power for Order n	Only Intermod Channel Power for the given Order n
ICF (sim)	Interferer (Tone) Channel Frequency	Tone Frequency used for Intercept Calculations
ICP (sim)	Interferer (Tone) Channel Power	Power Level of the Tone (Used for In-Band Measurements)
VTCP	Virtual Tone Channel Power	Power Level of the Virtual Tone based on the Desired Channel Power of the Test Signal (Used for Out-of-Band Measurements)

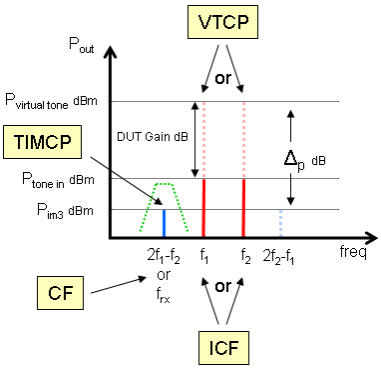
Table 2 - Spectrasys Intercept Results Measurements

Measurement	Description	Intermod Results
In-Band (Uses ICP)		
IIP (sim)	Input Intercept Point	Input Intercept Point for each Order up to the Maximum Order
IIPn (sim)	Input Intercept Point for Order n	Only Input Intercept Point for the given Order n
OIP (sim)	Output Intercept Point	Output Intercept Point for each Order up to the Maximum Order
OIPn (sim)	Output Intercept Point for Order n	Only Output Intercept Point for the given Order n
Out-of-Band (Uses VTCP)		
RX_IIP	Input Intercept Point	Input Intercept Point for each Order up to the Maximum Order
RX_IIPn	Input Intercept Point for Order n	Only Input Intercept Point for the given Order n
RX_OIP (sim)	Output Intercept Point	Output Intercept Point for each Order up to the Maximum Order
RX_OIPn (sim)	Output Intercept Point for Order n	Only Output Intercept Point for the given Order n

Note
In-Band and Out-of-Band intercept measurements will yield the same results if the tones are not attenuated.



In-Band Spectrasys Measurements



Out-of-Band Spectrasys Measurements

From the intermod power level and the measured tone power level intercept points for all orders up to and including the maximum order are calculated. However, only one of them will be valid since the intermod frequency will be different for each order.

In-Band Intercept Simulation

1. Create a two tone source and connect it to the DUT.
2. Create a path and set its frequency to the intermod frequency.
3. Set the "Tone (Interferer) Frequency" of the path to the tones that will be used to calculate the intercept point.
4. Set the Channel Bandwidth to a value wider than the intermod but narrow enough to exclude any power from the tone.
5. Add the IIPn or OIPn measurement to a graph or table when n is the intercept order.

Out-of-Band Intercept Simulation

1. Create a three tone source and connect it to the DUT.
2. Set the frequency of the third test signal to the frequency of the intermod.
3. Create a path and set its frequency to the intermod frequency.
4. Set the 'Tone (Interferer) Frequency' of the path to the tones that will be used to calculate the intercept point.
5. Set the Channel Bandwidth to a value wider than the intermod but narrow enough to exclude any power from the tone.
6. Add the RX_IIPn or RX_OIPn measurement to a graph or table when n is the intercept order.

Troubleshooting Intermod Path Measurements

Here are a couple of key points to remember when troubleshooting and intermod measurement problems. Using a table is generally much better at troubleshooting than level diagrams.

- Look at the 'Channel Frequency (CF)' measurement in a table. This must be the frequency of the intermods of interest.
- Make sure there are intermods within the channel by looking at the 'Total Intermod Channel Power (TIMCP)'.
- If there are no intermods in the channel look at the spectrum and verify that an intermod of the order of interest has been created at the 'Channel Frequency' of the path.
- If there are no intermods at the channel frequency make sure 'Calculate Intermods' has been enabled.
- If there are still no intermods make sure the nonlinear models have their nonlinear parameters set correctly.
- If the 'Total Intermod Channel Power' doesn't seem to be correct verify that the 'Channel Measurement Bandwidth' is wider than the intermod order being measured. i.e. The bandwidth of a third order intermod will 3 times the bandwidth of the two input tones (assuming equal bandwidth tones).
- If the 'Total Intermod Channel Power' still doesn't seem to be correct then verify that the 'Channel Power (CP)' measurement is showing the approximate expected power. The 'Channel Measurement Bandwidth' may be set so wide that other interferer frequencies fall within the channel and the 'Channel Power' measurement will be very high.
- If the 'Total Intermod Channel Power' seems to be too high the intermods may be traveling backwards from a subsequent stage. The reverse isolation of this stage can be increased to verify this effect.

- If the 'Input Intercept Point' measurements (IIP, OIP, RX_IIP, RX_OIP) don't seem to be correct then first verify that the 'Interferer Tone Channel Frequency (ICF)' is set to the interfering frequency.
- If the 'Interferer Channel Frequency' is set correctly then look at the 'Interferer Channel Power (ICP)' measurement for in-band intermod measurements or 'Virtual Tone Channel Power (VTCP)' measurement for out-of-band intermod measurements to verify an expected level of interferer channel power.
- If the 'Output Intercept Point' looks correct but the 'Input Intercept Point' doesn't then verify that the 'Interferer Cascaded Gain (ICGAIN)' measurement is correct for the in-band intermod measurement case or the 'Cascaded Gain (CGAIN)' measurement is correct for the out-of-band intermod measurement case.

Spectrasys Fundamentals

General Behavioral Model Overview

Behavioral models are unique in Spectrasys. Because of the unique simulation technique that is used, Spectral Propagation and Root Cause Analysis (*SPARCA*), behaviors for different types of spectrums can be modeled. SPARCA supports the following types of spectrums:

- Signal - These are spectrums create by signal sources and are generally the desired signals
- *Intermods and Harmonics* - These spectrums are created by signal spectrum and are generally undesired (expect for cases like frequency multipliers and dividers)
- Broadband Noise - These spectrums deal with the thermal noise through the system
- *Phase Noise* (sim) - These spectrums represent the behavioral phase noise through the system

The SPARCA simulation technique is so flexible that additional spectrum types can be added to support future needs.

Each model manages its behavior with respect these spectrum types. Furthermore, SPARCA knows which directions signals are flowing and which signals are desired or not. Every pin on a behavioral models serves both as an input and output pin. Each pin treats its input spectrum with the behavior appropriate to that pin and spectrum type.

Linear analysis uses two port (or N-port) analysis to determine the linear network response. Admittance or Y parameters are the most common types of parameters used for linear analysis. For a two port the current / voltage / Y parameter relationships are: I1 = y11V1 + y12V2 and I2 = y21V1 + y22 V2. These Y parameters can be put together in a matrix form to represent current / voltage relationships of any given model.

Linear models (resistors, capacitors, transmission lines, etc) use a Y-matrix to determine the input to output transfer function for each spectrum. For linear models the same input to output transfer function is applied to all spectrum types.

Non-linear models (amplifier, multipliers, mixers, etc) use a Y-matrix that is dependent on non-linear parameters such as P1dB, PSAT, IP3, and IP2. Non-linear parameters are used to create the behavior associated with the given model, input pin, output pin, and spectrum type.

Since a Y-matrix is used for calculations VSWR effects along a path and at the schematic nodes is automatically accounted for.

Source Models

Sources must be placed in the schematic and connected to the device under test before a system analysis can produce any useful data.

Note
Thermal noise is automatically added to the system analysis.

For specific source information in Part Catalog click the following links:

- MultiSource

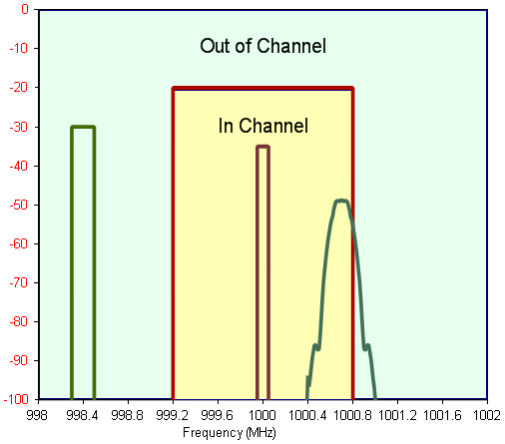
Channels

All measurements in Spectrasys are based on a channel.

Channels consist of:

1. Center Frequency
2. Bandwidth

h4 Channel Example
The channel center frequency is 1000 MHz with a bandwidth of 1.6 MHz (999.2 to 1000.8 MHz). Only the spectrums located in the yellow region will be integrated by the channel measurements.



There are several different channels used in Spectrasys:

1. **Main** Channel of the Path - Most of the measurements are based on this channel. Each new path can have a new center frequency however the bandwidth for all paths will be identical.
2. **Offset** Channel - This is a user defined channel. This channel is specified as an offset

relative to the main channel of the path. See the '*Add / Edit* (sim)' section of the dialog box reference for additional information on specifying this channel.

3. **Interferer** Channel - This is the channel that contains the tone or interferer used to calculate input and output intercept points. See the '*Add / Edit* (sim)' for additional information on specifying this channel.
4. **Adjacent** Channel - This channel is adjacent to the main channel of the path. It is provided as a convenience to the user.
5. 1st Mixer **Image** Channel - This channel is used to make image measurements of the first mixer of the transmitter or receiver chain. It is this mixer which typically sets up the main interferer because of relaxed gain / bandwidth tolerances at the beginning of the system chain.

Note
The bandwidth for **ALL** channels except the Offset Channel is the '*Channel Measurement Bandwidth* (sim)'.

Note
For measurements that require a channel only spectrum falling within the channel will be integrated. If the path center frequency is set incorrectly or the bandwidth is set too large or small measurement values may be different than expected. Mathematical integration is used and is precise. If a spectrum is split by the channel bandwidth then only that portion of the spectrum that falls within the channel will be measured. Of course, spectrum plots will show all spectrum regardless of whether they are in the channel or not.

Caution
When the Channel Frequency is less than 1/2 the Channel Bandwidth the lowest integration frequency used for measurements will be 0 Hz. This will result in Channel Noise Power measurements being different than when the full bandwidth is used.

Specifying Paths

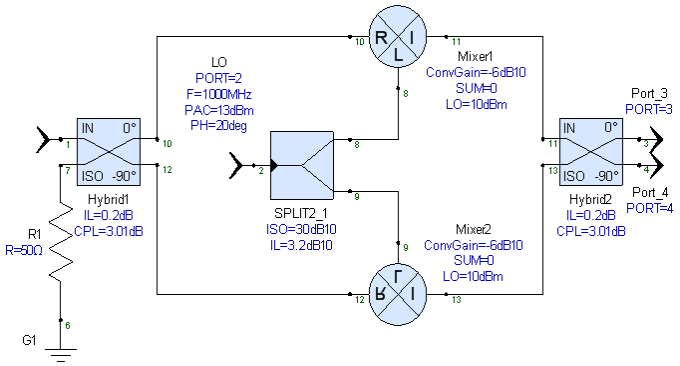
Spectrasys supports multiple paths through arbitrary architectures. Paths are not restricted to traditional 2 port cascaded lineups.

A path consists of:

1. Name
2. Beginning Node
3. Ending Node
4. Frequency

Spectrasys will find the shortest path between the specified nodes. If the user would like to select an alternate path then one or more 'thru nodes' can be added to the path to uniquely identify it. Through nodes can added until the path is uniquely identified.

For example,



The path from node 1 to node 3 could either be through Mixer1 or Mixer2. Since both paths are the same length the one Spectrasys would use would be the first one that is found. However if we wanted the path to go through Mixer1 then we could insert either node 10 or 11 into our path from 1 to 3. The path would then be forced to go through Mixer1.

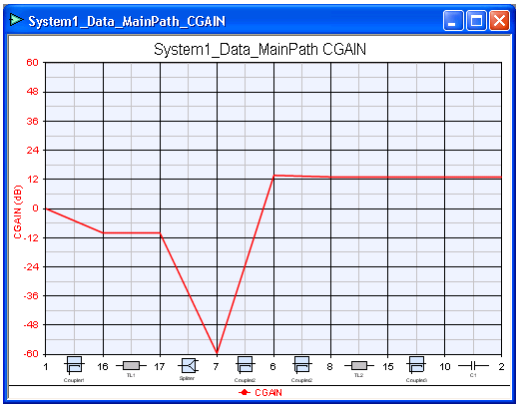
See paths for path dialog box information.

Level Diagrams

Level diagrams give the user a quick visual indication of the performance of the entire cascade. A level diagram can display measurements of cascaded stages along a user defined path. Each horizontal division of the X axis of the graph represents a stage along the path. The first division represents the input to the cascade and the last division represents the output of the cascade. Each vertical division is at the interface between the two stages. The value of the measurements are displayed on the vertical axis.

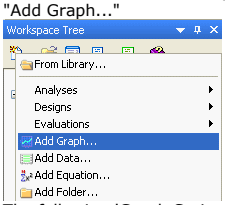
Node numbers are placed on the horizontal axis to show the node sequence of the path. Furthermore, schematic symbols are extracted from the schematic and placed at the bottom of the level diagram.

Sample Level Diagram:

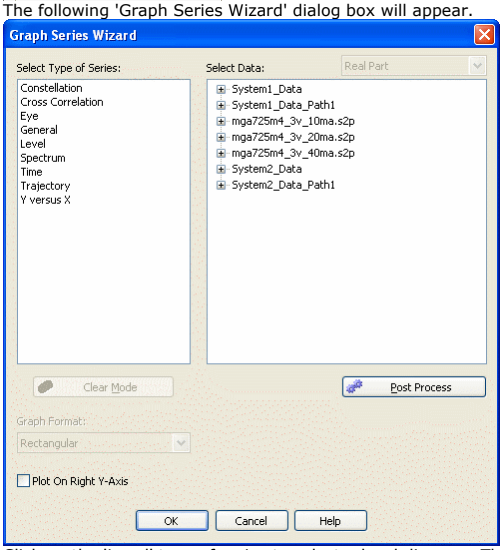


Adding a Level Diagram

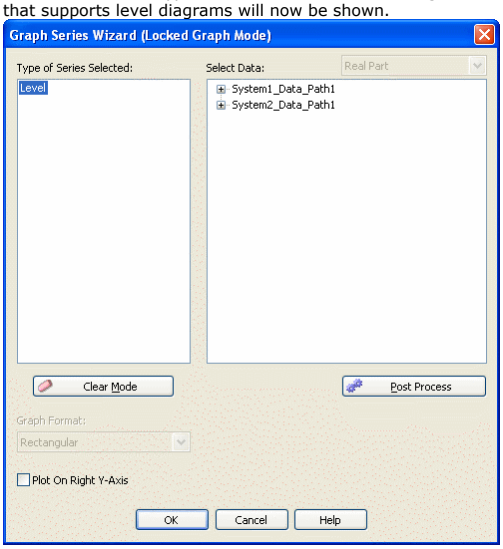
- Instructions on adding *common level diagrams*.
- Manually Adding a Level Diagram:
  1. Click the New Item button ( ) on the Workspace Tree toolbar and select "Add Graph..."



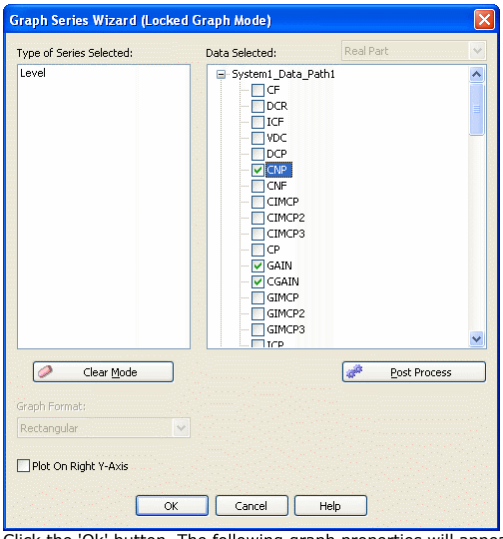
  2. The following 'Graph Series Wizard' dialog box will appear.



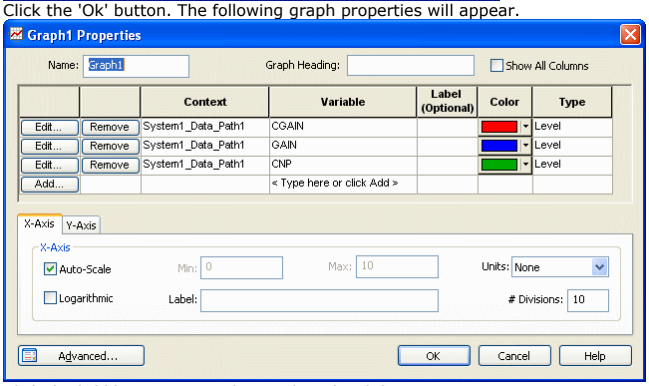
  3. Click on the 'Level' type of series to select a level diagram. The available data that supports level diagrams will now be shown.



  4. Select all desired measurements from the path data set.



5. Click the 'Ok' button. The following graph properties will appear.



6. Click the 'Ok' button to see the resulting level diagram.

For additional information, see *graphs* (users).

Getting the most out of a Level Diagram

- Use the **right Y axis** to examine additional path measurements.
- Change stage parameters directly from the level diagram by **double clicking the part** at the bottom of the level diagram.
- Use the mouse wheel to **zoom** in an out when the path contains many stages.
- The X axis range can be set manually by the user.

Note
Indexes are used in this case NOT node numbers. Index '0' is the first node along the path.

Hint
Use tables not level diagrams when troubleshooting problems. More parameters can be examined at the same time with tables than level diagrams. Checking channel frequencies and power levels are very important during the troubleshooting process.

Spectrum Plots and Tables

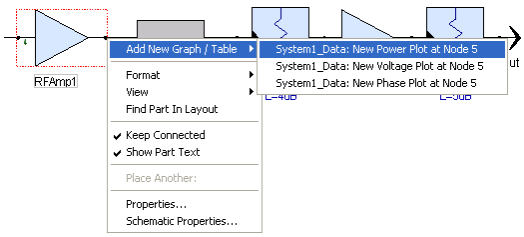
Spectrum plots in Spectrasys are unique because of the type of information displayed. They can display:

- Individual pieces of spectrum including signals, intermods and harmonics, thermal noise, and phase noise
- The total spectrum comprised of all individual pieces of spectrum in every direction through the node
- Spectrum analyzer trace for each total spectrum
- Spectrum like signals, intermods and harmonics, thermal noise, and phase noise can be grouped and shown instead of individual spectrums

For additional information see controlling what types of spectrums displayed.

Easiest Way to Add a Spectrum Plot

The easiest way to add a spectrum plot is to right click the node of interest then select the desired plot from the 'Add New Graph / Table ' submenu as shown below.

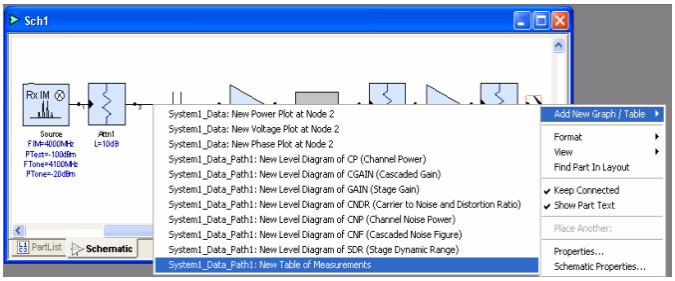


The spectrum plot will then appear:
Unable to render embedded object: File (added_spectrum_plot.gif) not found.

Easiest Way to Add a Table

The easiest way to add a path table is to right click THE NODE WHERE THE PATH ENDS then select the 'System1_Data_Path1: New Table of Measurements' from the 'Add New Graph / Table ' submenu as shown below.

Note
The path name may be different because this is based on the dataset and path name.



The following default table will appear:

System1_Data_Path1_Measurements

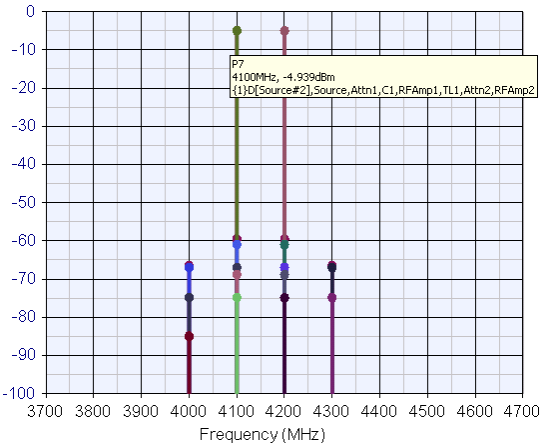
	NodeNames	Parts	CF	CP	CNP	GAIN	CGAIN	CNDR	CNF	SDR
1	1	Source	4000	-99.995	-133.826	0	0	29.831	0	116.989
2	3	Attn1	4000	-109.727	-133.833	-10.007	-10.007	11.779	10	126.996
3	4	C1	4000	-109.724	-133.833	-21.85e-9	-10.007	11.727	10	126.996
4	5	RFAmp1	4000	-71.962	-111.83	19.985	9.978	-17.992	12.017	18.01
5	6	TL1	4000	-72.818	-112.724	-0.9	9.078	-18.036	12.024	107.927
6	8	Attn2	4000	-76.768	-116.673	-4	5.078	-18.087	12.074	111.927
7	7	RFAmp2	4000	-66.451	-106.502	9.994	15.072	-18.415	12.252	21.932
8	2	Attn3	4000	-71.451	-111.485	-5	10.072	-18.415	12.269	106.932

Identifying Spectral Origin

Since each spectrum is tracked individually the user can find the origin and path of each spectrum by placing a marker on the graph or placing the mouse cursor over the spectrum of interest. When a graph marker is added to a plot, the marker will attach itself to the closest spectral data point. The mouse flyover text ONLY appears when the mouse is over the spectrum data point or the marker text on the right side of the graph. These spectrum data points can be enabled or disabled.

Note
If you are having a difficult time getting the mouse flyover text, then try enabling the spectrum data points to see the exact data locations for the mouse. The size of the data symbols can also be changed by right clicking on them and selecting the appropriate menu.

Placing the mouse over the data point on a spectrum yields the following:



The format of the spectral identification is as follows:

GENERAL FORMAT

- Line 1 - Measurement Name
- Line 2 - Marker Frequency, Marker Power (Voltage) Level
- Line 3 - {Coherency Number} Signal Type [Frequency Equation], Origin Part, Next Part, ..., Current Part

Coherency Number

All signals in Spectrasys are grouped according to a coherency number. All signals with the same coherency number are coherent with each other. For additional information see *coherency* (sim).

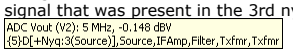
Signal Type

- **D** - Desired Signal. All spectrums are either marked desired or undesired. Desired spectrums are generally those of main interest in the simulation. Desired spectrums consist of signal sources, selected multiplication or division values through frequency multipliers and dividers, and sum or difference products and determined by the user.
- **None** - Undesired Signal. If there is no "D" displayed then the signal is an undesired signal.

Frequency Equation

From the frequency equation the user can identify which source frequencies created the spectrum. This equation is written like a typical mathematical equation. The equation will contain the name and combination of all the sources that created the spectrum. The frequency equation will depend on the model interacting with the signals. The following list of models will show how the frequency equation is modified by them.

Analog to Digital Converter (ADC) - Additional spectrum identification following an analog to digital converter is provided in the frequency equation. A sign in the frequency equation indicates whether the spectrum has been mirrored about the vertical axis or not in the down conversion / aliasing process. The + sign indicates no inversion whereas a - sign indicates inversion. The next piece of information shows the nyquist zone of the originating signal. The values in parenthesis is the frequency equation of the output spectrum. For example, in the following spectrum identification at the output of an analog to digital converter it shows that a 5 MHz signal is not inverted and it is the fundamental signal that was present in the 3rd nyquist zone at the ADC input.



SSB to PM (FREQ_MULT, FREQ_DIV, DIG_DIV) - Additional spectrum identification following an frequency multiplier, frequency divider, or a digital divider is provided in the frequency equation. When more than one signal appears at the input to a frequency multiplication device additional spectrum will appear around the harmonics at the output. These new spectrums are the converted phase modulated spectrums created from the single sided and main input spectrums. The identification is SSBtoPM('Harmonic' : 'Offset'), where 'Harmonic' is the identification of the output harmonic and 'Offset' is the identification of the SSB input spectrum. For example, the following spectrum identification shows that the given spectrum appears around the 2nd harmonic of the LO. The LO is the main input as well as another spectrum coming from a source labeled 'Sneak' which is the SSB spectrum. The identification shows that the offset spectrum appearing around the 2nd harmonic was created from the 'Sneak' input spectrum. Furthermore, the negative sign shows that the offset spectrum frequency is on the low side of the 2nd harmonic.

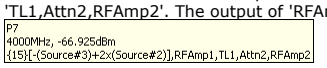
Unable to render embedded object: File (SSB to PM ID.gif) not found.

Path

The path of the spectral component can be determined by examining the comma delimited sequence of reference designators which identify the part where the spectrum was created and the part sequence that the signal took to arrive at the destination node. The first reference designator after the closing frequency equation bracket shows the reference designator where the spectrum was created. The subsequent reference designators indicate the path that the spectral component took to arrive at the node under investigation.

Example

This example shows a spectrum at 4000 MHz whose power level is about -67 dBm. It has a coherency ID of 15 and is a 3rd order intermod between 'Source#2' and 'Source#3'. The intermod was created in 'RFAmp1' and then followed the path through 'TL1,Attn2,RFAmp2'. The output of 'RFAmp2' is where the spectrum is being viewed.



Note
Spectrum identification information can only be displayed if '*Show Individual Spectrums* (sim)' has been enabled.

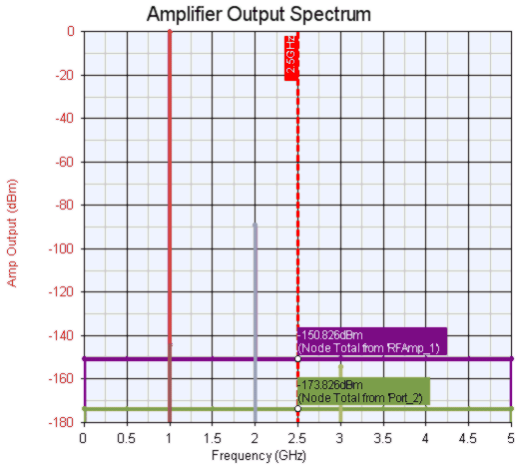
Broadband Noise

The *SPARCA* simulation technique enables Spectrasys to simulate broadband noise very quickly. The '*Ignore Frequency* (sim)' limits are used to specify the frequency of the broadband noise along with the frequency range of the simulator. The entire broadband noise spectrum is simulated with a small number of simulation points. To ensure accurate noise measurements Spectrasys uses a special technique called smart noise point insertion to guarantee that noise data is taken at desired spectrum frequencies. This allows the simulation to run much faster and reduce the number of noise data needed to make accurate noise measurements.

Broadband noise flows in all directions through a node. For example, if the output port was being examined then on a spectral plot the user would see the noise power flowing from the device driving the output port. This noise is obviously flowing toward the output port. The output port itself will also generate noise which will flow from the output port back towards the input. Impedances that these spectrums see may be different for every direction through the node. For this reason each of these total noise spectrum are displayed on a spectral plot.

Spectrasys uses complicated noise correlations matrices along with other special noise simulation techniques to be able to be able to propagate noise spectrums especially through multiport devices. The individual noise spectrums, by default, are not shown on spectral plots. However, the user can view this information if so desired. There are times, when debugging noise problems in an RF architecture that this information is extremely helpful.

The following figure illustrates multi directional noise between an amplifier and output port. The amplifier has a gain of 20 dB and a noise figure of 3 dB. The measurement bandwidth is 1 Hz.



As can be seen from the figure the noise from the output port is thermal noise whereas the noise power from the amplifier output is thermal noise plus the amplifier gain of 20 dB plus an additional 3 dB for the amplifier noise figure.

For additional information see *noise analysis* (sim).

It is not the purpose of this documentation to elaborate on noise correlation matrices and other noise simulation techniques. For additional information on noise correlation matrices see, "Computer-Aided Noise Analysis of Linear Multiport Networks of Arbitrary Topology", Vittorio Rizzoli and Alessandro Lipparini, IEEE Transactions on Microwave Theory and Techniques, Vol. MTT-33, No. 12, December 1985.

Two Port Amplifier Noise Analysis

Cascaded noise figure equations assume matched impedance's on the amplifier. Often this is not the case especially when the amplifier is proceeded with a filter. The noise figure of an amplifier is very dependent on its source impedance as explained below.

The noise figure of a two-port amplifier is given by:
$F = F_{min} + (r_n / g_s) * (| Y_s - Y_o |^2)$
where r_n is the equivalent normalized noise resistance of the two-port (i.e., $r_n = R_n / Z_o$), $Y_s = g_s + j * b_s$ represents the source admittance, and $Y_o = g_o + j * b_o$ represents that source admittance which results in the minimum noise figure, F_{min}).

Y_s and Y_o can be expressed in terms of the reflection coefficients Γ_s and Γ_o as $Y_s = (1 - \Gamma_s) / (1 + \Gamma_s)$ and $Y_o = (1 - \Gamma_o) / (1 + \Gamma_o)$

For more information, see G Gonzalez, Microwave Transistor Amplifiers, pgs 142-142, Prentice-Hall Inc, New Jersey, 1984

What this means in layman's terms is that the noise figure of an active device is very much a function of the source admittance or impedance.

Note
Cascaded noise figure equations make the assumption that noise figure of an active device is independent of the source impedance, which can clearly lead to erroneous results.

Propagation Basics

The basic operation of Spectrasys involves the propagation of individual source spectra and all of their derived products (intermods, harmonics, etc.) to every node in the system. These spectrums will keep propagating until no additional spectrums are created. For instance, any new inputs arriving at the input of an amplifier will cause intermods and harmonics to be created at the amplifier output at that particular time. If additional signals arrive at the amplifier input at a future time then new intermods, harmonics, and other spurious products will be created at the amplifier output. This process continues until no additional spectrums are created. If loops exist in the system, then the output from one part will feed the input of the next part and spectrum propagation could continue forever unless special features are placed within the software to limit spectral creation in this infinite loop. Spectrasys has special features to control loops and limit the total number of created spectrums.

Loops

Parts in parallel (parallel amplifiers connected via a 2 way splitter at the input and combined back together with a 2 way combiner at the output) can cause spectrums to be created that will propagate around this parallel path (or loop). If the gain of the amplifier is greater than its reverse isolation the spectrums will keep on growing as they travel around the path and will never die out (we would have an oscillator). The key point here is that if there are loops in the system schematic then it is very important to make sure that the part parameters are entered correctly so that signals don't grow in amplitude as they traverse around a loop. Once loop spectrums fall below the '*Ignore Spectrum Level Below* (sim)' threshold the spectrum will stop propagating around the loop.

Frequency Ranges

Since SPARCA is a continuous frequency simulation technique there is no upper frequency limit. As such, unnecessary simulation time and data may be taken on spectrums adding no value to the solution of interest. Two parameters are used to control which frequencies

will be propagated through the simulation engine. These are '*Ignore Spectrum Frequency Below* (sim)' and '*Ignore Spectrum Frequency Above*' (sim). By default the lowest frequency limit is set to 0 Hz and the upper frequency limit is set to 5 times the highest source frequency.

Controlling Analysis Data

Spectrasys saves data in 1 or more datasets. There is a main dataset associated with the system analysis that stores all the node spectral data such as frequencies, voltages, powers, and voltages. When paths are defined then a dataset is created for each path. The path dataset contains all measurements for the given path. Powers, voltages, and impedances for the path can also be saved to the dataset.

Spectrasys is a continuous frequency simulator and as such no frequency is outside the bounds of the simulator. However, since users work in frequency bands of interest the simulator can be speeded up by ignoring frequency bands outside a given window. Furthermore, spectral amplitudes can have large dynamic ranges. These dynamic ranges can be restricted to ranges of interest. As a general rule, the more data collected the longer the simulation time is.

Spectrasys supports several different spectrum types. The user can select which types of spectrum to simulate.

Controlling Frequency Ranges

There are 2 parameters that control the simulation frequency range. ALL frequencies outside this window will be ignored by the simulator under any condition. Thermal noise is automatically generated in this frequency range (as long as '*Calculate Noise* (sim)' has been enabled).

- *Ignore Frequency Below* (sim)
- *Ignore Frequency Above* (sim)

Controlling Spectrum Amplitude Ranges

There is 1 parameter used to control the simulation amplitude range. ALL spectrums whose power levels fall below the given amplitude value will be ignored.

- *Ignore Amplitude Below* (sim)

Controlling Spectrum Types

The following spectrum types can be ignored during simulation

- *Intermods* (sim)
- *Harmonics* (sim)
- *Thermal Noise* (sim)
- *Phase Noise* (sim)

Controlling Path Data

Every measurement is dependent on one or more types of spectrum. All measurements whose spectrum types have been enabled are added to the path dataset. Furthermore, path powers, voltages, and impedances can also be added to the dataset.

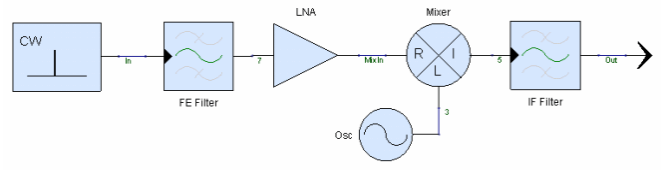
Sweeps of a Path

Sweeps of path measurements can be plotted in one of two ways:

- On a Level Diagram
- On a rectangular graph showing a Swept Variable vs Measurement at a given node

Background

The following schematic will be used for the discussion of this topic:



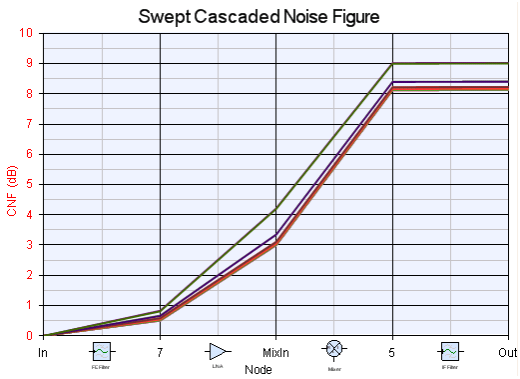
Notice that 3 of the nodes have been manually renamed to help illustrate concepts that will be shown hereafter. To rename a net right click the net then select 'Net' and then 'Rename...' from the sub menu. In this example the source frequency is swept from 2110 to 2170 MHz. An equation is used to synchronize the LO frequency with the input frequency to maintain a constant IF frequency throughout the sweep. For additional information see *parameter sweeps* (users).

When a sweep is performed on a system analysis the spectrum data is swept and a new swept main system dataset will be created. If the system analysis contains paths then each path will be swept and a new swept path dataset will be created for each path.

Swept Level Diagram

To create a swept level diagram do the following:

- Add a rectangular graph
- Set the 'Default Dataset or Equations' to the swept path dataset of interest
- Type in or select the path measurement

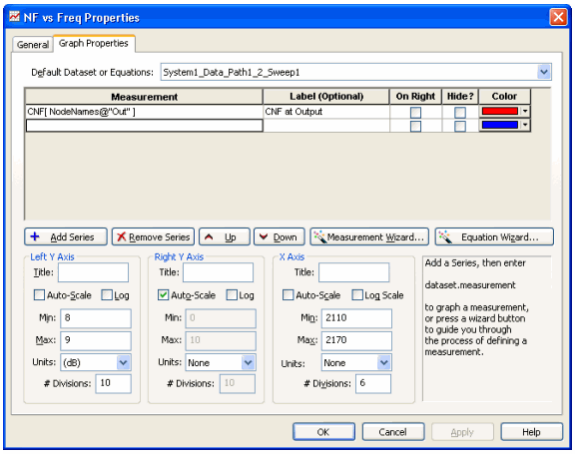


Swept Variable versus Measurement at a given node

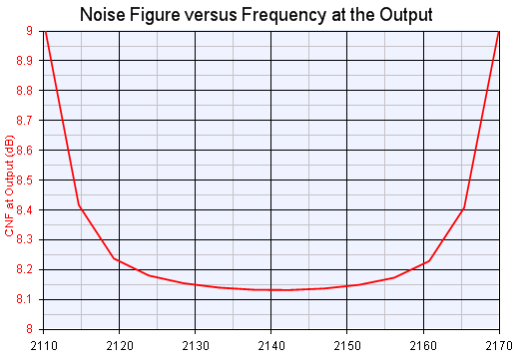
Many times users need a measurement at an input, output, or some other intermediate node. For example, the user may want to examine the cascaded noise figure at the output node versus the input frequency or the channel power at the output node versus a sweep of the input power.

To create a plot of a swept variable versus a measurement do the following:

- Add a rectangular graph
- Set the 'Default Dataset or Equations' to the swept path dataset of interest
- Add the measurement and then selected the array entry in the measurement for the node of interest. To do this use the following syntax:
  - **Measurement**[NodeNames[@"Name of Node"]
  - In this example this would be **CNF**[NodeNames@"Out"] as shown below



- The resulting graph is shown below:



- NodeNames is a variable placed in the path dataset which is the dependent variable for every path measurement. Note that the 'NodeNames' variable is really just an array of strings. The left most column is really just a row number like a spreadsheet or the *i* th entry in the array. For example, NodeNames[3] = "MixIn".

System1_Data_Path1_2

Variable	()	NodeNames
IP1DB=[ip1db(DCP, SDR,	1	In
MDS=[mds(CNP, CNF)]	2	7
MisMatchLoss=1=[imml(C	3	MixIn
NDCP	4	5
NNV	5	Out
NodeNames		
OP1DB=[op1db(DCP, SDF		
PartNo		
Parts		

-
  - The [@|equoperators] operator returns the index in the array of the operand. For example, the index of NodeNames@"MixIn" is 3.
  - In summary then the measurment **CNF**[NodeNames@"Out"] will return all the swept values of **CNF** at the node named "Out".
  - By default, the names of the nodes used in the schematic are numbers so the syntax of **CNF**[NodeNames@"2"] where "2" is the output node is very common.
  - Another way to understand this is to examine the CNF measurement in the swept path dataset. The dependent data is in the right most column titled CNF. This variable has two independent variables 1) the swept frequency and 2) the name of the node. The units of this measurement are show in the header row of the first column. Or in other words dB. The left most column is simply a row or index number. For example, CNF[12] of the swept path dataset would be 0.517 dB which is the 12 entry of the CNF array. The @ operator must be used to extract only the data at the node of interest.

System1_Data_Path1_2_Sweep1

Variable	(dB)	NodeNames	Equation_FRF_Swp_NodeNames	CNF
AN	1	In	2110	0
CCOMP	2	7	2110	0.831
CF	3	MixIn	2110	4.221
CGAIN	4	5	2110	9.023
CGAINV	5	Out	2110	9.032
CNDR	6	In	2120	0
CNF	7	7	2120	0.581
CNP	8	MixIn	2120	3.098
CNR	9	5	2120	8.214
CNRV	10	Out	2120	8.224
CNV	11	In	2130	0
COMP	12	7	2130	0.517
CP	13	MixIn	2130	3.017
CV	14	5	2130	8.139
DCP	15	Out	2130	8.149
DCR	16	In	2140	0
DCV	17	7	2140	0.5
Design	18	MixIn	2140	3
Equation_FRF_Sv	19	5	2140	8.122

Spectral Propagation and Root Cause Analysis (SPARCA)

A new simulation technique has been created to simulate RF architecture. This technique is called Spectral Propagation and Root Cause Analysis. Every spectrum at each node propagates both forward and backward to every node in the schematic. Along the way noise, intermods, harmonics, and phase noise spectrums are created and propagate to every node in the schematic. These spectrums contain spectral density information so the effects of bandwidth are automatically accounted for. As spectrums propagate through the system spectral genealogy is maintained providing users with the ability to identify the propagation path of every spectrum. Furthermore, this parentage information also includes coherency identification, desired or undesired status, and the frequency equation associated with the given spectrum. For more information see *spectral identification*. This simulation technique is extremely fast compared to traditional non-linear simulation techniques such as harmonic balance that requires convergences criteria and mathematical inversions of large matrices to achieve simulation solutions.

Users specify arbitrary paths through a single block diagram to gather cascaded information along a given path. Each path contains several types of paths such as desired, total, noise, phase noise, etc. Each spectrum along the designated path will be placed in the appropriate path category. Measurements operate on specific path types to create desired effects. For example, the channel noise power measurement excludes all signal, intermods and harmonics, and phase noise spectrums from its path spectrums giving the user only noise within the channel regardless of whether or not a much stronger signal is located at the same frequency. This is a huge advantage allowing the user to see and measure true in-channel signal to noise ratio.

SPARCA Simulation advantages:

- *Fast simulation speed* (sim) - This new technique is much faster than traditional non-linear simulation techniques
- *Identification of every spectrum* - Parentage information is retained for each spectrum and is displayed in graph tooltips
- Signals can be seen underneath other signals - i.e. Harmonics of an amplifier can be seen underneath the noise floor
- True in-channel signal to noise ratio measurements - All spectral components are retained individually and are segregated according to their type giving users a view of desired versus spurious signals even at the same frequency
- Spectral directionality - Spectrums propagate both forward and backward. Some path measurements may only operate in the forward direction of the path
- Bandwidths for all spectrums - All spectrums have bandwidth and spectral density. i.e. A 2nd harmonic with had twice the bandwidth of its fundamental
- *Broadband noise* - Noise is simulated from the lowest to highest frequencies (generally from DC to 5 times the highest signal source frequency)
- *Phase noise* (sim) - Behavioral phase noise is propagated through the system and measurements can operate on just this type of spectrum
- Path VSWR effects - Stage mismatch effects are included in all simulation results
- *Multiple path analysis* for single block diagram - Path analysis is NOT restricted to the traditional 2 port topologies
- Restrictive assumptions from traditional cascaded equations (i.e. *noise* (sim) and *intermods*) are removed - Spectrums are integrated and measurements operate on these spectrum to determine results
- Flexibility for future growth - New spectrum types can be defined and new propagation methods can be added to support changing needs

Getting Started with Spectrasys

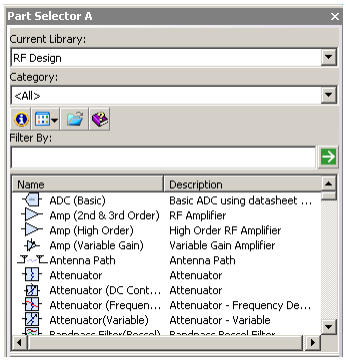
Spectrasys uses a new simulation technique called SPARCA that brings RF architecture design to a whole new level. This walk through will help you design a simple RF chain and measure the architectures noise and gain performance.

The **basic steps** for analyzing an RF system are:

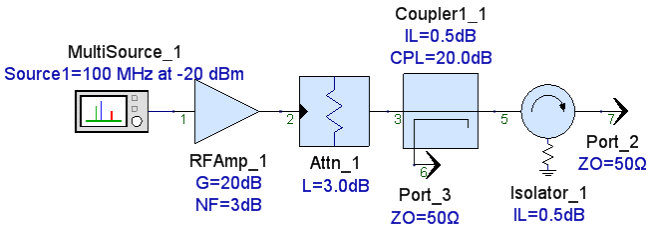
1. Create a System Schematic
2. Adding a System Analysis
3. Run the Simulation
4. Add a Graph or Table

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.



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



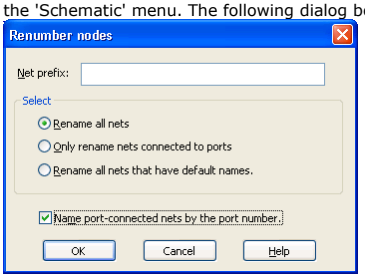
1. Select the '**Amp (2nd & 3rd Order)**' from the system toolbar or part selector.
2. Move the cursor and click inside the schematic window to place the part.
3. Use the prior steps to place a fixed **Attenuator**, **Coupler (Single Dir)**, and **Isolator**.
4. Place a **Source (Multi)** at the input. 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 a **Output Port** on the output of the isolator and the coupler.

> **Hint**
> 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.

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

7. The node numbers seen on your schematic may vary due to the order of the parts placed on the schematic.
   To '**Renumber Nodes..**' select the schematic then select 'Renumber Nodes...' from the 'Schematic' menu. The following dialog box will appear:



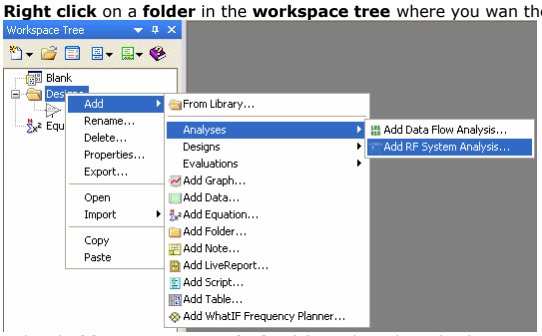
8. Select the desired options and click 'OK'.

Adding a System Analysis

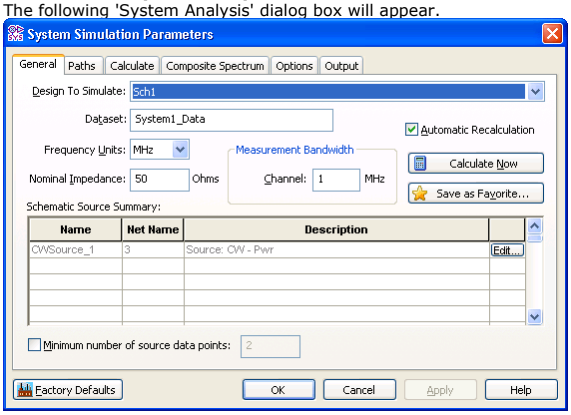
After creating a schematic a system analysis must be created. There a several ways to accomplish this. Only one way will be shown here. For additional information on *adding analyses* (users) click here.

To add a system analysis:

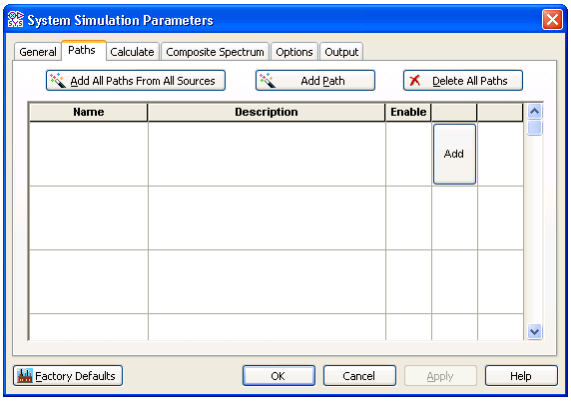
1. **Right click** on a **folder** in the **workspace tree** where you wan the analysis located.



2. Select '**Add RF System Analysis...**' from the selected sub menus as shown above.
3. The following 'System Analysis' dialog box will appear.



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



5. Click the '**Add All Paths From All Sources**' button.

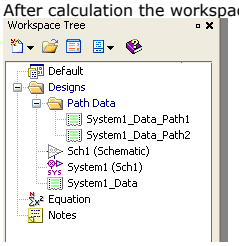
> **Note**
> Node numbers may be different than shown above depending on the node numbers in your schematic. For additional information on *specifying paths* click here.

6. 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 manual calculation is needed the calculate button (!red_calculate_button.gif!) will appear red and so will other items on 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:

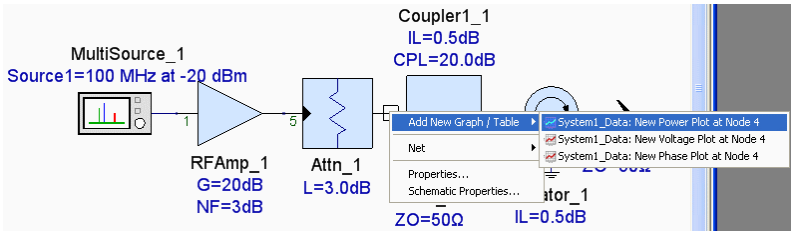


For more information on *datasets* (users) click here.

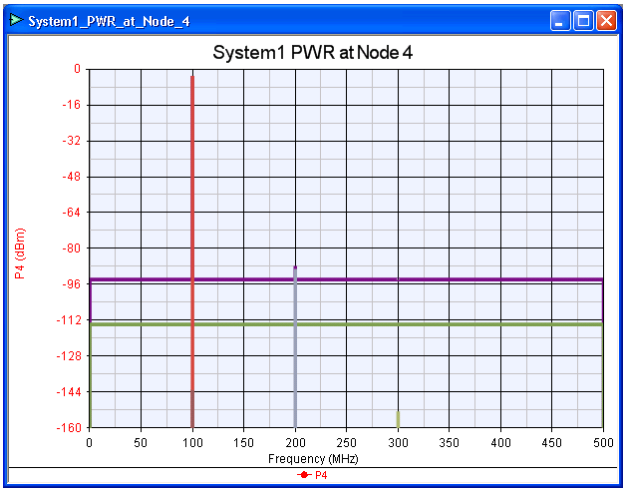
Add a Graph or Table

There are several ways to display data in Genesys. Only one way will be demonstrated here. For additional information on *graphs* (users), click here.

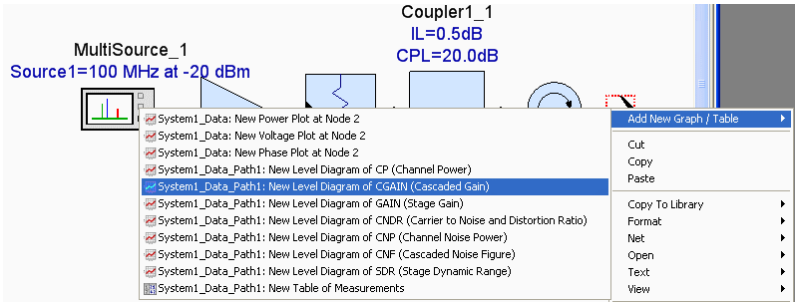
The easiest way to add a **spectral power, phase, or voltage plot** in Spectrasys is by right clicking the node to be viewed and selecting '**System1_Data: New Power Plot at Node x**' from the submenu '**Add New Graph/Table**'. (The output of the attenuator was selected in the following figure)



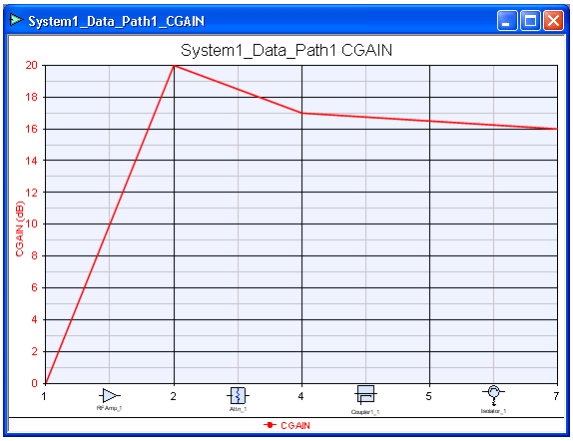
The following graph will appear:



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.

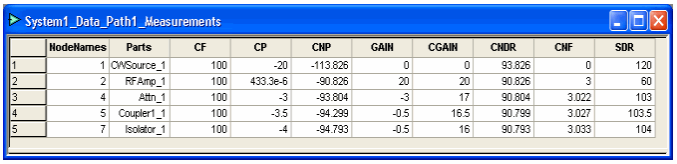


The following level diagram will appear:



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* (sim) information click here.

The default table will look like:



	NodeNames	Parts	CF	CP	CNP	GAIN	CGAIN	CNDR	CNF	SDR
1	1	ONSource_1	100	-20	-113.826	0	0	93.826	0	120
2	2	RFAmp_1	100	433.3e-6	-90.826	20	20	90.826	3	60
3	4	Attn_1	100	-3	-93.804	-3	17	90.804	3.022	103
4	5	Coupler1_1	100	-3.5	-94.299	-0.5	16.5	90.799	3.027	103.5
5	7	Isolator_1	100	-4	-94.793	-0.5	16	90.793	3.033	104

Hint
Right click on the table data to see additional table options.

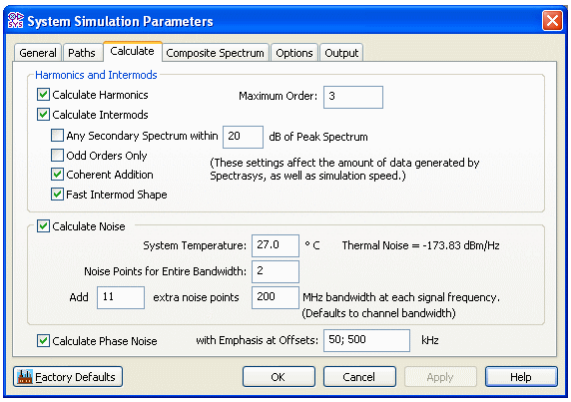
Dialog Box Reference

- *General Tab* (sim)
- *Paths Tab* (sim)
- *Add/Edit Path* (sim)
- *Calculate Tab* (sim)
- *Composite Spectrum Tab* (sim)

- *Options Tab* (sim)
- *Output Tab* (sim)

System Simulation Parameters - Calculate Tab

This page controls calculation of Intermods, Harmonics, Noise, and Phase Noise.



Harmonincs and Intermods

This section controls calculation parameters for harmonics and intermods. See the 'Calculate Intermods, Harmonics' section for more information on intermods and harmonics.

Calculate Harmonics

When checked the system analysis will calculate and save harmonic data to the system analysis dataset. Calculation time for harmonics is typically very quick.

Calculate Intermods

When checked the system analysis will calculate and save intermod data to the system analysis dataset. Intermod simulation time depends on the number of input signals, levels of the resulting intermods, number of non-linear stages, and how tightly coupled loops are. Unchecking this option can improve the simulation speed drastically during troubleshooting of a block diagram.

Any Secondary Spectrum within X dB of Peak Spectrum

When checked harmonics and intermods will only be created from all secondary signals appearing at the input to the non-linear part that are within the specified power range of the peak input spectrum. Secondary spectrum are considered to be any undesired spectrum like intermods and harmonics. This option typically requires longer simulation time since more spectral components are being created. However, there are some cases like the output of mixer where many undesired spectrum exist and only those that fall within the range of the peak will be used to create intermods and harmonics in subsequent stages. In this case the simulation speed may actually be faster.

Odd Order Only

When checked only odd order intermod and harmonics will be created.

Coherent Addition

When checked intermod, harmonics, and mixed signals will be added coherently. Generally, cascaded intermod equations assume coherent intermod addition.

Note: Desired signals will always be added coherently regardless of this setting. See the 'Coherency' section for more information.

Fast Intermod Shape

When checked undesired intermods and harmonics will be represented by only 2 data points. In most cases this is adequate. However, if one desires to examine an undesired intermod or harmonic though a filter then more points may be needed to accurately represent the shape of the signal. When unchecked the average number of points from all input signals is used to represent the undesired intermod or harmonic.

Note: Desired intermods and harmonics like the principle signal coming out of a frequency multiplier or divider will always be represented by the number of points of the input spectrum for a harmonic or the average number of points from all input signals in the case of an intermod.

Maximum Order

This parameter is used to limit the maximum order of the spectrums created in the simulation. This limit applies to all non linear parts. Each model has a limitation on the maximum order that it can generate. Please refer to the part help to determine the order limit for each model.

Calculate Noise

This section controls calculation parameters for thermal noise. When checked noise is calculated. The option must be enabled for path noise measurements. Every component in the schematic will create noise. A complex noise correlation matrix is used to determine the noise power for each part at every node. Unchecking this option can improve the simulation speed drastically during troubleshooting of a block diagram.

System Temperature

This is the global ambient temperature of the entire design under simulation. This is the temperature needed to determine the thermal noise power level. The actual thermal noise is shown for convenience.

Noise Points for Entire Bandwidth

This indicates the number of points used to represent the entire noise spectrum. Noise will automatically be created beginning at the frequency specified by '*Ignore Frequency Below* (sim)' and ending at the frequency specified by '*Ignore Frequency Above* (sim)'. These noise points will be uniformly distributed across this bandwidth.

Note: The more noise points used in the simulation the longer the simulation time generally takes. Since each component generates noise the more components in a schematic will also increase the simulation time. Better speed performance can be achieved for a large number of components by disabling noise calculations or reducing the number of simulation points.

Add Extra Noise Points

This is the number of extra noise points that will be inserted across the 'Extra Points Bandwidth' parameter. These additional noise points will be uniformly distributed across this bandwidth. The center frequency of these noise points is the center frequency of the desired signal frequency. These noise points will be added to every created desired spectrum. However, unused noise points will be removed to improve simulation time. See 'Broadband Noise' and 'Cascaded Noise Analysis' sections for additional information.

Extra Points Bandwidth

This is the bandwidth where additional noise points can be inserted. The center frequency of these noise points is the center frequency of each desired signal. This parameter is used when the user wants greater resolution of the noise like through a narrowband Intermediate Frequency (IF) filter. This bandwidth defaults to the channel bandwidth.

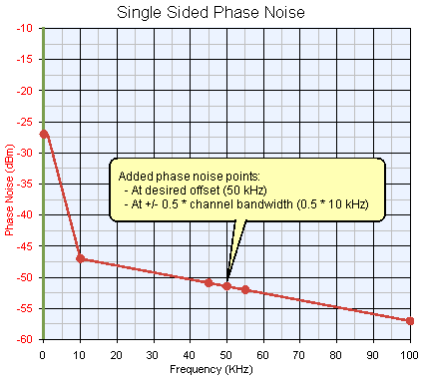
Calculate Phase Noise

When checked behavioral phase noise is calculated. By default the phase noise data points are only created at the user specified frequency offsets. If any measurements are made at specific phase noise offsets other than the user specified points additional phase noise points can be inserted so that phase noise passing through narrow band filters can be measured accurately.

with Emphasis at Offsets

This is a list of offsets where additional phase noise data points will be inserted. Each offset in the list is separated by a semicolon. Though only positive offsets are specified additional phase noise points will be inserted for both sides of the phase noise. At each offset 3 additional noise points will be inserted. One at the specified offset frequency and two others at the channel bandwidth edges. Thus ensuring that 3 phase noise points will appear in the channel of interest. Interpolation of the user entered phase noise is used to determine the amplitude for each inserted offset point. When this field is empty no addition phase noise points are inserted.

For example, if 50 was specified (50 kHz) and the channel bandwidth was 10 kHz the following phase noise points would be inserted: -55, -50, -45, 45, 50, and 55 kHz.



Add / Edit Path Dialog Box

This dialog box is used to specify the characteristics of a path. A basic path consists of a beginning and ending location in a schematic and a channel or path frequency. The path will nominally transverse the least number of stages to arrive at the end. However, there are options to allow users to determine alternate paths through the architecture. A path can be characterized in two ways, **Normal** and a **Quick Sweep**. For a **Normal** path characterization all source schematics will be used and chosen path measurements will be calculated. For a **Quick Sweeep** characterization the source at the beginning of the path will be disabled and the specified path **Channel Frequency** will be used in the sweep creation. Equation based measurements are used for amplitude sweeps to determine the compression point knee to minimize the number of sweep points needed to extract an accurate compression curve.

General Parameter Information

All fields can be numeric or a variable. When using a variable the name of the variable is entered into the field and the variable must be defined in an equations block. For example, if we wanted to use a variable for the offset channel frequency called MyOffset then we would type the string MyOffset into the 'Frequency Offset From Channel' field then define 'MyOffset = 10' in an equation block which is set to use display units or in other words the units used in the dialog box. If the equation units are changed to MKS then MyOffset = 10e6 would represent 10 MHz. To make a variable tunable a '?' is placed in front of the number in the equation block i.e. 'MyOffset = ?10'.

Name

This is the name used to identify the path.

Description

Description of the path. Used for identifying the path to the user. When 'Use auto-description' is checked the description will automatically be filled based on parameters set in the remaining fields of the dialog box.

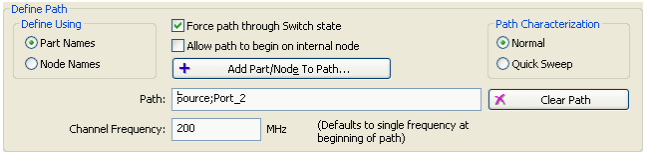
Use auto-description

Description will automatically be created from specified path parameters.

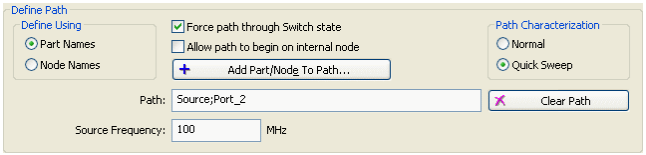
Define Path

This section will define the path to be calculated by the system analysis. A path can be specified with either part names or node names. The path will nominally transverse the least number of stages to arrive at the end. However, there are options to allow users to determine alternate paths through the architecture. In the case where loops or parallel paths exits it may be necessary to specify an intermediate part or node names to force the path through a direction of interest.

(Normal Characterization)



(Quick Sweep Characterization)



Part Names

When selected informs the system analysis that part names are being used for the simulation of the path.

Node Names

When selected informs the system analysis that node names are being used for the simulation of the path.

Force path through Switch state

When checked will force the path to follow the state of the any switches along the path. For example, this allows the path to track the state of a switch bank. When unchecked will allow the user to force the path direction through the switch.

Allow path to begin on internal Node

When checked any desired signals within the channel will be used to determine the desired channel power, gain, and cascaded gain. When unchecked the desired signal at a source must be locatable or the desired channel power, gain, and cascaded gain cannot be determined.

NOTE: When a path begins on an internal node the total power of ALL desired signals at the channel frequency will be used to determine the initial desired channel power and hence the gain and cascaded gain. When unchecked a single signal at the source is used for these measurements and then Spectrasys can distinguish this single desired signal from other desired signals at the same frequency.

Add Part/Node to Path Button

When clicked provides the user with a menu of parts or node names that can be selected in sequence to determine the path. Each click on the menu will add the selected name to the path.

Clear Path Button

When clicked with clear the path list.

Path

This contains a list of part or node names that define the path. These names are specified as an array and can be separated either with semicolons or commas. Spectrasys will find the shortest path containing all the names in the path list. Additional names can be inserted into the path list to force a path in a given direction.

The path can be defined as a string array in an equation block to dynamically change the path based on an equation. For example, a string array for the variable 'MyPath' would be defined:

'MyPath = ["Input Part Name";"...Optional Names";"Output Part Name"]

Note: This example uses part names but node names work equally as well. However the user must make the correct selection of 'Part Names' or 'Node Names'. Also, commas are NOT supported as a valid separator in a string array. A semicolon must be used.

Channel Frequency (Normal Path Characterization)

Defines the frequency at which path measurements are made. The channel measurement bandwidth (on the general tab of the system analysis) will be used in conjunction with the channel frequency to completely specify the channel location in the spectrum and bandwidth of the spectrum integration. If this field is empty the system analysis can determine the path frequency if there is a single signal source on the beginning node of the path. For multiple frequencies the system analysis doesn't know the intent of the user and will display an error. For intermod measurements this is the frequency of the intermod to be measured not the frequency of one of the two tones.

The channel frequency will automatically change along the path through frequency translation devices such as mixer, multipliers, dividers, etc.

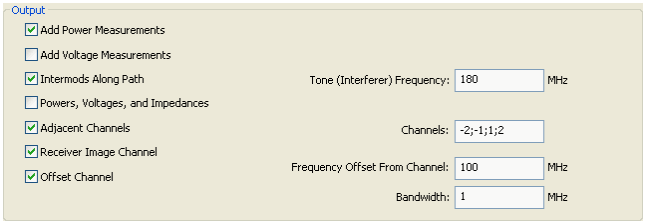
If the user would like to force the channel frequency to a particular value at a given node along the path this can be done by adding an additional frequency to the channel frequency list by separating each frequency by commas. The channel frequency and path list are paired beginning at the start of the path. So the second channel frequency in the list will correspond to the second node or part in the path list. For example, if the user wanted to track the 2nd harmonic created at the output of the LNA and the path is specified as '1;5;2' where node 1 is the input, 5 the LNA output, and 2 the system output. If the input frequency was 1000 MHz then the channel frequency would be specified as '1000;2000'. What this means is that at node 1 the 1000 MHz frequency will be used. When the path calculation reaches the next name in the path list, node 5, the next channel frequency will be used which is 2000 MHz. The channel from this point to the end of the path at node 2 will follows its normal course including through any frequency translation devices. NOTE: When using multiple frequencies node names are a better for path specification since parts generally have more than 1 node and the system analysis doesn't know which terminal of the part to use.

Source Frequency (Quick Sweep Characterization)

Since the a **Quick Sweep** can occur across frequency and amplitude the path frequency at the input is not limited to fixed frequencies that may appear in the schematic. For this reason the user must specify this frequency that is used for amplitude sweeps and will be the center source frequency for frequency sweeps.

Output (Normal Characterization)

This section determines which measurements are added to the dataset and some parameters needed by some measurements. Path measurements can be grouped into the following groups: fundamental, intermod, adjacent channel, receiver image channel, offset channel, and primitive path components of powers, voltages, and impedances. Fundamental measurements like Channel Frequency (CF) are always added to the path dataset.



Add Power Measurements

When checked power based measurements will be placed in the path dataset.

Add Voltage Measurements

When checked voltage based measurements will be placed in the path dataset.

Intermods Along a Path

When checked will calculate intermod measurements and add them to the path dataset. This is not to be confused with intermods created during a simulation. Those will always be created as long as calculation of intermods has been selected on the 'Calculate Tab' of the system analysis.

Tone (Interferer) Frequency

This is the frequency where the tone or interferer is located that will be used to determine the intercept point. In order to determine the intercept point the system analysis must measure the intermod power and the power of the tone or interferer that created the intermod. The path channel frequency is set to the frequency of the intermod and an additional tone (interferer) channel must be created at the frequency of the tone or interferer. The intercept measurement technique used in the system analysis is generally the same as the one used in the laboratory using signal generators and spectrum analyzers.

See Intermod and Harmonic Basics, Intermod Path measurement Basics, Cascaded Intermod Equations, In-Band Intermod Path Measurements, Out-of-Band Intermod Path Measurements, and Troubleshooting Intermod Path Measurements for additional information.

Powers, Voltages, and Impedances

When checked will add path powers, voltages, and impedances to the path. These spectrums are grouped into various categories which are then integrated by corresponding measurements.

Adjacent Channels

When checked will calculate the specified adjacent channels and place the results in the path dataset. The bandwidth of the channel is the channel measurement bandwidth specified on the 'General Tab' of the system analysis.

Channels

These are the adjacent channels which reside on either the lower or upper side of the main channel. These values are specified as an integer array where values are separated by semicolons. Negative numbers represent the channels lower than the main path frequency and positive values represent channels higher than the main path frequency. For example, [-2;-1;1;2] means that first and second lower and first and second upper adjacent channels will be measured.

Receiver Image Channel

When checked all measurements associated with the receiver image channel will be calculated and saved to the path dataset. The receiver image channel is defined as the channel from the input to the first mixer.

Offset Channel

When checked will calculate the offset channel power and frequency measurements and add them to the path dataset.

Frequency Offset from Channel

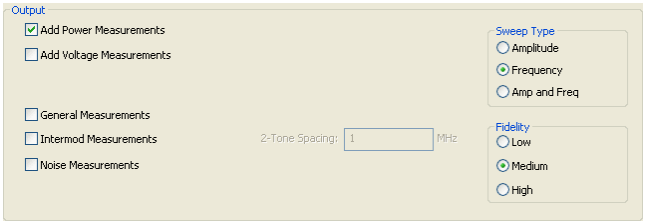
The is the frequency spacing between the main channel center frequency and the center frequency of the offset channel.

Bandwidth

This is the bandwidth of the offset channel power measurement.

Output (Quick Sweep Characterization)

This section determines which measurements are being swept and added to the dataset along with the type and fidelity of the sweep.



The following core measurements will always be added to the dataset:

- Channel Frequency (**CF**)
- Desired Channel Power (**DCP**)
- Desired Channel Phase (**DCPH**)

Add Power Measurements

When checked power based measurements will be placed in the path dataset.

Add Voltage Measurements

When checked voltage based measurements will be placed in the path dataset.

Add General Measurements

When checked will add the following measurements: Desired Channel Resistance (**DCR**), Stage Gain (**SGAIN**), Stage Input Impedance (**SZIN**), Stage Output Impedance (**SZOUT**), Stage Compression (**COMP**), Cascaded Compression (**CCOMP**).

For **Power** based measurements the following measurements will be added: Gain (**GAIN**), Cascaded Gain (**CGAIN**), Channel Power (**CP**), Stage Input P1dB (**SIP1DB**), Stage Input Saturation Power (**SIPSAT**), Stage Output P1dB (**SOP1DB**), Stage Output Saturation Power (**SOPSAT**), Input P1dB (**IP1dB**), Input Saturation Power (**IPSAT**), Output P1dB (**OP1DB**), Output Saturation Power (**OPSAT**), Total Node Power (**TNP**), and Stage Dynamic Range (**SDR**).

For **Voltage** based measurements the following measurements will be added: Channel Voltage (**CV**), Voltage Gain (**GAINV**), and Cascaded Voltage Gain (**CGAINV**).

Add Intermod Measurements

A three tone test is used for intermod measurements which allow the out-of-band-intercept characterization typically required for receivers. See Intermods Section in the Spectrasys simulation manual for additional information. A low level test signal is created at the source frequency used to determine the in-channel gain. Two tones are greated 40 dB above the test signal and are positioned according to the **2-Tone Spacing** value set by the user. For example, if the **Source Frequency** was set to 100 MHz and the **2-Tone Spacing** was set to 10 MHz then three tones would be created: a test tone at 100 MHz and two tones 40 dB larger in amplitude at 110 and 120 MHz. The 2-Tone spacing can be negative so the two tones will appear below the test signal.

When checked will add the following measurements: Interferer Channel Frequency (**ICF**).

For **Power** based measurements the following measurements will be added: Interferer Channel Power (**ICP**), Interferer Gain (**IGAIN**), Interferer Cascaded Gain (**ICGAIN**),

Conducted Intermod Channel Power (**CIMCP**, **CIMCP2**, and **CIMCP3**), Generated Intermod Channel Power (**GIMCP**, **GIMCP2**, and **GIMCP3**), Total Intermod Channel power (**TIMCP**, **TIMCP2**, and **TIMCP3**), Output Intercept Point (**OIP**, **OIP2**, and **OIP3**), Receiver Based Output Intercept Point (**RX_OIP**, **RX_OIP2**, and **RX_OIP3**), Input Intercept Point (**IIP**, **IIP2**, and **IIP3**), Receiver Based Input Intercept Point (**RX_IIP**, **RX_IIP2**, and **RX_IIP3**), Percent Intermods (**PRIM**, **PRIM2**, and **PRIM3**).

For **Voltage** based measurements the following measurements will be added: Interferer Channel Voltage (**ICV**).

Add Noise Measurements

When checked will add the following measurements: Image Channel Frequency (**IMGF**).

For **Power** based measurements the following measurements will be added: Channel Noise Power (**CNP**), Cascaded Noise Figure (**CNF**), Added Noise (**AN**), Carrier to Noise Ratio (**CNR**), Carrier to Noise and Distortion Ratio (**CNDR**), Image Channel Power (**IMGP**), Image Channel Noise Power (**IMGNP**), Image Rejection Noise Ratio (**IMGNR**), Image Rejection Ratio (**IMGR**), Minimum Detectable Signal (**MDS**), Noise and Distortion Channel Power (**NDCP**), Phase Noise Channel Power (**PNCP**), and Percent Noise Figure (**PRNF**).

For **Voltage** based measurements the following measurements will be added: Channel Noise Voltage (**CNV**), Carrier to Noise Voltage Ratio (**CNVR**), Node Noise Voltage (**NNV**), Phase Noise Channel Voltage(**PNCV**), and Total Noise Voltage (**TNV**).

Sweep Type

There are three types of sweeps:

- **Amplitude**
- **Frequency**
- **Amplitude and Frequency**

Amplitude

The amplitude is swept from 100 dB below the estimated cascaded 1 dB compression point (rounded down to the nearest 10 dB) to 20 dB above the estimated cascaded saturation point (round up to the nearest 10 dB). The amplitude sweep is broken up into 4 regions. Each region has its own fidelity. For example, if the cascaded input intercept point was +8 dBm and input saturation point was +11 dBm the amplitude sweep would go from -90 dBm (+8 dBm - 100 dB then rounded down) to +40 dBm (+11 dBm + 20 then rounded up).

Frequency

The frequency is swept across 10 channel bandwidths. The center frequency is the specified **Source Frequency**. The channel bandwidth is specified on the General Tab of the System Analysis dialog box. The frequency sweep is broken down into 3 regions. The optimum power used for this sweep is 50 dB below the estimated 1 dB compression point (rounded down to the nearest 10 dB). For example, if the estimated 1 dB compression point is -4 dBm, the source frequency was set to 500 MHz and the Channel Bandwidth set to 1 MHz then the amplitude would be set to -60 dBm (-4 dBm - 50 then rounded down) and the frequency would be swept from 495 to 505 MHz.

NOTE: The channel bandwidth value specified on the General Tab is used to determine the frequency sweep range. Internally this bandwidth is changed to **10 Hz** during the simulation. All characterization values are taken at 10 Hz.

Amplitude and Frequency

This sweep is a nested sweep that is a combination of a frequency sweep with an amplitude sweep at each frequency point.

Fidelity

The Fidelity is broken up into 3 categories:

- **Low**
- **Medium**
- **High**

The fidelity affects both amplitude and frequency sweeps.

Amplitude Fidelity

The following table shows the power levels of each region of the power in versus power out curve and the power step size in each specified region. Equation based measurements of input P1dB (EIP1DB) and saturation power (EIPSAT) are used as estimates to determine the boundaries between each region.

Fidelity	Region1-2	Region2-3	Region3-4	Step 1	Step 2	Step 3	Step 4
	(dBm)	(dBm)	(dBm)	(dB)	(dB)	(dB)	(dB)
Low	EIP1DB - 10	EIP1dB - 5	EIPSAT + 5	30	5	2	20
Medium	EIP1DB - 15	EIP1dB - 3	EIPSAT + 5	20	4	1	10
High	EIP1DB - 15	EIP1dB - 5	EIPSAT + 5	10	2	1	4

Frequency Fidelity

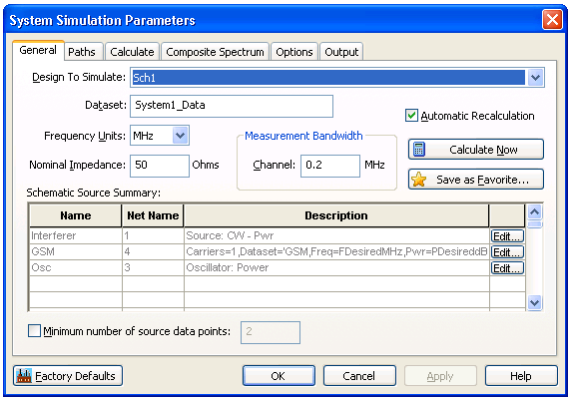
The following table shows each of the frequency sweep regions and the frequency step size resolution within each region.

Region 1 - Region in between 10 and 5 channel bandwidths.
Region 2 - Region in between 5 and 1 channel bandwidths.
Region 3 - Center channel of 1 channel bandwidth.

Fidelity	Step 1	Step 2	Step 3
	(dB)	(dB)	(dB)
Low	Chan BW	Chan BW / 2	Chan BW / 5
Medium	Chan BW	Chan BW / 5	Chan BW / 10
High	Chan BW	Chan BW / 10	Chan BW / 20

System Simulation Parameters - General Tab

This page sets the general settings for a Spectrasys Simulation.



Parameter Information

Design to Simulate

User specified name. This is the source name that will appear in the spectrum identification.

Dataset

Name of the dataset where the simulation data is stored.

Frequency Units

These are the frequency units used for the entire system simulation dialog box.

Nominal Impedance

Default impedance used for power measurements when no impedance information is available during the simulation.

Measurement Bandwidth

Width of the channel used in path measurements. This is analogous to the resolution bandwidth on a spectrum analyzer.

Automatic Recalculation

When checked enables Spectrasys to automatically recalculate the simulation every time the design or system simulation is out of date.

Calculate Now Button

Closes the dialog and initiates a simulation.

Save as Favorite Button

When clicked will save all the parameters associated with the system analysis dialog as the default to be used when the next system analysis is created.

SCHEMATIC SOURCE SUMMARY

This section summarizes all the sources found in the given design. For example, see the schematic source summary in the figure above.

Name

This is the name of the source or part designator.

Net Name

This is the name of the net where the source is connected.

Description

This is a summary description of the source.

Edit Button

When clicked this button will bring up the edit parameters dialog for this part.

Minimum number of source data points

When checked each source signal is represented by the specified number of data points.

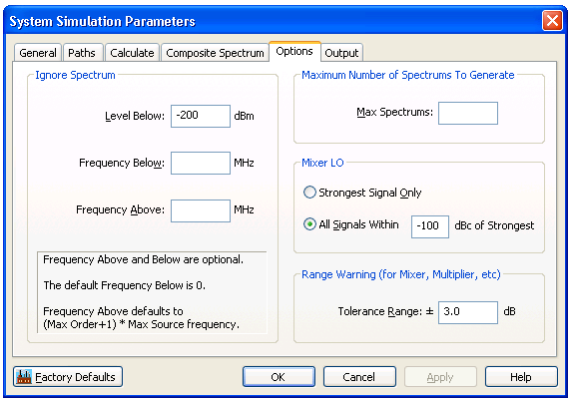
When unchecked 2 points will be used. This parameter is ignored for all continuous frequency sources whose amplitude is dependent on several data points. This option is extremely useful when wideband signals are being simulated through filters since impedance can vary drastically across its bandwidth . If only 2 data points are used then spectrum power and noise will only be represented by these 2 points.

Factory Defaults

Sets all properties to their default values.

System Simulation Parameters - Options Tab

This page contains miscellaneous Spectrasys options.



IGNORE SPECTRUM

This group is used to limit or restrict the number of spectrums created by Spectrasys. These thresholds apply at every calculated node. Consequently, if a signal is heavily attenuated or outside the given frequency range during a portion of the path and are then amplified or frequency translated back into the given frequency range then these thresholds must be set so that the spectrums will not be ignored along the calculation path. Once an individual spectrum is ignored it will not continue to propagate. However, all spectrums previously calculated will still be available at the nodes where there were within the specified limits. For example, If we had a 2 GHz transmitter that had an IF frequency of 150 MHz and we set the 'Ignore Frequency Below' limit to 200 MHz then the entire IF signal would not be present and consequently neither would the 2 GHz RF signal.

Level Below (default = -200 dBm)

All spectrums that are below this threshold will not be created or propagated. This threshold should be set to the highest acceptable level if faster simulations are important. Spectrums are not actually ignored unless they are more than about 20 dB below this threshold since several spectrums can be added together to give a total result that would be greater than this threshold.

Simulation Speed-Up

As with any other type of simulation the more spectral components that need to be processed the longer the simulator time. Setting these limits to only calculate the frequencies and amplitude ranges of interest can speed up the calculation process drastically especially when calculating intermods. However, take caution when setting these limits so that intentional spectrums are not ignored.

Frequency Below (default = 0 Hz)

All spectral components whose frequency is below this threshold will be ignored. Spectrums falling below this limit will not continue to propagate. However, there are several cases where negative frequencies may be calculated at interim steps (i.e. through a mixer) which will be folded back onto the positive frequency axis. This parameter will only affect the final folded frequencies and not the interim frequency steps. Likewise, this is the lower noise frequency limit.

Frequency Above (default = ['*Max Order* (sim)' + 1] times the highest source frequency)

All spectral components whose frequencies are above this threshold will be ignored and will not be created. Spectrums falling above this limit will not continue to propagate. Likewise, this is the upper noise frequency limit.

MAXIMUM NUMBER OF SPECTRUMS TO GENERATE

This group is used to limit or restrict the maximum number of spectrums that will be created.

Max Spectrums

Limits the maximum number of spectrums that are created. Once this limit is reached during a simulation no additional spectrums will be created.

Note: This option must be used with care since a premature limitation of the number of total spectrums will more than likely affect the accuracy of all the results.

MIXER LO

This section controls the effects of all mixer LOs in a simulation.

Strongest Signal Only

When selected, the frequency of the strongest LO signal is used to determine the output frequency of all mixed signals regardless of the number of other signals that may be present on the LO.

All Signals Within X, dBc of Strongest

When selected, all frequencies of LO signals falling within the specified range of the peak LO signal will be used to create new mixed output spectrum.

RANGE WARNING (FOR MIXER, MULTIPLIER, ETC)

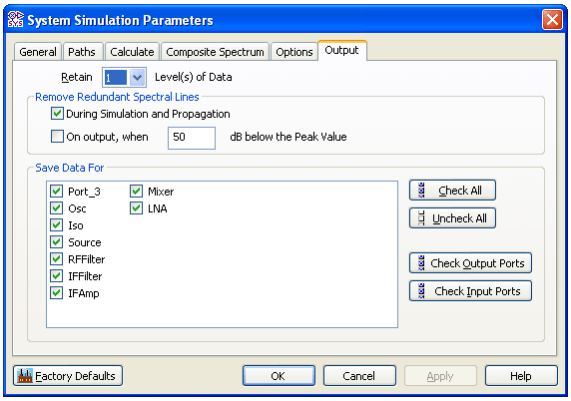
This group is used to control range warnings used by some parts.

Tolerance Range

This threshold range is used by some parts to warn the user when a given power level falls outside the specified range. This range applies to each part on a case by case basis. For example the total LO power for the given mixer will be determined by integrating the LO spectrum and then comparing this power level to the 'LO Drive Level' for the given mixer. If this power level is outside the 'Tolerance Range' window then a warning will be issued for this mixer either indicating that the mixer is being starved or over driven.

System Simulation Parameters - Output Tab

This page contains miscellaneous Spectrasys options.



Retain X Levels of Data

Specifies the number of data levels that will be saved to the dataset. For example, 1 level of data is the data for the top level design only. This parameters only refers to the data retained in the dataset and not the whether a sub-circuit will be analyzed or not. All sub-circuits will always be analyzed.

Remove Redundant Spectral Lines

This section controls the amount of redundant data that will retained and propagated during the simulation. A significant amount of disk storage and simulation time is spend processing spectrums whose powers levels don't contribute to the total power level for the given spectrum type. Signals must be at the same frequency and bandwidth to be considered candidates for spectrum elimination. For instance, a set of carriers will create several intermods. Many of these intermods will fall at the same frequency. However, many of those may be insignificant and don't contribute to the total power of the spectrum. The status of the spectrum elimination will be shown in the Simulation Log for the system analysis.

During Simulation and Propagation

During the simulation redundant spectrum occurs in three places:

1. input carriers that create intermods in a non-linear device
2. processing of a resultant path spectrum that will be used by measurements
3. during the calculation of the total voltage or power spectrum at each node.
   When checked, typically, at least a 2 to 1 increase in simulation speed is observed.

Input Carriers

Input carriers are automatically eliminated due to some amplitude pre-testing that determines if a resultant intermod will exceed the ignore below threshold or not. If the worst case potential intermod voltage is lower than the specified threshold the input carrier will be ignored.

Path Spectrum

There are several different categories of spectrums used for path measurements. They are: Desired, Noise, Total, Phase Noise, and various types of Intermod spectrums. When these resultant path spectrums are being created all spectrums that don't contribute to these resultant spectrums will be eliminated.

Total Spectrum

The spectral plots shown to the user in a graph contains a total trace fore every direction through a node. Only those spectrums that contribute to the total power of voltage of the trace will be retained. Otherwise, they will be ignored.

On Output, When X dB Below the Peak Value

When checked this option will eliminate all spectrums at the same frequency and bandwidth that are below the specified value of the peak signal at the same frequency and bandwidth. This has no affect on any path measurements. The sole purpose of this feature is to reduce the amount of spectrum that graph has to deal with when plotting a power, voltage, or phase spectral plot at the node.

Save Data For

This section controls the data that will be saved to the system analysis dataset. Each part in the schematic is listed and can be checked or unchecked. When checked the data for all nodes of the specified part will be saved. Several buttons have been added for convenience is selecting or unselecting parts.

Check All

When clicked with select all components.

Uncheck All

When clicked will unselect all components.

Check Output Ports

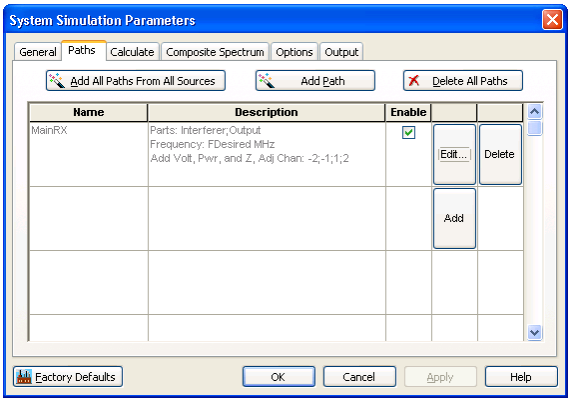
When clicked will select only the output ports.

Check Input Ports

When clicked will select only the input ports.

System Simulation Parameters - Paths Tab

Many measurements require the definition of a path.



Add All Paths From All Sources

Automatically adds all possible paths between sources and output ports.

Add Path

Invokes the Add / Edit Path dialog box.

Delete All Paths

Deletes all paths in the system analysis.

Path Summary Table

This table summarizes the aspects of the path.

Name

User defined name of the path.

Description

User defined description of the path.

Enable

When checked will create and add a path dataset to the workspace tree. When unchecked will remove the existing path dataset from the workspace tree.

Add / Edit Button

When clicked will invoke the Add / Edit Path dialog box.

Delete Button

Will delete the path from the system analysis.

Spectrasys designs in Data Flow schematics

About using Spectrasys designs in Data Flow schematics

SystemVue supports two types of analyses:

- **RF System Analysis** (see *Getting Started with Spectrasys* (sim)): In the following, a design that has been setup for use with RF System Analysis is called a Spectrasys design.
- **Data Flow Analysis** (see *Create Your First Data Flow Simulation* (sim)): In the following, a schematic that has been setup for use with Data Flow Analysis is called a Data Flow schematic (schematic that contains at least one Data Flow part, that is, a part from a library other than *RF Design* or *RF Vendor Kits*).

A Spectrasys design is created to define an RF system with analysis performed in the frequency domain. The analysis is broadband and includes the many harmonics and intermodulation tones associated with nonlinearities and mixers. The analysis inherently includes signal flows for forward transmissions, reflections due to impedance mismatches, and reverse transmissions due to non-ideal isolations.

A Data Flow schematic is created to define a Comms system at the algorithmic level with analysis performed in the time domain for baseband signals and RF signals. The analysis for an RF signal is bandpass with the signal bandpass information bandwidth centered at the RF signal carrier frequency (more typically called the RF characterization frequency). The analysis inherently only deals with signal forward transmission flow for which impedance is not a relevant concept.

SystemVue supports using Spectrasys designs within Data Flow schematics. This enables the design and exploration of Comms systems at the algorithmic level and in the time domain with inclusion of RF systems defined in the frequency domain.

Requirements

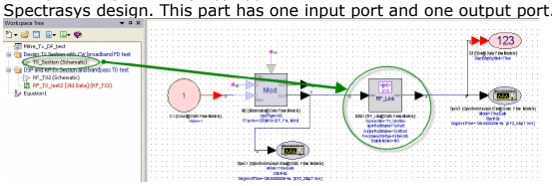
The use of Spectrasys designs in Data Flow schematics requires licenses. The licensing simply requires that the W1719 RF Design Kit be enabled. The feature name **"sv_rfdesign_kit"** can be viewed in the license preference tool as described below:

- From the About SystemVue look for "RF Design Kit"
- From the License Preference tool, either look for the "RF Design Kit" feature in a bundle
- Or look directly for the feature itself

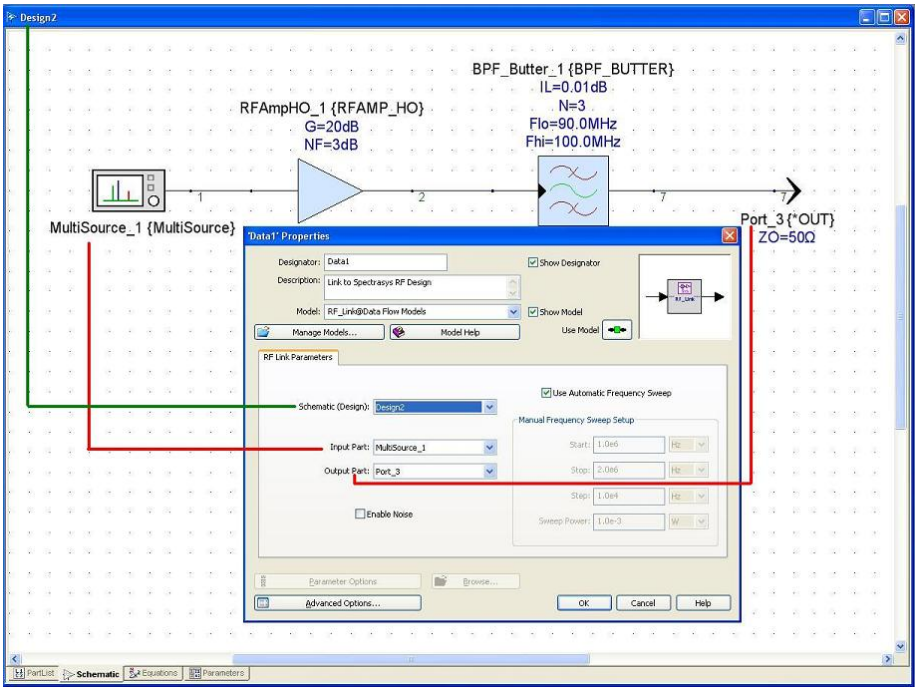
Quick Start Guide

The use of Spectrasys designs in Data Flow schematics is achieved by first selecting (left mouse clicking) a Spectrasys design in the Workspace Tree and then dragging and dropping it into a Data Flow schematic.

When this is done, a Data Flow *RF_Link* (algorithm) part (from the Algorithm Design library, Analog/RF category) appears in the Data Flow schematic and represents the Spectrasys design. This part has one input port and one output port.



The RF_Link part is automatically setup with its **Schematic** set to the name of the Spectrasys design, its **Input Part** set to one of the available sources/inputs (if more than one exist the desired one can be selected in the drop down box) and its **Output Part** to one of the outputs (if more than one exist the desired one can be selected in the drop down box). For designs with only one input and one output there is nothing more that needs to be setup other than optionally enabling the noise analysis (check the **Enable Noise** checkbox) and setting the noise temperature.



With this minimal setup, the RF_Link part is ready for use by a Data Flow analysis. When run in a Data Flow analysis, it automatically extracts from the Spectrasys design the frequency domain data associated with the path starting at **Input Part** and ending at **Output Part**. It uses a sophisticated and automated behavioral modeling process that provides a high fidelity time domain representation of the Spectrasys RF system design defined by its extracted frequency domain data.

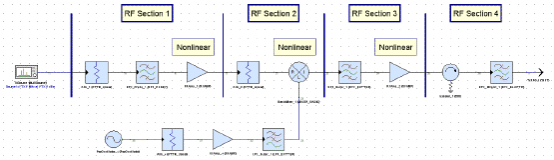
See the *RF_Link* (algorithm) model documentation for details on all of its parameters.

Understanding Spectrasys design use in Data Flow

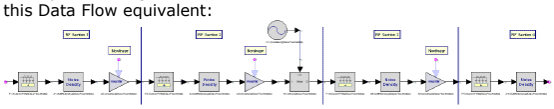
To use a Spectrasys design in a Data Flow schematic (via the RF_Link), an array of frequency domain data is extracted from the Spectrasys design for the path defined from **Input Part** to **Output Part** and converted to its time domain representation for use in the Data Flow simulation.

This process is achieved by automatically reviewing the complete path in the Spectrasys design from **Input Part** to **Output Part** to identify where every nonlinearity exists. All mixers are defined as nonlinearities. The full path is simulated at each analysis frequency in the frequency domain such that all impedance mismatches are included, all signal reverse transmissions are included, all harmonic and intermodulation products produced by the nonlinearities and mixers are included in defining the characteristics for each nonlinearity, and the small signal noise density is obtain using a full noise wave analysis of the design. The full path is then divided into RF sections that end at every nonlinearity and mixer. Each RF section is modeled in the time domain with Data Flow models that include an FIR filter, additive noise density (if the **Enable Noise** checkbox is checked), nonlinear amplifier (if the RF section includes a nonlinearity) and mixer with oscillator LO signal (if the RF section ends with a mixer).

For example, consider this Spectrasys design with all nonlinearity and RF sections highlighted.



This specific design is characterized as 4 RF sections and automatically represented with this Data Flow equivalent:



As can be seen from the above figures, the automatic conversion from the frequency domain Spectrasys design to its time domain equivalent is "correct by construction" with proper positioning of linear filtering, additive thermal noise, nonlinearities and up or down converting mixers. The time domain equivalent is assured to have time causality. Thus, if a frequency domain characteristic is not time causal, such as a frequency domain characteristic with zero phase shift at all frequencies, it will have an appropriate amount of time delay applied to force it to be causal.

Each RF section is replaced with one or more of these Data Flow models:

- *CustomFIR* (algorithm) used to define the RF section small signal gain and phase response versus frequency.
- *AddNDensity* (algorithm) used to define the RF section thermal noise density characteristic versus frequency.

- *Amplifier* (algorithm) used to define the RF section nonlinearity gain and phase change from small signal condition versus power.
- *Mixer* (algorithm) and *Oscillator* (algorithm) used to define the RF section frequency conversion characteristic for upconversion or downconversion.

A Spectrasys design can have any number of input ports and output ports. However, its use in a Data Flow schematic (via RF_Link) is restricted to one specific path starting at a *MultiSource* (rfdesign) or *Input* (rfdesign) part and ending at an *Output Port* (rfdesign) part. The characterization of the path takes into account the full Spectrasys design characteristics. Thus, the Data Flow equivalent replacement for the Spectrasys design path has the highest fidelity in its time domain representation.

The Data Flow input to RF_Link must be a *complex envelope signal* (sim) with a non-zero characterization frequency. This signal is typically defined by use of an *Oscillator* (algorithm), *Modulator* (algorithm) or *CxToEnv* (algorithm) model.

However, RF_Link supports mixer conversions to DC (0 Hz) and output baseband signals. Thus, ZIF (zero-IF) downconverter applications are supported. This includes any non-ideal isolation from LO to mixer input and mixer input to LO, which results in downconverter spectral products at 0 Hz.

By default, RF_Link characterizes the Spectrasys design path over a set of input frequencies in the range [Fc - SR/2, Fc + SR/2] where:

- **Fc** = the input signal complex envelope characterization frequency.
- **SR** = the input signal simulation sample rate.

This input frequency range is divided into 101 equi-distant frequency points. This input frequency range is automatically converted within the Spectrasys design to the appropriate frequency range resulting from frequency conversions through mixers.

Alternatively, RF_Link allows the user to specify their preferred frequency range by unchecking the **Use Automatic Frequency Sweep** checkbox. See the *RF_Link* (algorithm) model documentation for more details.

To enable thermal noise, check the **Enable Noise** checkbox. When this checkbox is checked, the Spectrasys design characterization will include thermal noise from all parts that generate thermal noise. This includes thermal noise from passive parts and noise due to noise figure from active parts. However, thermal noise from the source model associated with the **Input Part** is not included. The RF_Link input noise is presumed to be already included in the input Data Flow signal. The noise analysis is performed over the same frequency range defined above. The system temperature can be changed by the user.

Limitations

Use of Spectrasys designs in Data Flow schematics has certain limitations. These limitations include:

1. The path from **Input Part** to **Output Part** must not include any parts from the RF Design library that provide frequency multiplication, frequency division, or analog to digital conversion. These parts include *FREQ_MULT* (rfdesign), *FREQ_DIV* (rfdesign), *DIG_DIV* (rfdesign) and *ADC_BASIC* (rfdesign).
2. The Doubled Balanced *MIXER_DBAL* (rfdesign) and Table Mixer *MIXER_TBL* (rfdesign) models are not currently supported.
3. Phase noise from the Spectrasys design is ignored.
4. The *IR* (Image Rejection) parameter for mixers in the Spectrasys design is ignored.
5. Interfering signals in Spectrasys that fall in the channel are currently ignored. However, the compression levels of non linear devices in Spectrasys will be affected by these other interfering signals. For example if two tones were introduced in Spectrasys the two tones would determine compression points of non linear devices however the 3rd order intermod created by these two tones is currently being ignored even if it falls within the channel.

Advanced Spectrasys

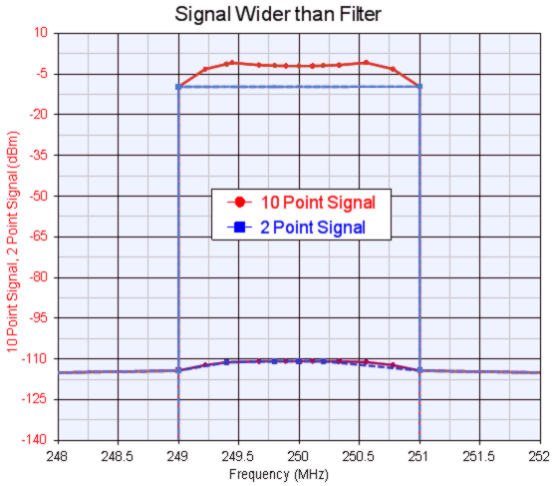
Cascaded Noise Analysis

Cascaded noise figure is an important figure of merit especially for receiver design. Traditional cascaded noise figure equations are not used in Spectrasys. A more elaborate technique is used to include the effects of frequency, impedances or VSWR, bandwidth, image frequencies, and multiple paths in cascaded noise measurements.

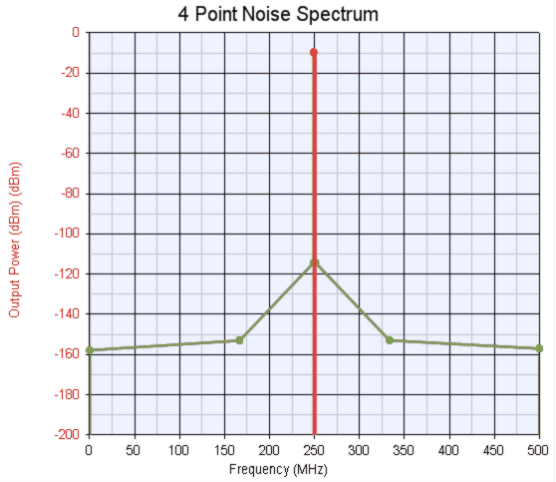
NOTE: Because traditional cascaded noise figure equations are not used in the Spectrasys a new formulation to calculate noise figure was developed. This formulation relies on the *cascaded gain* (sim) measurement to accurately determine the *cascaded noise figure* (sim). The cascaded gain measurement requires a signal to be present in the channel for this measurement to be made. Only noise or intermods in the channel are insufficient.

In general, no adjustments in the noise set up is need to obtain accurate noise analysis. There are a few cases where noise accuracy can be improved:

- Signal bandwidths are as wide or wider than the narrowest filter bandwidths in the simulation
  - Traditional cascaded noise figure equations assume NO bandwidth and that the cascaded noise figure is ONLY valid at a single frequency where noise factor and gain values are taken. In practice bandwidths must be considered. Rarely, do impedances remain constant across a channel especially when any type of filter is involved. Consequently, channel signal and noise power may not be constant across a channel. When using wide channel bandwidths relative to narrow filter bandwidths channel noise power may appear to decrease through a cascaded. This is due to bandwidth reduction. Furthermore, power levels due to channel integration can be less accurate if only a few points are used to represent either the signal or noise through a device whose impedances vary across the channel. For more information see *additional data points* (sim).
  - As can be seen in the following figure a wideband signal is defined to be slightly larger than the narrowest filter bandwidth. If only 2 points are used to represent the signal then the channel power will only be as accurate as those 2 points. If additional data points are add to the signal a better representation of the signal and noise will be achieved. Remember, cascaded noise figure is not only dependent on the channel noise power but also on the cascaded gain.

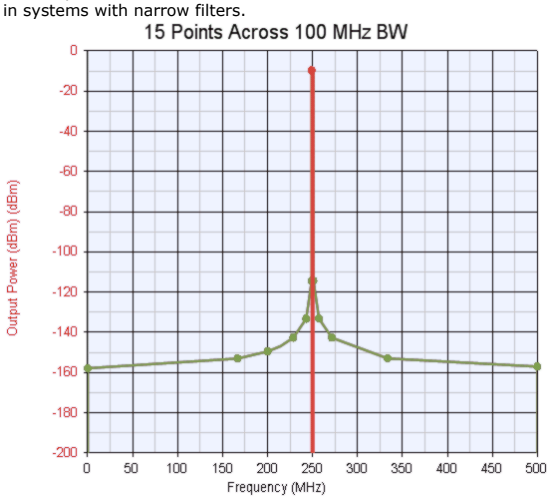


- More accurate noise pedestals would like to be seen on spectrum plots:
  - By default Spectrasys uses a very small number of noise points to represent the entire noise spectrum. The following figure shows what noise would look like in was only represented by 4 points. These 4 noise points are evenly spaced between 0 and 500 MHz (0, 166.67, 333.33, and 500 MHz). You see additional noise points around 250 MHz because of smart noise points insertion which is discussed later. The number of noise points can be changed in 2 ways: 1) adding more evenly spaced noise points across the entire noise spectrum and/or 2) adding more evenly spaced noise points around a user specified bandwidth of desired frequencies.



  - Adding 15 noise points across a 100 MHz bandwidth around the desired signal of

250 MHz show a more accurate representation of the filter noise pedestal. NOTE: Improving the noise spectral shape will generally not improve the accuracy of cascaded noise measurements unless wide measurements are used in systems with narrow filters.



- Smart Noise Point Insertion
  - Another great benefit of the SPARCA simulation technique is that we know which spectrums are desired and which are not. Having this knowledge noise points can be added at the correct frequencies to ensure noise data is collected at the frequency of interest. Adding these noise points to the simulation is called 'Smart Noise Point Insertion'. Without this technique noise simulation through filters would assuredly fail as shown in the figure using 4 noise simulation points. This technique is crucial for all frequency translation models such as mixers, multipliers, and dividers.
- Noise Point Removal
  - As more and more noise points are added to the simulation the simulation will become slower and slower. For this reason noise points that add no value are automatically removed from the spectrum.

NOTE: The '*Calculate Noise* (sim)' option must be enabled before noise figure measurements will be added to the path dataset.

Cascaded Noise Figure Equations

The traditional cascade noise equation is as follows:
$F_{cascade} = F_1 + (F_2 - 1) / G_1 + (F_3 - 1)/(G_1 G_2) + ... + (F_n - 1)/(G_1 G_2 ... G_{n-1})$
where F_n is the noise factor of stage n and G_n is the linear gain of stage n

Note: This equation contains no information about:

- frequency
- impedances or VSWR (see *Two Port Amplifier Noise* (sim))
- bandwidth
- image frequencies
- gain compression
- phase noise
- or multiple paths

These **limitations are very restrictive** and **lead to additional design spins**. For these reasons traditional cascaded noise figure equations are NOT used in Spectrasys.

The SPARCA (Spectral Propagation and Root Cause Analysis) technique used in Spectrasys does not suffer from these restrictive assumptions. Occasionally users are troubled if Spectrasys simulations give different answers than the traditional approach. Using Spectrasys under the same assumptions as the traditional approach will always yield identical answers. To use Spectrasys under the same assumptions only frequency independent blocks like attenuators and amplifiers must be used. Even so, amplifiers must be used as linear devices with infinite reverse isolation. Filters cannot be used since their impedance varies with frequency. Mixers cannot be used since noise from the image band will be converted into the mixer output. Only 2 port devices must be used because of the single path assumption. Of course the bandwidth must also be very narrow.

A more general formulation is used which will include all the effects of frequency, VSWR, bandwidth, images, and multiple paths and will reduce to the traditional case under traditional assumptions.

The general formulation is:
Cascaded Noise Figure[n] = Channel Noise Power[n] + Phase Noise Channel Power[n] - Channel Noise Power[0] - Cascaded Gain[n] (dB), where n = stage number
Cascaded Gain[n] = Desired Channel Power[n] - Desired Channel Power[0] (dB)

Basically, the way to look at this is the cascaded noise figure is equivalent to the noise power added between the first and last stage minus the cascaded gain. Channel noise power includes the amplified noise as well as the noise added by each model. Therefore, cascaded gain must be subtracted to get just the noise added by the system.

Coherency

Since all signals in Spectrasys are treated on a individual basis so must coherency for each of the spectrums created during the simulation. Coherent signals will add in voltage and phase, whereas non-coherent signals will add in power. For example, if two coherent voltages had the same amplitude and phase the resulting power would be 6 dB higher. If they were exactly 180 degrees out of phase having the same amplitude the two signals would cancel each other. If the two signals where non-coherent then the power would only increase by 3 dB irrespective of the phase.

Some of the coherency of Spectrasys can be controlled by the user. The user can determine whether intermods and harmonics add coherently and whether mixer output signals consider the LO signal when determining coherency. See the '*Calculate Tab* (sim)' of the System Simulation Dialog Box for more information on this setting.

How it Works

When a new spectrum is created a coherency number is assigned to each spectrum. These coherency numbers are used to group spectrums together to determine what the resulting total spectrum is after a coherent addition. Coherent additions are especially important at the input to non-linear devices since the total spectrum from coherent signals will yield a different power than individual spectrums. This total power is needed to correctly determine the operating point of the non-linear devices. The coherency number can be viewed by the user when examining the spectrum identification.

The coherency number of a new spectrum will use an existing coherency number if the two spectrums are coherent. Several rules are followed to determine if a newly created spectrum is coherent with an existing spectrum.

All of these rules must be followed before any two signals can be considered coherent:

NOTE: If the 'Coherent Addition' option is unchecked then all intermods and harmonics will always be non-coherent and well as any mixed products out of a mixer regardless of the following comments.

1. **Each source is only coherent with itself OR if the sources have the same reference clock.** When no reference clock is specified for the source only signals created from this source will be coherent assuming the other coherency constraints are met. When multiple sources have the same reference clock name and the other coherency constraints are met their signals are considered to be coherent.
2. **Signals must be of the same type.** Signals generally have the following categories: Source, Intermod, Harmonic, and Thermal Noise. Coherent signals only apply to the same category of signals. For example, a source spectrum can never be coherent with an intermod spectrum and vice versa. A source spectrum can be coherent with source spectrum and intermod spectrum can be coherent with intermod spectrum, etc. Coherency of phase noise is determined by the coherency of the phase noise parent spectrum. If the two parent spectrums that have phase noise are coherent then the phase noises themselves are coherent.
3. **Signals must have the same center frequency and bandwidth.** All coherent signals must have the same center frequency and bandwidth. For example, a 2nd harmonic cannot be coherent with a 3rd harmonic since both the center frequency and bandwidths are not the same. However, if we had a cascade of two amplifiers then the 2nd harmonic generated in 1st amplifier would be coherent with the 2nd harmonic generated in the 2nd amplifier from the same signal source. In this case both the center frequency and bandwidth are the same with both harmonics being created from the same signal source.
4. **Must have the same LO source (mixers only).** When a new spectrum is created at the output of a mixer Spectrasys will determine the coherency of the mixer input signal as well as the LO signal. A new coherency number will be assigned for unique input and LO signals. If there is more than one mixer in the simulation then coherency numbers for the second mixer may come from the first mixer if the all of the above rules are met for the input signal as well as the LO signal. A good example of this is an image reject mixer. A single input port is split 2 ways that drive the input to 2 mixers. A single LO signal is also split 2 ways and phase shifted before being applied to the mixer LO ports. The mixer outputs are combined back together to form the image reject mixer output. Since both mixers have the same input source as well as LO source then all signals that have the same type, frequency, and bandwidth will have the same coherency number.

   NOTE: The coherency number is displayed in the spectrum identification information. This will aid the user in understanding their circuit operation as well debugging any problems. See the 'Spectrum Identification' section for more information.

   NOTE: Phase noise uses the coherency of its parent signal spectrum.

Coherent vs. Non Coherent Addition

"Coherent addition is more conservative than non coherent addition, i.e., the coherent assumption indicated a less linear system than the non coherent equations indicated. In a worst-case scenario, coherent addition should be used."

"When designing low-noise receiving systems, it was found that well-designed cascades usually behave as though the distortion products are adding up non coherently. For the most part, these system have achieved the equivalent of non coherent summation plus one or two dB. With wide-band systems, the cascaded SOI [Second Order Intercept] or TOI [Third Order Intercept] will stay at non coherent levels over most of the frequency range of the system. However, over narrow frequency ranges, the SOI and TOI will increase to coherent summation levels."

"In a well-designed system (where the equivalent intercept points of all the devices are equal), the difference between coherent and non-coherent summation is 4 to 5 dB. When designing a system, it is best to calculate the numbers for both the coherent and non coherent cases to assess the variation likely to be expected over time and frequency." (McClanning, Kevin and Vito, Tom, *Radio Receiver Design* . Noble Publishing, 2000)

Comparing Coherency with Harmonic Balance

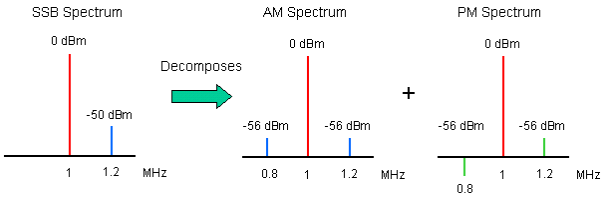
Harmonic balance is a well established nonlinear circuit simulation technique. Signals used it this technique have no bandwidth and all spectrums are of the same type. In harmonic balance it is assumed that two spectrums having the same frequency by definition are coherent.

Coherency and SPARCA

One of the great advantages of SPARCA is that not only is the coherent total spectrum available (which is the only type of spectrum available in harmonic balance) to the user but so are the individual spectrums that make it up. This aids the user in understanding how the design is behaving and is also a great help during the architecture debugging process.

Single Sideband to AM and PM Decomposition

This section will help the user understand how single sideband (SSB) signals are decomposed into AM and PM components. In Spectrasys, non linear devices like digital dividers, frequency multipliers, and frequency dividers will treat all input spectrums other than the peak as single sideband input spectrum which can be decomposed into its AM and PM counterparts. These AM and PM counterparts are then processed as well as the peak spectrum by the non linear part. Please refer to chapter 3 'Modulation, Sidebands, and Noise Spectrums' in William F. Egan's book *Frequency Synthesis by Phase Lock, 2nd Ed 1* for more information.

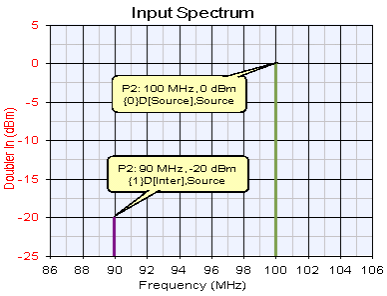


The above illustration shows how the -50 dBm SSB signal located at 1.2 MHz is decomposed into its AM and PM counterparts. The AM sidebands will drop in power by 6 dB and both have the same phase. Whereas the PM sidebands will also drop in power by 6 dB but the lower sideband will be 180 degrees out of phase with the lower sideband in the AM spectrum. When the AM and PM spectrums are added together the lower sidebands cancel out and the -56 dBm upper sidebands add coherently to yield the single upper sideband of -50 dBm.

Single Sideband Decomposition Identification

Offset spectral components generated in frequency multipliers and dividers can be broken down and identified at the output.

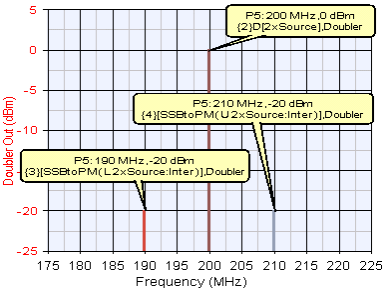
In the following graph the main input to a frequency doubler is identified as a 100 MHz source spectrum labeld '**Source**'. A single interfering signal at 90 MHz is also present and is labeled '**Inter**'.



At the output of the doubler the offset spectrum created around the desired 2nd harmonic can be seen. The identification information for the SSB offset spectrum has the format:

SSBtoPM(*Carrier Side*, *Harmonic* x *Spectrum ID for Peak Input Signal* : *Spectrum ID for Offset Signal*)

- **Carrier Side** - This will either be a **L** or a **U** standing for **Lower** or **Upper** side of the parent harmonic.
- **Harmonic** - This is the multiplication / division factor of the device. i.e '2' for a doubler.
- **Spectrum ID for Peak Input Signal** - This is the identification string for the **peak input** signal to the multiplier / divider.
- **Spectrum ID for Offset Signal** - This is the identification string for the offset spectrum at the **input** that created the resulting output offset spectrum.



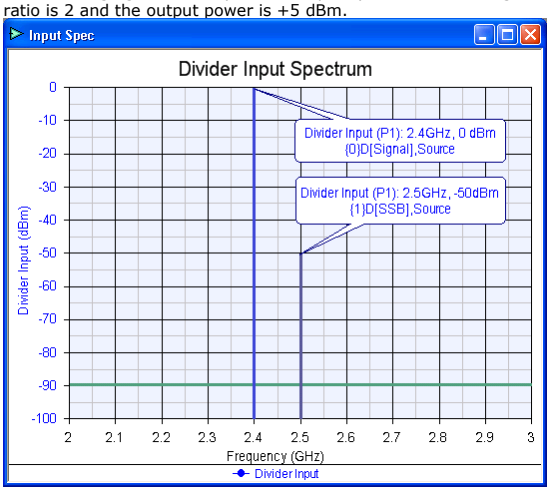
Digital Dividers, Frequency Multipliers and Dividers

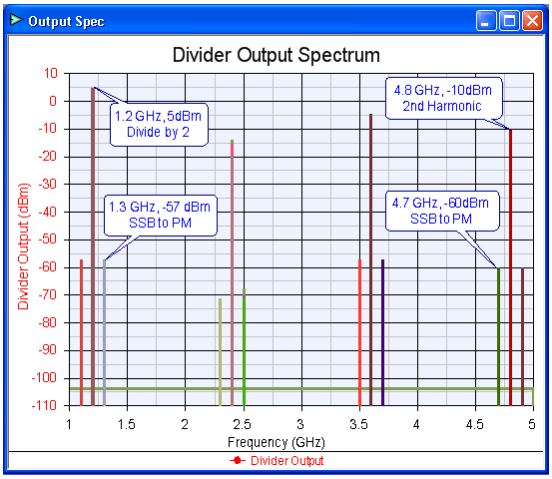
Since digital dividers, frequency multipliers and dividers operate in the hard limiting region, the decomposed AM spectrum is not accounted for, leaving only the PM spectrum. Every input spectrum other than the peak spectrum is treated as a SSB component. There may be multiple SSB components along with the peak input spectrum driving one of these devices. Each SSB component is decomposed into its AM and PM counterparts. Consequently, each harmonic output signal will contain all of the decomposed PM components. Obviously, without filtering between multiplier and dividers stages the number of spectrums grows rapidly due to the multiplication and SSB to AM and PM decomposition. If input SSB spectrum are within 10 dB of the peak input spectrum a

warning is given to the user indicating that the decomposition may not be accurate.

The PM component will change in amplitude according to 20 Log(N) where N is the multiplication ratio. For a divider N = M / D where M is the harmonic of the divider output and D is the division ratio. For example, in the multiplier output PM spectrum was -56 dBm as shown above before multiplication the 2nd harmonic PM spectrum would be 6 dB higher (-56 dBm + 20Log(2)) or -50 dBm. Likewise the 4th harmonic PM spectrum would be -44 dBm (-56 dBm + 20Log(4)). For a divide by 2 device the divide by 2 PM spectrum would be -62 dBm (-56 dBm + 20Log(1/2)) and the 4th harmonic to the divide by 2 device would have PM spectrum at -50 dBm (-56 dBm + 20Log(4/2)).

The following figures and input and output spectrum from a digital divider whose division ratio is 2 and the output power is +5 dBm.



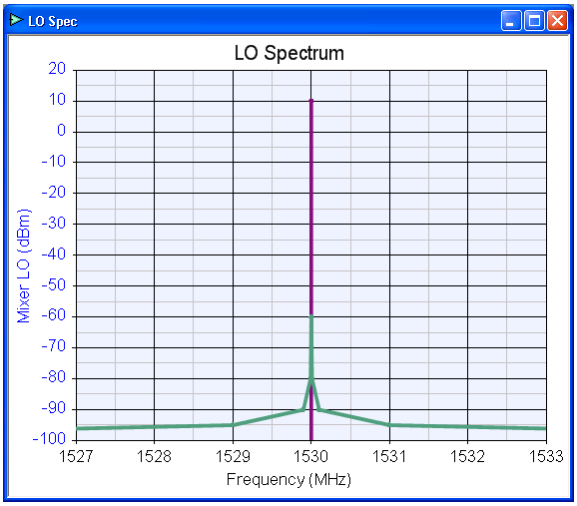


1. William F. Egan (2000), "Frequency Synthesis by Phase Lock", 2nd Ed., John Wiley & Sons, pp. 71-78s

Behavioral Phase Noise

The SPARCA engine supports behavioral phase noise. Phase noise can be specified on certain source and oscillator models. Phase noise is an independent type of spectrum and as such measurements can operate on this spectrum in the presence of others spectrums of different types. This independence allows phase noise to be modified through mixers, multipliers, and dividers without affecting the parent spectrum. This is illustrated in the figure below where the signal spectrum type is shown in one color and the phase noise in another.

Note
Phase noise must be enabled on the '*Calculate* (sim)' tab of the system analysis AND enabled on the source model.



Phase Noise Specification

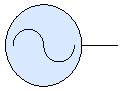
Phase noise is specified by two lists.

- A list of frequency offsets
- A list of power levels in dBc/Hz

Phase noise can also be specified single or double sided. If double sided negative offset frequencies must be used.

Note
There must be the same number of entries in both frequency and amplitudes lists. If not a warning will be issued and the lists will be truncated to the smaller of the two lists.

The following example shows the single sided specification of an oscillator. In this example, phase noise is specified for -70, -90, -100, -105, and -110 dBc/Hz for the respective offset frequencies of 1, 10, 100, 1000, 10000 kHz. The carrier center frequency is 1530 MHz.



Osc
F=1530MHz
Foff=1;10;100;1000;10000KHz
PhaseN=-70;-90;-100;-105;-110dB10

The list data does not need to occur in ascending frequency order, though this is a more readable format. However, the first frequency entry will be associated with the first phase noise power level entry, the 2nd frequency entry with the 2nd phase noise power level entry, etc.

Enabling Phase Noise

Phase noise must be enabled in two places before phase noise spectrums will propagate through the analysis.

- In the source itself (see the phase noise enable parameter)
- In the *system analysis* (sim) dialog

Phase Noise Simulation

Each phase noise spectrum is associated with a parent signal spectrum. The phase noise goes through the same transfer function as its parent. Through all linear models the phase noise will be transformed as is the signal spectrum. Through nonlinear models the phase noise may remain the same, increase, or decrease in amplitude relative to its parent spectrum. For example, through a frequency doubler the phase noise will increase 6 dB relative to its parent spectrum.

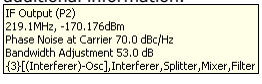
Phase noise is also processed by the mixer. Mixed output spectrum is a combination of the input and LO spectrum phase noise.

Phase Noise Coherency

The coherency of the phase noise is the same as the coherency of the parent spectrum.

Viewing Phase Noise Data

In a graph the phase noise can either be displayed in dBm/Hz or scaled by the channel measurement bandwidth. When the mouse is placed over the phase noise the tool tip information will show any bandwidth scaling that should be applied to the phase noise spectrum. See the '*Composite Spectrum Tab* (sim)' tab of the dialog box reference for additional information.



The absolute frequency point along with its power is displayed. The phase noise power at the carrier frequency is displayed in dBc/Hz. The bandwidth scaling factor is also displayed. The last line shows the spectrum coherency number in braces then the phase noise equation in square brackets followed by the path that the phase noise took to get to the view destination.

The phase noise in path measurements are always scaled to the appropriate bandwidth before the results are used.

Spectrum Analyzer Display

The spectrum analyzer mode is a display tool to help the user visualize what the simulation would like on a spectrum analyzer. This mode is extremely useful when out of phase signals may cause the total spectrum to cancel.

NOTE: This mode is NOT used for any path measurement data and is for display purposes only. The parameters used in this mode have NO bearing on the accuracy of the real simulation results.

The spectrum analyzer mode performs a convolution of a 5 pole gaussian filter on the total spectrum trace. A gaussian filter is used because this is the type of filter used in real spectrum analyzers.

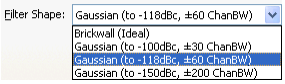
The challenges associated with convolving the spectrum is as follows.

- Because of broadband noise frequencies can literally go from DC to daylight
- Frequencies are not evenly spaced
- Spectrums generally appear in groups with larges spacing between groups

Because of practical issues associated with performing a convolution several parameters have been added to this mode to decrease the simulation time and restrict the amount of collected data.

Filter Shape

The user can use a brick wall filter or the traditional spectrum analyzer gaussian filter. The variations in gaussian filters are used to determine frequency at which the convolution will begin and end. These values are specified in terms of channel bandwidths. Obviously, the wider the filter end frequency is the longer the simulation time will be and the more data will be collected. The amplitude range of the filter that is associated with the given end frequency is also given for convenience.



The analyzer modes settings are found on the 'Composite Spectrum' tab of the system analysis.

Analyzer Simulation Process

The analyzer goes through the following steps before the analyzer trace is ready to display in a graph.

1. The total spectrum trace for the path is created.
2. This spectrum is broken down into bands to find groups of spectrums. Each group will have guard bands equivalent to the stop band of the selected filter.
3. Before the convolution occurs the group must be discretized. Since some spectrums have bandwidths as small as 1 Hz they can be completely ignored during a standard discretization process. To eliminate skipped spectrums the discretization process will keep track of peak spectrum values that may fall in between discretization points. This process guarantees that peaks will not be missed.
4. The analyzer noise floor will be added.
5. The discretized spectrum will be convolved with the gaussian filter.
6. Noise randomization is added.
7. Analyzer spectrum will be displayed on a graph.

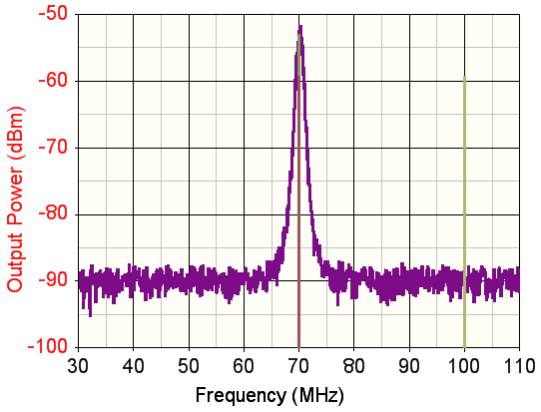
NOTE: Noise floor spectrums between spectrum groups is not shown in the spectrum analyzer mode.

Frequency Limits

Frequency limits have been added as a simulation speedup. There is a lot of data and simulation time required for the analyzer mode especially as the frequency ranges become very large. The users can specify a frequency band of interest to apply the spectrum analyzer mode to. These settings require a start, stop, and step frequency. The step frequency can be set within a given range. Warnings will be given to the user if the step size becomes so small that maximum number of analyzer simulation points are exceeded. On the other end of the range, a warning will also be given if the step size becomes so large that the accuracy has been compromised. In these cases the analyzer will select a good default.

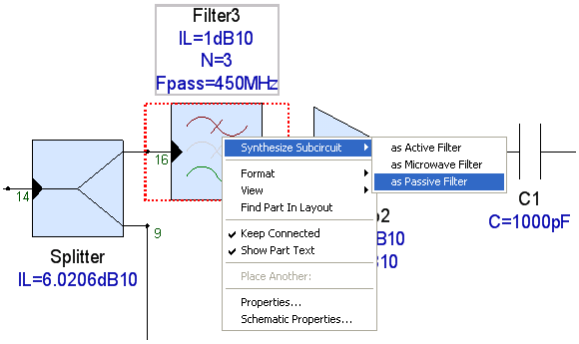
Example

The following figure shows a good example of the spectrum analyzer mode. This plot is taken from an 'Image Rejection Mixer' example where 2 signals at 70 MHz add coherently to give an increase of 6 dB and where 2 signals at 100 MHz are 180 degrees out of phase and thus cancel. Using the spectrum analyzer mode we quickly see the peak at 70 MHz and even though we see the individual spectrums at 100 MHz we know their total must be equivalent to or lower than the noise floor. Random noise is also shown in the example.

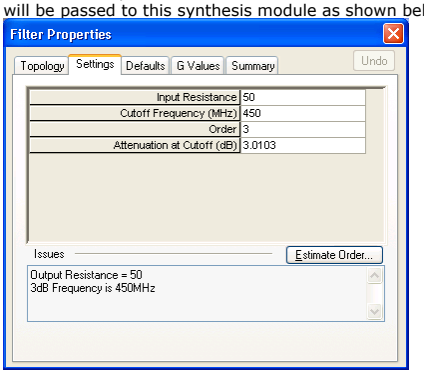


Synthesis

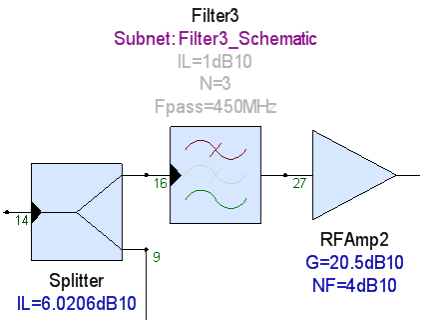
Some behavioral models can directly synthesized from Spectrasys. Right clicking on the behavioral model will bring a context sensitive menu. This menu will list the synthesis modules available for the given model.



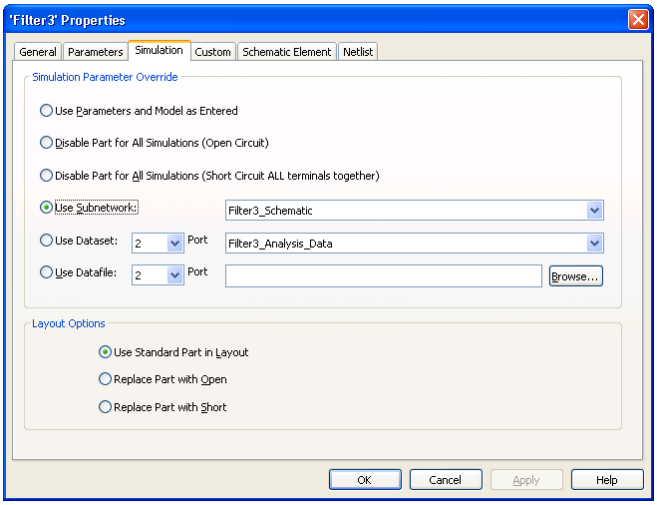
The selected synthesis module will be invoked and the parameters of the behavioral model will be passed to this synthesis module as shown below.



Once the model has been synthesized the synthesized circuit is substituted back into the behavioral model.



At this point the parameters for the behavioral model will be disabled.



See the specific Direct Synthesis from Spectrasys section for more information about each synthesis tool.

Directional Energy (Node Voltage and Power)

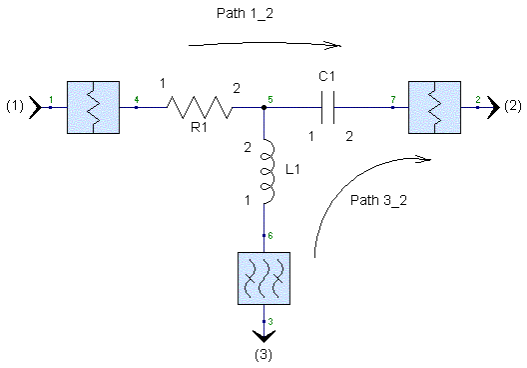
When three or more connections occur at a node a convention must be established in order to make sense of the information along path.

NOTE: The path value reported for a node along a path that has more than three or more parts is the value seen by the series part in the path entering the node.

For example, in the following example we have defined two paths 'Path1_2' which is the path from node 1 to node 2 and 'Path3_2' which is the path from node 3 to node 2. On a level diagram or in a table the value reported at node 5 for 'Path1_2' would be the value of the measurement leaving terminal 2 of the resistor R1 entering node 5. Likewise, the impedance seen along this path is that seen looking from terminal 2 of the resistor R1 into node 5. Consequently, the impedance seen by R1 is the L1 to port 3 network in parallel with the C1 to port 2 network. In a similar manner the value reported at node 5 for the 'Path3_2' would be the value of the measurement leaving terminal 2 of inductor L1 entering node 5. The impedance for the node looking from terminal 2 of inductor L1 is most likely to be completely different from the impedance seen by R1 or even C1 because from the inductors perspective, the R1 to port 1 network is in parallel with the C1 to port 2 network.

Spectrasys knows about the direction of all of the paths and will determine the correct impedance looking along that path. As a result all measurements contain the correct values as seen looking along the path of interest.

Remember, absolute node impedance and resulting measurements based on that impedance don't make any sense since they are totally dependent on the which direction is taken through the node.



Transmitted Energy

When an incident propagating wave strikes a boundary of changing impedances a transmitted and reflected wave is created. Obviously, the transmitted wave is only the energy of the wave flowing in the forward direction. The *SPARCA* (sim) engine calculates the transmitted wave ONLY from the incident wave. This transmitted energy is what is used for all path measurements.

See 'No Attenuation Across a Filter' for an illustration of this principle.

Circuit Co-Simulation

Improved accuracy can be achieved if specific circuit implementations are known for any given system or behavioral model. *SPARCA* (sim) is a dedicated RF architectural / system simulation engine that is used by Spectrasys.

Linear Circuit

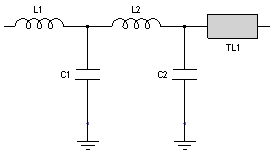
The *SPARCA* (sim) simulation technique is tightly coupled with linear simulation. Linear

lineup works well. However, when tight linear loops are created accuracy will begin to decrease. When sub-circuits are create strictly out of linear components the linear simulator runs on this sub-circuit and feeds this information to Spectrasys to be used in its simulation.

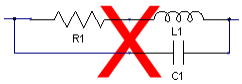
Linear Co-Simulation

Spectrasys automatically co-simulates with linear components such as resistors, capacitors, inductors, transmission lines, etc.

Linear circuits that don't have internal loops can co-simulate **directly** with the Spectrasys *SPARCA* (sim) engine. A good example is the following circuit.



However, a circuit that has a tight loop as shown will reduce the accuracy and slow the simulation speed if this circuit is run at the same hierarchy level as Spectrasys.



Note
Any type of linear circuits can be placed in a sub-circuit. When in a sub-circuit the linear simulator is called and the number of spectrums needed to represent the linear response is drastically reduced and the simulation speed will be increased.

Caution
When a Spectrasys behavioral model is found in a schematic with linear parts the Spectrasys simulation engine will run on that schematic. Isolation between Spectrasys and linear analysis is achieved by keeping Spectrasys behavioral models out of linear schematics.

Linear Co-Simulation Use Model

Linear co-simulation is automatic when linear components are placed in a sub-circuit. The steps are as follows:

1. Add a new sub-circuit schematic (design).
2. Place the linear components and ports in the sub-circuit.
3. Drag the sub-circuit schematic from the workspace window onto the schematic of interest.

Non-Linear Circuit

The *SPARCA* (sim) technique is not appropriate for non-linear circuit simulation. Harmonic Balance is a proven non-linear simulation technique that works very well for non-linear circuits. Harmonic balance is a frequency domain analysis technique for simulating distortion is nonlinear circuits. Harmonic balance simulation obtains frequency-domain voltages and currents, directly calculating the steady-state spectral content of voltages or currents in the circuit.

Caution
In SystemVue the only non-linear circuit component allowed for co-simulation is the X-parameters model. For complete non-linear circuit simulation the user can use Spectrasys within the Genesys environment.

Non-Linear Co-Simulation

Non-linear simulation uses a harmonic balance simulation technique that is not used by linear simulation. Harmonic balance simulation terminates once a predetermined level of convergence is achieved. For this reason non-linear simulation has additional convergence parameters not needed for a linear simulation. Linear aspects of non-linear models can generally simulate in a linear simulator.

Non-Linear Co-Simulation Use Model

A Circuit Link part is used to point to a non-linear circuit. The Circuit Link part has some basic parameters needed by the harmonic balance simulation engine that it uses. The easiest way to create a non-linear co-simulation is as follows:

1. Add a new sub-circuit schematic (design).
2. Place the linear, non-linear components and ports in the sub-circuit.
3. Drag the sub-circuit schematic from the workspace window onto the schematic of interest. When non-linear circuit components are are detected in a sub-circuit that is being dragged and dropped onto another schematic the user will be prompted if they want to create a Circuit Link part for this sub-circuit. When prompted to create a Circuit Link part say **Yes**.
4. A Circuit Link part will be created and the schematic that it will simulate is the one used during the drag and drop operation. When this schematic is set in the Circuit Link part the number of external ports is auto detected.

Signal spectrum differences between Harmonic Balance and Spectrasys

In harmonic balance their is no signal bandwidth, voltage and currents exist at a point. In Spectrasys every spectrum has bandwidth. By definition a CW has a 1 Hz bandwidth.

During co-simulation the average power of Spectrasys sources will be used as sources for the harmonic balance co-simulation. Obviously, harmonic balance will ignore Spectrasys sources bandwidths.

Note
When harmonic balance finishes the co-simulation and hands it's calculated spectrum back to Spectrasys the original bandwidth information contained by Spectrasys will lost.

Coherency differences between Harmonic Balance and Spectrasys

In harmonic balance by definition all spectral components at the same frequency are all phase coherent with each other. In Spectrasys there are several rules that govern coherency and users have control over source coherency. See the Coherency section for more information on Spectrasys coherency.

The best way to think of this is to imagine two signal generators on a lab bench and combined together. If both generators are set at the same carrier frequency, power, level, and phase these two carriers will be non-coherent since the signal generators are not **locked** to the same phase reference and will have random phases compared to each other. Once the generators are **phase locked** together by using a common reference clock (oscillator) these two carriers become phase coherent. When phase coherent the output power increases by 6 dB (assuming both carriers have the same phase and amplitude) and when non-coherent only by 3 dB.

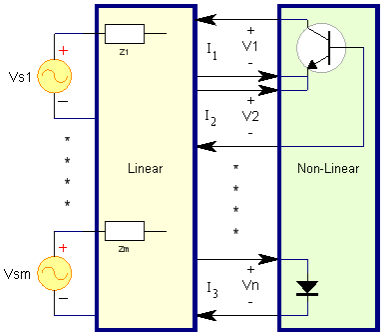
In the signal generator example **harmonic balance** is equivalent to **always** have the reference clock connected. In **Spectrasys** by default the reference clock is **not connected**, however reference clocks can be added to Spectrasys sources. Furthermore, multiple reference clocks are supported in Spectrasys.

Before Spectrasys hands off source spectrum to a harmonic balance co-simulation it will combine all the spectrum at the same frequency both coherently and non-coherently and determine a single source voltage for that harmonic balance source frequency.

Non-Linear Circuit Simulation Background

Basic Concepts of Harmonic Balance

In harmonic balance the circuit to be simulated is first separated into linear and non-linear parts. The internal impedances Zi of the voltage sources are associated with the linear parts. Admittance (Y) matrices are used for the **linear** part to determine the voltages (V1 ... Vn) and currents (I1 ... In) for all the linear / non-linear interface nodes (1 ... n) due to all the sources (Vs1 ... Vsm). For the **non-linear** part currents are modeled as a function of charge. When the currents and voltages at the linear / non-linear interface nodes are balanced, by Kirchhoff's current law for all specified harmonics we have a **converged** solution.



Harmonic Balance Strengths

Harmonic balance is ideally suited as a non-linear, frequency-domain, steady-state circuit simulation technique.

Harmonic Balance Limitations

Harmonic balance is limited in that the signal must be quasi-periodic and representable as a superposition of a number M of discrete tones. As M becomes large, the amount of required internal memory becomes excessive since the internal matrix size grows as M^2.

Increasing Simulation Speed

There are several options available to increase the simulation speed of Spectrasys.

Note
For additional understanding of why these parameters affect the speed of the simulation see 'Propagation Basics'.

Loops

The larger the gain around a closed loop the longer it will take for spectrums to fall below the 'Ignore Spectrum Level Below' threshold. Furthermore, more data is collected each time spectrums are propagated around a loop.

A quick test to verify if this is the problem with the simulation is to increase the isolation of one of the main parts in the loop to a very high value, like 200 or 300 dB. This will force loop spectrum to fall below the 'Ignore Spectrum Level Below' threshold.

Ignore Spectrum

There are 3 Ignore spectrum parameters that affect the simulation speed and the amount of data collected. They are 1) 'Ignore Spectrum Level Below', 2) 'Ignore Spectrum Frequency Below', and 3) 'Ignore Spectrum Frequency Above'. These parameters can be changed to reduce the frequency range and amplitude dynamic range for which the simulator is collecting and analyzing the data. The biggest speed improvement usually comes from raising the 'Ignore Spectrum Level Below' threshold.

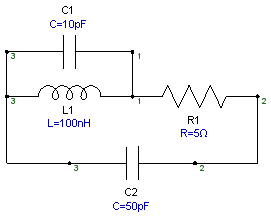
Redundant Spectrum Reduction

A significant amount of simulation time can be used in calculating spectrums that don't affect the overall accuracy of power levels of the spectrums of interest. The calculation of these spectrums can be eliminated to increase simulation speed. See the Output Tab of the system analysis dialog box for more information.

Tight Loops

The more nodes and parts in a design, the more spectrums that will be created and propagated. Linear parts formed in tight loops should be moved to a subcircuit and called from the parent design. If **only** linear parts exist in the a subcircuit a linear analysis is used for the subcircuit instead of the spectral propagation engine. If the system simulator finds a nonlinear behavioral model on the subcircuit the spectral propagation engine will be used.

For example, a circuit such as the following should be moved to a **LINEAR ONLY** subcircuit.



Intermods

One of the largest time consuming operations in Spectrasys is the calculation of a large number of intermods. The number of intermods generated is determined by the 'Maximum Order' of intermods and the number of carriers used to create the intermods. Besides reducing the maximum intermod order raising the 'Ignore Spectrum Level Below' threshold will eliminate all intermods below this threshold.

Spectrum Analyzer Display Mode

During the system simulation an analyzer trace will be created for every node in the system. Consequently, for systems with large number of nodes the integrated analyzer traces alone can be time consuming if the analyzer properties are not optimized. The simulation speed can be reduced by a careful selection of "Analyzer Mode" settings. If large frequency ranges are integrated with a small resolution bandwidth then the amount of data collected will be much larger and the simulation speed will decrease. Furthermore, enabling the 'Randomize Noise' feature may also slow down the simulation. In order to increase the simulation speed with the 'Analyzer Mode' enabled the user can disable the 'Randomize Noise' feature, increase the 'Resolution Bandwidth', and/or limit the frequency range over which a spectrum analyzer trace will be created. See the 'Spectrum Analyzer Display' section for additional information.

Paths

The more paths contained in the simulation the longer it will take to simulate and the more data that will be collected. Delete all unnecessary paths.

Noise

The more noise points that are simulated, generally the longer it takes the simulation to run. See 'Broadband Noise' and 'Cascaded Noise Analysis' sections for additional information on controlling noise.

Mixers

The more LO Signals used or the higher the 'Maximum Order' to create new mixed spectrum the longer simulation time and more data that will be collected. The number of LOs used in the simulation can be reduced to increase the simulation speed. The intermod 'Maximum Order' can be decreased to reduce the number of spectrums created.

Reducing the File Size

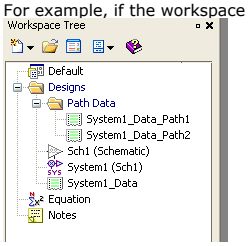
Most of the size of a file is due to simulated data.

The file size can be reduce in one of two ways: 1) completely removing the datasets and 2) only keeping the node data of interest.

Completely Removing Datasets

1. Closing all graphs and tables (this will keep these items from complaining when they have no data to show).
2. Deleting all system analysis and path datasets. Be sure to include all those associated with sweeps. NOTE: Be careful not to delete static data like S parameters and other data that doesn't change during the simulation.
3. Save the file.

For example, if the workspace tree was:



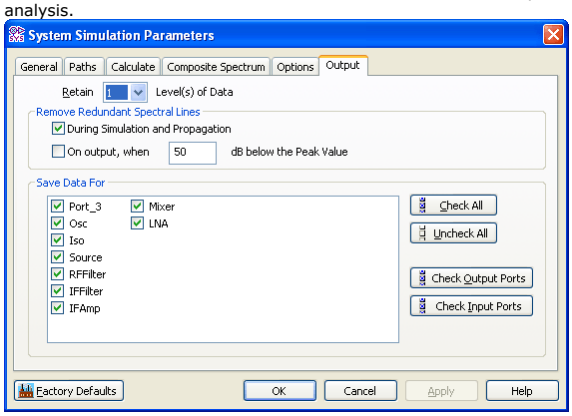
Then the file size could be reduced by deleting:

- System1_Data_Path1
- System1_Data_Path2

- System1_Data

Keeping Node Data of Interest

The node data that the simulator retains is controlled on the 'Output Tab' of the system analysis.

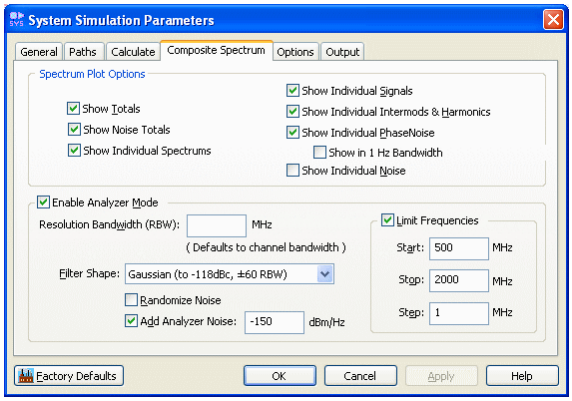


Only checking the devices of interest will reduce the file size.

NOTE: Even though some devices are not checked all data for these devices are calculated during the simulation. These output options only affect the data being saved to the system analysis dataset.

System Simulation Parameters - Composite Spectrum Tab

This page controls how the data is displayed on a graph.



Spectrum Plot Options

This information only affects the displayed output and not internal calculations. Spectrums can be displayed in groups or individually.

Show Totals

Shows a trace representing the total power traveling in each direction of travel through a node. This total includes all signals, harmonics, intermods, thermal noise, and phase noise. For example, if three parts were connected at a particular node then power would be flowing in three different directions. A unique color would represent each trace.

Show Noise Totals

Shows a trace representing the total noise power traveling in each direction of travel through a node.

Show Individual Spectrums

When checked individual spectrums will be displayed otherwise groups will be shown.

Note: Individual spectrum identification information cannot be shown for a group.

When grouping spectrum, all spectrums in the same group are represented by a single trace.

Show Individual Signals (Show Signal Group)

Shows a trace for each fundamental signal spectrum or signal group.

Show Individual Intermods & Harmonics (Show Intermods & Harmonics Group)

Shows a trace for each intermod and harmonic spectrum or intermod and harmonic group.

Show Individual PhaseNoise (Show PhaseNoise Group)

Shows a trace for each phase noise spectrum or phase noise spectrum group.

Show in 1 Hz Bandwidth

When checked shows phase noise in dBm/Hz. When unchecked the phase noise will be scaled according to the channel measurement bandwidth.

Show Individual Noise (Show Noise Group)

Shows a trace for each noise spectrum or noise spectrum group.

Enable Analyzer Mode

This checkbox enables the analyzer mode and its settings. This mode can help the engineer visualize what the simulated spectrum would look like on a common spectrum analyzer. The analyzer mode has been added to allow the user to correlate the simulation data with spectrum analyzer data measured in the lab.

Note: This mode affects only the display and in no way will affect the integrated measurements.

Resolution Bandwidth (RBW)

The analyzer mode can be thought of just like a spectrum analyzer that has a sweeping receiver and peak detects the total power within the resolution bandwidth. The user can specify the resolution bandwidth of this sweeping receiver. The default resolution bandwidth is the 'Measurement Channel Bandwidth'.

Filter Shape

This parameter determines the shape of the resolution bandwidth filter. This filter shape is analogous to the resolution bandwidth filter shape in a spectrum analyzer which uses a 5 pole Gaussian filter. Likewise in the system analysis this same filter is also used. The user is able to select three widths for this particular filter which are based on an integer number of channel bandwidths. No spectrum integration will occur outside the width of this filter. This filter width is used to reduce the amount of data collected, saved, and processed. A brickwall filter can be created theoretically and is also included.

Brickwall (Ideal)

This filter is an ideal rectangular filter whose skirts are infinitely steep.

Gaussian (to -100 dBc, 30 Chan BW)

Data will be ignored that is farther than 30 channels away from the center frequency. Attenuation 30 channels from the center will be about -100 dBc.

Gaussian (to -117 dBc, 60 Chan BW)

Data will be ignored that is farther than 60 channels away from the center frequency. Attenuation 60 channels from the center will be about -117 dBc.

Gaussian (to -150 dBc, 200 Chan BW)

Data will be ignored that is farther than 200 channels away from the center frequency. Attenuation 200 channels from the center will be about -150 dBc.

Randomize Noise

When enabled, random noise will be added around the resulting analyzer sweep. The output trace will be more representative of a typical spectrum analyzer, at the expense of additional computation time.

Add Analyzer Noise

All spectrum analyzers have a limited dynamic range. They are typically limited on the upper end by intermods and spurious performance at an internal mixer output. On the lower end they are limited by noise of the analyzer. This noise is a function of the internal architecture of the specific spectrum analyzer and internal RF attenuator.

Analyzer Noise Floor

Specifies the noise floor in dBm/Hz of the spectrum analyzer mode.

Limit Frequencies

When checked the frequency range of the analyzer mode is limited. By default the entire spectrum from the 'Ignore Spectrum Frequency Below' lower frequency limit to the highest frequency limit of 'Ignore Spectrum Frequency Above' will be processed by the analyzer for every node in the system. In some cases this may be very time consuming. In order to improve the simulation speed and just process the area of interest, frequency limits can be enabled to restrict the computation range of the analyzer.

Start

Beginning frequency of the analyzer.

Stop

Ending frequency of the analyzer.

Step

This is the frequency step size between analyzer data points. The step size can be reduced until the maximum number of simulation points is reached.

Number of Simulation Points

The number of simulation points used for the graph is determined internally within Spectrasys. This parameter cannot be changed by the user. Since Spectrasys can deal with large frequency ranges, the amount of data collected for a single spectrum analyzer trace could be enormous. Furthermore, the analyzer function is not a post processing function and the number of simulation points cannot be changed without rerunning the

simulation. In order to better control the amount of data collected, which is proportional to the simulation time, Spectrasys internally determines the number of simulation points to use.

Simulation Speed-Ups

During the system simulation the analyzer will create an analyzer trace for direction of travel for every node in the system. Consequently, for systems with large number of nodes, the convolution routines used to calculate the analyzer traces alone can be time consuming if the analyzer properties are not optimized. If simulation speed is important then using the narrowest filter shape will have the best simulation speed.

File Size

The size of the data file will increase when the analyzer mode is enabled. Furthermore, the file size can grow rapidly depending on the settings of the analyzer mode. For example, the smaller the resolution bandwidth the more data points are needed to represent the data, the larger the data file will be, and most likely the simulation time will increase.

Analyzer Troubleshooting

What does it mean when the signal doesn't seem to be lined up with the integrated spectrum? All this means is the frequency resolution isn't small enough to accurately represent the signal of interest. If this is the case, there are a few things that can be done to increase this resolution. First, the resolution bandwidth can be reduced. If this is inadequate, the 'Limit Frequencies' feature should be enabled and the user can specify the 'Start', 'Stop', and 'Step' frequencies used for the analyzer.

Troubleshooting

Cannot Load a C++ Model Library

This can be because of one of the following causes:

S.No	Causes	Solution
1.	A DLL will not be loaded if it does not contain any C++ model in it or none of the models had defined interface correctly.	Make sure that you are following the documentation correctly for creating C++ models.
2.	The C++ models built with one version of SystemVue are not guaranteed to work with future/previous versions of SystemVue.	Rebuild the DLL against the SystemVue installation that you are using to load the DLL.

Spectrasys Measurement Index

Path Measurements

Spectrasys General Measurements

Name	Description	Syntax	Spectrum Type	Equation
ACF	*Adjacent Channel Frequency* (sim)	ACF(Side, iChanNo) Side: U or L Upper / Lower iChanNo: any int > 0	None	
CF	*Channel Frequency* (sim)	CF	None	
DCR	*Desired Channel Resistance* (sim)	DCR	Desired	Average resistance at CF
DCPH	*Desired Channel Phase* (sim)	DCPH	Desired	Phase of desired CW signal in the channel
OCF	*Channel Frequency (Offset)* (sim)	OCF	None	
	Group Delay (sim)			
ICF	*Interferer Channel Frequency* (sim)	ICF	None	
IMGF	*Image Channel Frequency* (sim)	IMGF	None	

Spectrasys Power Measurements

Name	Description	Syntax	Spectrum Type	Equation
ACP	*Adjacent Channel Power* (sim)	ACP(Side, iChanNo)	**Total**	Channel power at ACF(Side, Chan No)
AN	*Added Noise Power* (sim)	AN	Same as CNF	AN[i] = CNF[i] - CNF[i-1], Where AN[0] = 0 dB
CCOMP	*Cascaded Compression Point* (sim)	CCOMP	None	CCOMP[i] = Summation(COMP[i]) from 1 to i dB
CGAIN	*Cascaded Gain* (sim)	**cgain**(X)	Same as X	CGAIN[i] = X[i] X[0]
CIMCP	*Intermod Channel Power (Conducted)* (sim) (All Orders)	CIMCP	Same as TIMP & DCP	CIMCP[i] = TIMP[i-1] + gain(DCP)[i], Where CIMCP[0] = -300 dBm
CIMCP2	2 nd Order Conducted Intermod Power	**cimcpn**(CIMCP, 2)	Same as CIMCP	CIMCP for 2 nd Order
CIMCP3	3 rd Order Conducted Intermod Power	**cimcpn**(CIMCP, 3)	Same as CIMCP	CIMCP for 3 rd Order
CND	*Channel Noise Density* (sim)	CNP	Same as CNP	Same as CNP but in a 1 Hz bandwidth
CNDR	*Carrier to Noise and Distortion Ratio* (sim)	**cndr**(DCP, NDCP)	Same as DCP & NADP	CNDR[i] = DCP[i] - NDCP[i]
CNF	*Cascaded Noise Figure* (sim)	**cnf**(DCP, CNP, PNCP)	Same as CNP & PNCP	CNF[i] = CNP[i] + PNCP[i] - CNP[0] - cgain(DCP)[i]
CNP	*Channel Noise Power* (sim)	CNP	**Noise**	Noise power at CF
CNR	*Carrier to Noise Ratio* (sim)	**cnr**(DCP, CNP)	Same as DCP & CNP	CNR[i] = DCP[i] - CNP[i]
COMP	*Compression Point* (sim)	COMP	None	Calculated compression of each stage
CP	*Channel Power* (sim)	CP	**Total**	Total power at CF
DCP	*Channel Power (Desired)* (sim)	DCP	**Desired**	Desired power at CF
GAIN	*Power Gain* (sim)	**gain**(X)	Same as X	GAIN[i] = X[i] X[i-1], Where GAIN[0] = 0 dB
GIMCP	*Intermod Channel Power (Generated)* (sim) (All Orders)	GIMCP	**Generated Intermod**	Generated intermod power at CF for all orders
GIMCP2	2 nd Order Generated Intermod Power	**gimcpn**(GIMCP, 2)	Same as GIMCP	GIMCP for 2 nd Order
GIMCP3	3 rd Order Generated Intermod Power	**gimcpn**(GIMCP, 3)	Same as GIMCP	GIMCP for 3 rd Order
ICGAIN	*Interferer Cascaded Gain* (sim)	**cgain**(ICP)	Same as ICP	ICGAIN[i] = ICP[i] ICP[0]
ICP	*Interferer Channel Power* (sim)	ICP	**Desired**	Interferer power at ICF
IGAIN	*Interferer Gain* (sim)	**gain**(ICP)	Same as ICP	IGAIN[i] = ICP[i] ICP[i-1], Where IGAIN[0] = 0 dB
IIP	*Input Intercept Point (All Orders)* (sim)	**iip**(OIP, ICGAIN)	Same as OIP & ICGAIN	IIP[i] = OIP[i] - ICGAIN[i], where OIP and IIP contain all orders
IIP2	2 nd Order Input Intercept Point	**iipn**(IIP, 2)	Same as IIP	IIP for 2 nd Order
IIP3	3 rd Order Input	**iipn**(IIP, 3)	Same as IIP	IIP for 3 rd Order

	Intercept Point			
IP1DB	*Input 1 dB Compression Point* (sim)	**inref**(SIP1DB, CGAIN)	Same as CGAIN	1 / IP1dB = 1 / SIP1DB1 + 1 / (SIP1DB2 - CGAIN[1]) + 1 / (SIP1DB3 - CGAIN[2]) + ... + 1 / (SIP1DBX - CGAIN[X-1]), where X is the nth stage dBm
IPSAT	*Input Saturation Point* (sim)	**inref**(SIPSAT, CGAIN)	Same as CGAIN	I1 / IPSAT = 1 / SIPSAT1 + 1 / (SIPSAT2 - CGAIN[1]) + 1 / (SIPSAT3 - CGAIN[2]) + ... + 1 / (SIPSATX - CGAIN[X-1]), where X is the nth stage dBm
IMGNP	*Image Channel Noise Power* (sim)	IMGNP	**Noise**	Noise power at IMGF
IMGP	*Image Channel Power* (sim)	IMGP	**Total**	Total power at IMGF
IMGNR	*Image Channel Noise Rejection* (sim)	**imgnr**(CNP, IMGNP)	Same as CNP & IMGNP	IMGNR[i] = CNP[i] - IMGNP[i]
IMGR	*Image Channel Rejection* (sim)	**imgr**(DCP, IMGP)	Same as DCP & IMGP	IMGR[i] = DCP[i] - IMGP[i]
MDS	*Minimum Detectable Signal* (sim)	**mds**(CNP, CNF)	Same as CNP & CNF	MDS[i] = CNP[0] + CNF[i]
MML	*Source Mismatch Loss* (sim)		Same as DCP	Loss due to source and system input impedance mismatch
NDCP	*Noise and Distortion Channel Power* (sim)	**ndcp**(CNP, TIMP, PNCP)	Same as PNCP, CNP, & TIMP	NDCP[i] = PNCP[i] + CNP[i] + TIMP[i]
OCP	*Channel Power (Offset)* (sim)	OCP	**Total**	Total power at OCF
OIP	*Output Intercept Point* (sim) (All Orders)	**oip**(ICP, DELTA)	Same as ICP	OIP[i] = ICP[i] + Delta[i] / (Order-1) Delta[i] = ICP[i] TIMCP[i]
OIP2	2 nd Order Output Intercept Point	**oipn**(OIP, 2)	Same as OIP	OIP for 2 nd Order
OIP3	3 rd Order Output Intercept Point	**oipn**(OIP, 3)	Same as OIP	OIP for 3 rd Order
OP1DB	*Output 1 dB Compression Point* (sim)	**outref**(SOP1DB, GAIN)	Same as GAIN	1 / OP1dB = 1 / (SOP1dB1 + Gain2 ... + GainX) + 1 / (SOP1dB2 + Gain3 ... + GainX) + ... + 1 / SOP1dBX dBm, where X is the nth stage
OPSAT	*Output Saturation Point* (sim)	**outref**(SOPSAT, GAIN)	Same as GAIN	1 / OPSAT = 1 / (SOPSAT1 + Gain2 ... + GainX) + 1 / (SOPSAT2 + Gain3 ... + GainX) + ... + 1 / SOPSATX dBm, where X is the nth stage
PNCP	*Phase Noise Channel Power* (sim)	PNCP	**Phase Noise**	Phase noise power at CF
PRNF	*Percent Noise Figure* (sim)	PRNF	Same as AN & CNF	PRNF[i] = AN[i]/CNF[iLastStage]
PRIM	*Percent Intermods* (sim) (All Orders)	**prim**(GIMCP, ICGAIN, TIMCP)	Same as GIMP, CGAIN, & TIMP	IMREF = GIMCP[i] + (CGAIN[iLastStage] - CGAIN[i]) PRIM[i] = IMREF[i]/TIMP[iLastStage](this is a ratio in Watts), Where PRIM[0] = 0 %
PRIM2	Percent 2 nd Order Intermods	**primn**(PRIM, 2)	Same as PRIM	See PRIM
PRIM3	Percent 3 rd Order Intermods	**primn**(PRIM, 3)	Same as PRIM	See PRIM
RX_IIP	*Input Intercept Point (Receiver)* (sim) (All Orders)	**rx_iip**(RX_OIP, CGAIN)	Same as RX_OIP & CGAIN	RX_IIP[i] = RX_OIP[i] - CGAIN[i], where RX_OIP and IIP contain all orders
RX_IIP2	Receiver 2 nd Order Input Intercept Point	**iipn**(RX_IIP, 2)	Same as RX_IIP	RX_IIP for 2 nd Order
RX_IIP3	Receiver 3 rd Order Input Intercept Point	**iipn**(RX_IIP, 3)	Same as RX_IIP	RX_IIP for 3 rd Order
RX_SFDR	*Receiver Spurious Free Dynamic Range* (sim)	RX_SFDR	Same as RX_IIP3 & MDS	RX_SFDR[i] = 2/3 [RX_IIP3[i] - MDS[i]]
SFDR	*Spurious Free Dynamic Range* (sim)	SFDR	Same as IIP3 & MDS	SFDR[i] = 2/3 [IIP3[i] - MDS[i]]
SDR	*Stage Dynamic Range* (sim)	SDR	Same as TNP	SDR[i] = SOP1DB[i] - TNP[i]
SGAIN	*Stage Gain* (sim)	SGAIN	None	Stage entered value
SIP1DB	*Stage Input 1 dB Compression Point* (sim)	**sip1db**(SOP1DB, SGAIN)	Same as SOP1DB & SGAIN	SIP1DB[i] = SOP1DB[i] SGAIN[i]
SIIP	*Stage Input Intercept Point* (sim) (All Orders)	**siip**(SOIP, SGAIN, SZIN, SZOUT)	Same as SOIP & SGAIN	SIIP[i] = SOIP[i] SGAIN[i]
SIIP2	Stage 2 nd Order Input Intercept Point	**sipn**(X, 2)	None	SIIP for 2 nd Order
SIIP3	Stage 3 rd Order Input Intercept Point	**sipn**(X, 3)	None	SIIP for 3 rd Order
SIPSAT	*Stage Input Saturation Point* (sim)	**sipsat**(SOPSAT, SGAIN, SZIN, SZOUT)	Same as SOPSAT & SGAIN	SIPSAT[i] = SOPSAT[i] SGAIN[i]
SNF	*Stage Noise Figure* (sim)	SNF	None	Stage entered value
SOP1dB	*Stage Output 1 dB Compression Point* (sim)	SOP1dB	None	Stage entered value
SOIP	*Stage Output Intercept Point* (sim) (All Orders)	SOIP	None	Stage entered value
SOIP2	Stage 2 nd Order Output Intercept Point	SOIP2	None	Stage entered value
SOIP3	Stage 3 rd Order Output Intercept Point	SOIP3	None	Stage entered value
SOPSAT	*Stage Output*	SOPSAT	None	Stage entered value

	Saturation Point (sim)			
SZIN	*Stage Input Impedance* (sim)	SZIN	None	Stage entered value
SZOUT	*Stage Output Impedance* (sim)	SZOUT	None	Stage entered value
TIMCP	*Intermod Channel Power (Total)* (sim) (All Orders)	TIMCP	**Total Intermod Order N**	Total intermod power for all orders at CF
TIMCP2	Total 2 nd Order Intermod Channel Power	**timcpn**(TIMCP, 2)	Same as TIMCP	TIMCP for 2 nd Order
TIMCP3	Total 3 rd Order Intermod Channel Power	**timcpn**(TIMCP, 3)	Same as TIMCP	TIMCP for 3 rd Order
TIMP	*Total Intermod Power* (sim)	TIMP	**Total Intermod**	Total intermod power at CF
TNP	*Total Node Power* (sim)	TNP	**Total**	Power of entire spectrum at node i
VTCP	*Virtual Tone Channel Power* (sim)	VTCP	Same as ICP and CGAIN	ICP[0] + CGAIN[i]

Spectrasys Voltage Measurements

Name	Description	Syntax	Spectrum Type	Equation
CV	*Channel Voltage* (sim)	CV	**Total**	Peak voltage at CF
CGAINV	*Cascaded Voltage Gain* (sim)	**cgain**(DCV)	Same as DCV	CGAINV[i] = DCV[i] DCV[0]
CNRV	*Carrier to Noise Voltage Ratio* (sim)	**cnr**(DCV, CNV)	Same as DCV and CNV	CNRV[i] = DCV[i] - CNV[i]
CNV	*Channel Noise Voltage* (sim)	CNV	**Noise**	Peak noise voltage in the channel
DCV	*Channel Voltage (Desired)* (sim)	DCV	**Desired**	Desired peak voltage at CF
GAINV	*Voltage Gain* (sim)	**gain**(DCV)	Same as DCV	GAINV[i] = DCV[i] DCV[i-1], Where GAINV[0] = 0 dB
ICV	*Interferer Channel Voltage* (sim)	ICV	**Desired**	Peak interferer voltage at ICF
NNV	*Node Noise Voltage* (sim)	NNV	**Noise**	Peak average noise voltage at the node
OCV	*Offset Channel Voltage* (sim)	OCV	**Total**	Peak voltage at OCF
PNCV	*Phase Noise Channel Voltage* (sim)	PNCV	**Phase Noise**	Peak phase noise voltage at CF
SIIV	*Stage Input Voltage Intercept Point* (sim) (All Orders)	**siiv**(SOIV, SVGAIN)	Same as SOIV and SVGAIN	SIIV[i] = SOIV[i] - SVGAIN[i]
SIIV2	Stage 2 nd Order Input Voltage Intercept Point	**sivn**(SIIV, 2)	None	SIIV for 2 nd Order
SIIV3	Stage 3 rd Order Input Voltage Intercept Point	**sivn**(SIIV, 2)	None	SIIV for 3 rd Order
SIV1DB	*Stage Input 1 dB Voltage Compression Point* (sim)	siv1db(SOV1DB, SVGAIN)	Same as SOV1DB and SVGAIN	SIV1DB[i] = SOV1DB[i] - SVGAIN[i]
SIVSAT	*Stage Input Voltage Saturation Point* (sim)	sivsat(SOVSAT, SVGAIN)	Same as SOVSAT and SVGAIN	SIVSAT[i] = SOVSAT[i] - SVGAIN[i]
SOIV	*Stage Output Voltage Intercept Point* (sim) (All Orders)	SOIV	None	Stage Entered Intercept Point
SOIV2	Stage 2 nd Order Output Voltage Intercept Point	**sivn**(SOIV, 2)	None	Stage Entered Intercept Point
SOIV3	Stage 3 rd Order Input Voltage Intercept Point	**sivn**(SOIV, 3)	None	Stage Entered Intercept Point
SOV1DB	*Stage Output 1 dB Voltage Compression Point* (sim)	SOV1dB	None	Stage Entered Compression Point
SOVSAT	*Stage Output Voltage Saturation Point* (sim)	SOVSAT	None	Stage Entered Output Saturation Voltage
SVGAIN	*Stage Voltage Gain* (sim)	SVGAIN	None	Stage Entered Voltage Gain
SVNI	*Equivalent Input Noise Voltage* (sim)	SVNI	None	Conversion from stage entered noise figure
TNV	*Total Node Voltage* (sim)	TNV	**Total**	Peak voltage at the node
VDC	*DC Voltage* (sim)	VDC	**Total**	DC voltage
VNI	*Equivalent Input Noise Voltage* (sim)	**NFtoVNI**(CNF, DCR, Temperature)	Same as CNF and DCR	VNI[i] = NFtoVNI(CNF[i], DCR[i], Temperature)

Spectrasys Equation Based Measurements

(These measurements use traditional cascaded equations based on user entered data)

Name	Description	Syntax	Spectrum Type	Equation
ECGAIN	*Cascaded Gain* (sim)	ECGAIN	None	Traditional cascaded gain equation
ECNF	*Cascaded Noise Figure* (sim)	ECNF	None	Traditional cascaded noise figure equation
EIIP2	*2 nd Order Input Intercept Point* (sim)	EIIP2	None	Traditional cascaded input 2nd order intercept
EIIP3	*3 rd Order Input Intercept Point* (sim)	EIIP3	None	Traditional cascaded input 3rd order intercept
EIP1DB	*Input 1 dB Compression Point* (sim)	EIP1DB	None	Traditional cascaded input compression point
EIPSAT	*Cascaded Input Saturation Poin* (sim)t	EIPSAT	None	Traditional cascaded input saturation point
EMDS	*Minimum Detectable Signal* (sim)	EMDS	None	Traditional minimum detectable signal
EOIP2	*2 nd Order Output Intercept Point* (sim)	EOIP2	None	Traditional cascaded output 2nd order intercept point
EOIP3	*3 rd Order Output Intercept Point* (sim)	EOIP3	None	Traditional cascaded output 3rd order intercept point
EOP1DB	*Output 1 dB Compression Point* (sim)	EOP1DB	None	Traditional cascaded output compression point
EOPSAT	*Cascaded Output Saturation Point* (sim)	EOPSAT	None	Traditional cascaded output saturation point
ESFDR	*Spurious Free Dynamic Range* (sim)	ESFDR	None	Traditional cascaded spurious free dynamic range

Caution: Equation based measurements exclude all frequency, VSWR, and compression effects.

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

S Parameters
Equation based measurements use stage parameters directly entered by the user like gain, noise figure, intercept point, etc. If the model doesn't allow the user to enter one of these parameters, like the S parameter model, then equation based measurements will ignore these parameters.

Node Measurements

Frequency, Voltage, Power, and Identification information is saved for selected output nodes. See the '*Output Tab* (sim)' of the system simulation dialog box reference for additional information on controlling which nodes to retain the data.

Name	Description	Units
RFPwrIn	*Total RF Power Entering a Part* (sim)	dBm

Added Noise (AN)

This measurement is the noise contribution of each individual stage in the main channel along the specified path as shown by:

AN[n] = *CNF* (sim)[n] - CNF[n-1] (dB), where AN[0] = 0 dB, n = stage number

This measurement is simply the difference in the 'Cascaded Noise Figure' measurement between the current node and the previous node. This measurement is very useful and will help the user identify the contribution to the noise figure by each stage along the path.

Channel Used: Main Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: Same as CNF

Travel Direction: Only spectrum traveling in the forward direction are included in this measurement

Adjacent Channel Frequency (ACF[U or L][n])

This measurement is the frequency of the specified adjacent channel. All adjacent channel frequencies are relative to the main 'Channel Frequency'. Consequently, channels exist above and below the main reference channel frequency. The user can specify which side of the main or reference channel that the adjacent channel is located on and also the channel number. The channel number is relative to the main or reference channel. Therefore, channel 1 would be the first adjacent channel, channel 2 would be the second adjacent channel, and so on.

U - Upper Side

L - Lower Side

n - Channel Number (any integer > 0)

For example, ACFU1 if the first adjacent channel above that specified by the 'Channel Frequency'. If CF was 100 MHz and the channel bandwidth was 1 MHz then the main channel would be 99.5 to 100.5 MHz. Consequently, then ACFU1 would then be the channel 100.5 to 101.5 MHz and ACFL1 would be 98.5 to 99.5 MHz.

NOTE: Only the first few adjacent channels on either side of the reference channel are listed in the 'Measurement Wizard". However, there is no restriction on the Adjacent Channel Number.

Adjacent Channel Power (ACP[U or L)[n))

This measurement is the integrated power of the specified adjacent channel. All adjacent channels are relative to the main channel (identified by the 'Channel Frequency' and 'Channel Measurement Bandwidth'). Consequently, channels exist above and below the main reference channel frequency. The user can specify which side of the main channel the adjacent channel is located on along with the channel number. The channel number is relative to the main channel. Therefore, channel 1 would be the first adjacent channel, channel 2 would be the second adjacent channel, and so on.

U - Upper Side

L - Lower Side

n - Channel Number (any integer > 0)

For example, ACPL2 is the power of the second adjacent channel below that specified by the channel frequency. If CF was 100 MHz and the channel bandwidth was 1 MHz then the main channel would be 99.5 to 100.5 MHz. Consequently, then ACPL2 would then be the integrated channel power between 97.5 and 98.5 MHz and ACPL1 would be the integrated channel power between 98.5 and 99.5 MHz.

Note: Only the first 2 adjacent channels on either side of the reference channel are listed in the 'Measurement Wizard". However, there is no restriction on the Adjacent Channel Number other than it must be non-negative and greater than or equal to 1.

This measurement is simply a '*Channel Power* (sim)' measurement at the '*Adjacent Channel Frequency* (sim)'.

Channel Used: Corresponding Adjacent Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: Same as CP

Travel Direction: Same as CP

Calculated Stage Compression (COMP)

This measurement is the calculated compression point of each individual stage. This value is determined during the simulation process based on the total input power of the stage.

This measurement includes **ALL SIGNALS**, **INTERMODS**, **HARMONICS**, **NOISE**, and **PHASE NOISE** traveling in **ALL** directions through the node that fall within the main channel.

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Channel Used: No channel is used for this measurement

Types of Spectrums Used: *All SIGNALS, INTERMODS, HARMONICS, NOISE, and PHASE NOISE*

Travel Direction: All directions through the node

Carrier to Noise and Distortion Ratio (CNDR)

This measurement is the ratio of the 'Desired Channel Power' to 'Channel Noise and Distortion Power' along the specified path as shown by:

CNR[n] = *DCP* (sim)[n] - *NDCP* (sim)[n] (dB), where n = stage number

Channel Used: Main Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: Same as DCP and NDCP

Travel Direction: Same as DCP and NDCP

Carrier to Noise Ratio (CNR)

This measurement is the ratio of the 'Desired Channel Power' to 'Channel Noise Power' along the specified path as shown by:

CNR[n] = *DCP* (sim)[n] - *CNP* (sim)[n] (dB), where n = stage number

Channel Used: Main Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: Same as DCP and CNP

Travel Direction: Same as DCP and CNP

Carrier to Noise Voltage Ratio (CNRV)

This measurement is the ratio of the 'Desired Channel Voltage' to 'Channel Noise Voltage' along the specified path as shown by:

CNRV[n] = *DCV* (sim)[n] - *CNV* (sim)[n] (dB), where n = stage number

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Channel Used: Main Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: Same as DCV and CNV

Travel Direction: Same as DCV and CNV

Cascaded Compression (CCOMP)

This measurement is the cascaded compression of each stage along the path.

CCOMP[n] = Summation(*COMP* (sim)[0 to n]) (dB), where n = stage number

For each stage n a summation is performed on the compression point of all previous stages.

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Channel Used: Same as COMP

Types of Spectrums Used: Same as COMP

Travel Direction: Same as COMP

Cascaded Gain (CGAIN)

This measurement is the cascaded gain of the main channel along the specified path. The 'Cascaded Gain' is the difference between the 'Desired Channel Power' measurement at the nth stage minus the 'Desired Channel Power' measurement at the input as shown by:

CGAIN[n] = *DCP* (sim)[n] - DCP[0] (dB), where n = stage number

Channel Used: Main Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: Same as DCP

Travel Direction: Same as DCP

NOTE: Under matched conditions CGAIN and S21 from a linear analysis are the same. As shown in the above equation the cascaded gain at the first node by definition is 0 dB. This may not be true if there is an impedance mismatch between the source and the first model in the path. The cascaded gain measurement does not take into account this initial mismatch because cascaded gain is always assumed to be 0 dB at the first stage. This mismatch can be accounted for by taking the difference between the power level specified in the source with the 'Channel Power (CP)' at the first node and adding this value to the cascaded gain. In this case cascaded gain _ source mismatch will equal S21.

Cascaded Gain - Equation Based (ECGAIN)

This is the traditional cascaded gain measurement based on the ***user entered gain values for the stages*** (*SGAIN* (sim)). This measurement can be used to compare with traditional calculations typically found in spreadsheets.

ECGAIN[n] = SGAIN[0] + ... + SGAIN[n] (dB), where n = stage number

Caution: This measurement excludes all frequency, VSWR, and compression effects.

See *CGAIN* (sim) as the general cascaded gain measurement that includes all secondary effects.

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Channel Used: None

Types of Spectrums Used: None

Travel Direction: N/A

S Parameters
Equation based measurements use stage parameters directly entered by the user like gain, noise figure, intercept point, etc. If the model doesn't allow the user to enter one of these parameters, like the S parameter model, then equation based measurements will ignore these parameters.

Cascaded Noise Figure (CNF)

This measurement is the cascaded noise figure in the main channel along the specified path. The 'Cascaded Noise Figure' is equal to the 'Channel Noise Power' measurement at the output of stage n plus the 'Phase Noise Channel Power' at the output of stage n minus the 'Channel Noise Power' measurement at the path input minus the 'Cascaded Gain' measurement at stage n as shown by:

CNF[n] = *CNP* (sim)[n] + *PNCP* (sim)[n] - CNP[0] - *cgain(DCP)* (sim)[n] (dB), where n = stage number

Caution: When wide channel bandwidths are used channel noise power and cascaded gain are affected more by VSWR and frequency effects. In this case is it extremely important that sufficient noise points are used to represent the noise in the channel of interest. Furthermore, it is very possible because of these frequency effects that the channel noise power and the cascaded gain can change in a nonlinear way so that cascaded noise figure appears to drop from a prior node. Additionally, looking at cascaded noise figure through a hybrid combining network may also be deceptive since the cascaded gain used to determine the cascaded noise figure is from the current path and not all paths in the system.

See the '*Broadband Noise* (sim)' and '*Two Port Amplifier Noise* (sim)' sections for more information.

Channel Used: Main Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: Same as CNP and CGAIN

Travel Direction: Same as CNP and CGAIN

Traditional Cascaded Noise Figure

NFcascade = F1 + (F2 - 1) / G1 + (F3 - 1) / G1G2 + ... + (Fn - 1) / G1G2...Gn-1

Note: Traditional cascaded noise figure equations are not used. The are very restrictive and suffer from the following conditions:

- Ignore effects of VSWR, frequency, and bandwidth
- Assume noise contributions are from a single path
- Ignore mixer image noise
- Ignores effects of gain compression
- Ignores contribution of phase noise

The SPARCA (Spectral Propagation and Root Cause Analysis) technique used in Spectrasys does not suffer from these restrictive assumptions. Occasionally users are troubled if Spectrasys simulations give different answers than the traditional approach. Using Spectrasys under the same assumptions as the traditional approach will always yield identical answers. To use Spectrasys under the same assumptions only frequency independent blocks like attenuators and amplifiers must be used. Even so, amplifiers must be used as linear devices with infinite reverse isolation. Filters cannot be used since their impedance varies with frequency. Mixers cannot be used since noise from the image band will be converted into the mixer output. Only 2 port devices must be used because of the single path assumption. Of course the bandwidth must also be very narrow.

Cascaded Noise Figure - Equation Based (ECNF)

This is the traditional cascaded noise figure measurement based on the ***user entered gain and noise figure values for the stages*** (*SGAIN* (sim) and *SNF* (sim)). This measurement can be used to compare with traditional calculations typically found in spreadsheets.

Traditional Cascaded Noise Figure

NFcascade = F1 + (F2 - 1) / G1 + (F3 - 1) / G1G2 + ... + (Fn - 1) / G1G2...Gn-1

Caution: This measurement (and the traditional cascaded noise figure equation) has the following issues:

- Ignore effects of VSWR, frequency, and bandwidth
- Assume noise contributions are from a single path
- Ignore mixer image noise
- Ignores effects of gain compression
- Ignores noise contribution due to phase noise

The SPARCA (Spectral Propagation and Root Cause Analysis) technique used in Spectrasys does not suffer from these restrictive assumptions. Occasionally users are troubled if Spectrasys simulations give different answers than the traditional approach. Using Spectrasys under the same assumptions as the traditional approach will always yield identical answers. To use Spectrasys under the same assumptions only frequency independent blocks like attenuators and amplifiers must be used. Even so, amplifiers must be used as linear devices with infinite reverse isolation. Filters cannot be used since their impedance varies with frequency. Mixers cannot be used since noise from the image band will be converted into the mixer output. Only 2 port devices must be used because of the single path assumption. Of course the bandwidth must also be very narrow.

See *CNF* (sim) as the general cascaded noise figure measurement that includes all secondary effects.

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Channel Used: None

Types of Spectrums Used: None

Travel Direction: N/A

S Parameters
Equation based measurements use stage parameters directly entered by the user like gain, noise figure, intercept point, etc. If the model doesn't allow the user to enter one of these parameters, like the S parameter model, then equation based measurements will ignore these parameters.

Cascaded Voltage Gain (CGAINV)

This measurement is the cascaded voltage gain of the main channel along the specified path. The 'Cascaded Voltage Gain' is the difference between the 'Desired Channel Voltage' measurement at the nth stage minus the 'Desired Channel Voltage' measurement at the input as shown by:

CGAINV[n] = *DCV* (sim)[n] - DCV[0] (dB20), where n = stage number

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Channel Used: Main Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: Same as DCV

Travel Direction: Same as DCV

Channel Noise Density (CND)

This measurement is the integrated noise power in a 1 Hz bandwidth in the main channel along the specified path.

Note: The power level of the first node is that seen by the internal source not the actual power delivered to the first stage. As such any impedance mismatch between the source and the system input is automatically accounted for.

This measurement is the same as the Channel Noise Power (CNP) except a 1 Hz bandwidth is used.

Channel Used: Main Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: *ONLY NOISE*

Travel Direction: Only spectrums traveling in the FORWARD path direction

Channel Bandwidth Caution: When the Channel Frequency is less than 1/2 the Channel Bandwidth the lowest integration frequency used for measurements will be 0 Hz. This will result is Channel Noise Power measurements potentially being different than when the full bandwidth is used.

Channel Noise Power (CNP)

This measurement is the integrated noise power in the main channel along the specified path.

Note: The power level of the first node is that seen by the internal source not the actual power delivered to the first stage. As such any impedance mismatch between the source and the system input is automatically accounted for.

For example, if the 'Channel Measurement Bandwidth' was specified to 100 kHz and the 'Channel Frequency' was 2000 MHz then the CNP is the integrated noise power from 1999.95 to 2000.05 MHz.

See comments in the 'Cascaded Noise Figure' measurement or 'Broadband Noise' section for additional insights.

Channel Used: Main Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: *ONLY NOISE*

Travel Direction: Only spectrums traveling in the FORWARD path direction

Channel Bandwidth Caution: When the Channel Frequency is less than 1/2 the Channel

Bandwidth the lowest integration frequency used for measurements will be 0 Hz. This will result is Channel Noise Power measurements potentially being different than when the full bandwidth is used.

Channel Noise Voltage (CNV)

This measurement is the peak noise voltage in the main channel along the specified path.

For example, if the 'Channel Measurement Bandwidth' was specified to 100 kHz and the 'Channel Frequency' was 2000 MHz then the CNV is the average integrated noise voltage from 1999.95 to 2000.05 MHz.

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Default Unit: dBV

Channel Used: Main Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: *ONLY NOISE*

Travel Direction: Only spectrums traveling in the FORWARD path direction

Channel (or Path) Frequency (CF)

Since each spectrum can contain a large number of spectral components and frequencies Spectrasys must be able to determine the area of the spectrum to integrate for various measurements. This integration area is defined by a 'Channel Frequency' and a 'Channel Measurement Bandwidth' which become the main channel for the specified path. Spectrasys can automatically identify the desired 'Channel Frequency' in an unambiguous case where only one frequency is on the 'from node' of the designated path. An error will appear if more than one frequency is available. For this particular case the user must specify the intended frequency for this path in the 'System Simulation Dialog Box'.

A 'Channel Frequency' exists for each node along the specified path. Consequently, each node along the path will have the same 'Channel Frequency' until a frequency translation part such as a mixer or frequency multiplier is encountered. Spectrasys automatically deals with frequency translation through these parts. The individual mixer parameters of 'Desired Output (Sum or Difference)' and 'LO Injection (High of Low)' are used to determine the desired frequency at the output of the mixer.

The 'Channel Frequency' is a critical parameter for Spectrasys since most of the measurements are based on this parameter. If this frequency is incorrectly specified then the user may get unexpected results since many measurements are based on this frequency.

The easiest way to verify the 'Channel Frequency' that Spectrasys is using is to look at this measurement in a table or the dataset.

Channel Power (CP)

This measurement is the total integrated power in the main channel along the specified path.

This measurement includes **ALL SIGNALS**, **INTERMODS**, **HARMONICS**, **NOISE**, and **PHASE NOISE** traveling in **ALL** directions through the node that fall within the main channel.

Note: The power level of the first node is that seen by the internal source not the actual power delivered to the first stage. As such any impedance mismatch between the source and the system input is automatically accounted for.

For example, if the 'Channel Measurement Bandwidth' was specified to .03 MHz and the 'Channel Frequency' was 220 MHz then the CP is the integrated power from 219.985 to 220.015 MHz.

Channel Used: Main Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: *All SIGNALS, INTERMODS, HARMONICS, NOISE, and PHASE NOISE*

Travel Direction: All directions through the node

Channel Voltage (CV)

This measurement is the peak voltage across the main channel along the specified path.

This measurement includes **ALL SIGNALS**, **INTERMODS**, **HARMONICS**, **NOISE**, and **PHASE NOISE** traveling in **ALL** directions through the node that fall within the main channel.

For example, if the 'Channel Measurement Bandwidth' was specified to .03 MHz and the 'Channel Frequency' was 220 MHz then the CV is the average voltage from 219.985 to 220.015 MHz.

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Default Unit: dBV

Channel Used: Main Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: *All SIGNALS, INTERMODS, HARMONICS, NOISE, and PHASE NOISE*

Travel Direction: All directions through the node

Conducted Intermod Channel Power [All Orders] (CIMCP)

This measurement is the total intermod power in the main channel conducted from the prior stage. This measurement includes all intermods that are traveling in the forward path direction. In equation for the conducted third order intermod power is:

CIMCP[n] = *TIMCP* (sim)[n-1] + *GAIN* (sim)[n] (dBm), where CIMCP[0] = 0 dB and n = stage number

Each column in this measurement is for a different intermod order up to the Maximum Order specified on the '*Calculate Tab* (sim)' of the System Analysis Dialog Box. The column number is the same as the order starting from the left with order 0.

Remember intermod bandwidth is a function of the governing intermod equation. For example, if the intermod equation is 2F1 - F2 then the intermod bandwidth would be: 2BW1 + BW2. Note: Bandwidths never subtract and will always add. The channel bandwidth must be set wide enough to include the entire bandwidth of the intermod to achieve the expected results.

Channel Used: Main Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: Same as TIMCP and GAIN

Travel Direction: Same as TIMCP and GAIN

Desired Channel Phase (DCPH)

This measurement is the phase of the **desired CW signal** in the channel along the specified path.

This measurement includes **ONLY DESIRED SIGNALS** on the beginning node of the path, traveling in **FORWARD** path direction. All other intermods, harmonics, noise, and phase noise signals are ignored.

Note: A 'D' is placed next to the equation in the identifying flyover help in a spectrum plot to indicate desired signals.

Channel Used: Main Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: *ONLY DESIRED SIGNALS*

Travel Direction: Only in the FORWARD direction

Desired Channel Power (DCP)

This measurement is the total integrated power in the main channel along the specified path.

This measurement includes **ONLY DESIRED SIGNALS** on the beginning node of the path, traveling in the **FORWARD** path direction. All other intermods, harmonics, noise, and phase noise signals are ignored.

Note: The power level of the first node is that seen by the internal source not the actual power delivered to the first stage. As such any impedance mismatch between the source and the system input is automatically accounted for.

Note: A 'D' is placed next to the equation in the identifying flyover help in a spectrum plot to indicate desired signals.

For example, if the 'Channel Measurement Bandwidth' was specified to .03 MHz and the 'Channel Frequency' was 220 MHz then the DCP is the integrated power from 219.985 to 220.015 MHz. This power measurement will not even be affect by another 220 MHz signal traveling in the reverse direction even if it is much larger in amplitude.

Channel Used: Main Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: *ONLY DESIRED SIGNALS*

Travel Direction: Only in the FORWARD direction

Desired Channel Resistance (DCR)

This measurement is the desired average resistance across the main channel along the specified path. This is not the magnitude of the impedance but its real part.

This measurement includes **ONLY DESIRED SIGNALS** on the beginning node of the path, traveling in **FORWARD** path direction. All other intermods, harmonics, noise, and phase noise signals are ignored.

Note: A 'D' is placed next to the equation in the identifying flyover help in a spectrum plot to indicate desired signals.

For example, if the 'Channel Measurement Bandwidth' was specified to .03 MHz and the 'Channel Frequency' was 220 MHz then the DCR is the average resistance from 219.985 to 220.015 MHz. This resistance measurement will not even be affect by another 220 MHz signal traveling in the reverse direction even if it is much larger in amplitude.

Channel Used: Main Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: *ONLY DESIRED SIGNALS*

Travel Direction: Only in the FORWARD direction

Desired Channel Voltage (DCV)

This measurement is the desired peak voltage across the main channel along the specified path.

This measurement includes **ONLY DESIRED SIGNALS** on the beginning node of the path, traveling in **FORWARD** path direction. All other intermods, harmonics, noise, and phase noise signals are ignored.

Note: A 'D' is placed next to the equation in the identifying flyover help in a spectrum plot to indicate desired signals.

For example, if the 'Channel Measurement Bandwidth' was specified to .03 MHz and the 'Channel Frequency' was 220 MHz then the DCV is the average voltage from 219.985 to 220.015 MHz. This voltage measurement will not even be affect by another 220 MHz signal traveling in the reverse direction even if it is much larger in amplitude.

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Default Unit: dBV

Channel Used: Main Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: *ONLY DESIRED SIGNALS*

Travel Direction: Only in the FORWARD direction

Equivalent Input Noise Voltage (VNI)

This measurement is the stage equivalent input noise voltage. Noise figure, source resistance, and temperature are used to determine equivalent input noise voltage. The following equation is used to take the noise factor at each stage, the desired channel resistance at each stage, and the system analysis temperature in Celsius and converts these parameters to an equivalent input noise voltage.

VNI[n] = NFtoVNI (*CNF* (sim)[n], *DCR* (sim)[n], TempC) (nV / sqrt(Hz)), where n = stage number

See *SVNI* (sim) for additional information.

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Default Unit: nV / sqrt(Hz)

Channel Used: No channel is used for this measurement

Types of Spectrums Used: None

Travel Direction: N/A

Generated Intermod Channel Power [All Orders] (GIMCP)

This measurement is the generated intermod power in the main channel created at the output of the current stage. In equation form the generated third order intermod power is:

GIMCP[n] = integration of the intermods generated at stage n across the channel bandwidth (dBm)

Each column in this measurement is for a different intermod order up to the Maximum Order specified on the '*Calculate Tab* (sim)' of the System Analysis Dialog Box. The column number is the same as the order starting from the left with order 0.

Remember intermod bandwidth is a function of the governing intermod equation. For example, if the intermod equation is 2F1 - F2 then the intermod bandwidth would be: 2BW1 + BW2. Note: Bandwidths never subtract and will always add. The channel bandwidth must be set wide enough to include the entire bandwidth of the intermod to achieve the expected results. The 'Automatic Intermod Mode' will set the bandwidth appropriately.

Channel Used: Main Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: *ONLY INTERMODS and HARMONICS (separated according to their order)*

Travel Direction: Only in the FORWARD direction

Group Delay (System)

This section will demonstrate how to measure group delay in Spectrasys.

BACKGROUND

"Group Delay measures the differential time delay caused by a filter, i.e., it indicates if certain frequency components will be delayed more than other components and by how much" (Radio Receiver Design, Kevin McClaning and Tom Vito, Noble Publishing, pg 174)

Spectrasys has the ability to show group delay results, including across frequency translations, like mixers and dividers, by using a wide carrier source and the linear simulator. The group delay measurement function will then be applied to any path voltage gathered by Spectrasys.

The path voltage in Spectrasys is equivalent to the voltage gain E21. To convert E21 to S21 for group delay calculation the following equation must be used:

Sij = Eij * (conj(ZPortj) + ZPortj * Sjj) / (ZPorti * sqrt(real(ZPortj) / real(ZPorti))), where ZPort is the port impedance the circuit sees.

For ZPorti = ZPortj = R + j0

Then

Sij = Eij * (1 + Sjj)

or

S21 = E21 * (1 + S11)

Since Spectrasys doesn't have an S11 measurement the linear simulator must be used. A frequency vector can be retrieved from the Spectrasys path dataset so that the frequencies gathered by the linear simulator are the same as those used in Spectrasys.

STEPS FOR GROUP DELAY MEASUREMENT

1. Add a wideband signal source. The bandwidth should be wide enough to cover the group delay range of interest. Caution must be exercised when setting the carrier power since a wideband carrier can easily drive non-linear devices into saturation. Note: A multisource can be used here and the number of points for the wideband carrier can be specified per source.
2. Increase the number of simulation points for the carrier on the 'General' tab of the System Analysis. The default number of simulation points per carrier is 2. Note: This will increase the number of simulation points used by all spectrums derived from the

source spectrum and will increase the simulation time and file size.
3. Add a path to the System Analysis on the 'Path' tab.
4. Include option "Add Powers, Voltages, and Impedances to Path Dataset" located on the 'Edit/Add Path' dialog box of the System Analysis.
5. Add a linear analysis so the S11 of the system can be determined.
6. Place the Spectrasys path frequency vector (i.e. 'F2_Desired' in the 'System1_Data_Path1' dataset) in the 'List of Frequencies' used by the linear analysis so S11 will be determined at the same frequencies as the Spectrasys path voltages.
7. S21 for the system will need to derived from the equation relating E21 to S21, where the desired Spectrasys path voltage at the output node is equivalent to E21. Copy the following equation block code to the equation block in the workspace. This code assumes the output node is named "2". The path frequency vector in the linear analysis and the voltage vector in the equation block must be change if the output node is other than 2 (i.e if the output node was 6 then 'F6_Desired' would be used in the linear analysis and 'V6_Desired' would be used in the equation block.
8. Use the group delay function 'gd(SystemS21)' to plot the group delay.

Equation Block Code

```
using( "System1_Data_Path1" )

'---------- Convert Spectrasys E21 (V2_Desired) to S21 ----------

SystemS21 = V2_Desired * ( 1 + [Linear1_Data].S11 )

setunits( "SystemS21", "dB" )     ' Set the units to dB

GroupDelay = gd( SystemS21 )   ' Get the group delay
```

NOTE: See Spectrasys group delay examples that ship with the product for illustration of this.

Image Channel Noise Power (IMGNP)

This measurement is the integrated noise power of the image channel from the path input to the first mixer. After the first mixer the 'Mixer Image Channel Power' measurement will show the same noise power as the main channel noise power.

For example if we designed a 2 GHz receiver that had an IF frequency of 150 MHz using low LO side injection then the LO frequency would be 1850 MHz and image frequency for all stages from the input to the first mixer would be 1700 MHz. If the receiver bandwidth was 5 MHz then the image channel would be from 1697.5 to 1702.5 MHz.

This measurement is simply a '*Channel Noise Power* (sim)' measurement at the '*Image Frequency* (sim)'.

Channel Used: Image Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: *ONLY NOISE*

Travel Direction: Only spectrums traveling in the FORWARD path direction

Image Channel Power (IMGP)

This measurement is the image channel power from the path input to the first mixer. After the first mixer the this measurement will show the same power as the main channel power.

For example if we designed a 2 GHz receiver that had an IF frequency of 150 MHz using low LO side injection then the LO frequency would be 1850 MHz and image frequency for all stages from the input to the first mixer would be 1700 MHz. If the receiver bandwidth was 5 MHz then the image channel would be from 1697.5 to 1702.5 MHz. All noise and interference must be rejected in this channel to maintain the sensitivity and performance of the receiver.

This measurement is simply a '*Channel Power* (sim)' measurement at the '*Image Frequency* (sim)'.

Channel Used: Image Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: Same as *CP* (sim)

Travel Direction: Same as CP

Image Frequency (IMGF)

This measurement is the image frequency from the input to the first mixer.

Since Spectrasys knows the 'Channel Frequency' of the specified path it also has the ability to figure out what the image frequency is up to the 1st mixer. After the 1st mixer the 'Image Frequency' measurement will show the main channel frequency. This measurement will show what that frequency is.

For example if we designed a 2 GHz receiver that had an IF frequency of 150 MHz using low LO side injection then the LO frequency would be 1850 MHz and image frequency for all stages from the input to the first mixer would be 1700 MHz.

Image Noise Rejection Ratio (IMGNR)

This measurement is the ratio of the 'Channel Noise Power' to 'Image Channel Noise Power' along the specified path as shown by:

IMGNR[n] = *CNP* (sim)[n] - *IMGNP* (sim)[n] (dB), where n = stage number

This measurement is very useful in determining the amount of image noise rejection that the selected path provides.

For this particular measurement basically two channels exist both with the same 'Channel Measurement Bandwidth' 1) main channel and 2) 1st mixer image channel.

Channel Used: Main Channel Frequency, Image Channel Frequency, and Channel Measurement Bandwidth

Types of Spectrums Used: Same as CNP and IMGNP

Travel Direction: Same as CNP and IMGNP

Image Rejection Ratio (IMGR)

This measurement is the ratio of the 'Channel Power' to 'Image Channel Power' along the specified path as shown by:

IMGR[n] = *DCP* (sim)[n] - *IMGP* (sim)[n] (dB), where n = stage number

For this particular measurement basically two channels exist both with the same 'Channel Measurement Bandwidth' 1) main channel and 2) 1st mixer image channel. The only difference between these two channels are their frequencies, one being at the 'Channel Frequency' and the other is at the 'Mixer Image Frequency'.

Channel Used: Main Channel Frequency, Image Channel Frequency, and Channel Measurement Bandwidth

Types of Spectrums Used: Same as DCP and IMGP

Travel Direction: Same as DCP and IMGP

Input 1 dB Compression - Equation Based (EIP1DB)

This is the traditional cascaded input 1 dB compression measurement based on the ***user entered gain and 1 dB compression values for the stages*** (*SGAIN* (sim) and *SOP1DB* (sim)). This measurement can be used to compare with traditional calculations typically found in spreadsheets.

1 / EIP1DB[n] = 1 / (SOP1DB[0] / ECGAIN[0]) + 1 / (SOP1DB[1] / ECGAIN[1]) + ... + 1 / (SOP1DB[n] / ECGAIN[n]) (Linear), where n = stage number

Caution: This measurement excludes all frequency, VSWR, and compression effects.

See *IP1DB* (sim) as the general cascaded input 1 dB compression measurement that includes all secondary effects.

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Channel Used: None

Types of Spectrums Used: None

Travel Direction: N/A

S Parameters
Equation based measurements use stage parameters directly entered by the user like gain, noise figure, intercept point, etc. If the model doesn't allow the user to enter one of these parameters, like the S parameter model, then equation based measurements will ignore these parameters.

Input 1 dB Compression Point (IP1DB)

This measurement is the system input 1 dB compression point referenced to the path input. The operating gain is used to refer the input 1 dB compression points of the individuals stages back to the input. The user specified gain will only equal the operating gain in a perfectly matched system operating under linear conditions.

1 / IP1dB = 1 / *SIP1DB* (sim)1 + 1 / (SIP1DB2 - *CGAIN* (sim)[1]) + 1 / (SIP1DB3 - CGAIN[2]) + ... + 1 / (SIP1DBX - CGAIN[X-1]), where X is the nth stage dBm

See *EIP1DB* (sim) for an equation based measurement that uses the user specified gain instead of the actual operating gain.

Channel Used: Same as CGAIN

Types of Spectrums Used: Same as CGAIN

Travel Direction: Same as CGAIN

NOTE
This measurement is an approximation based on the cascaded gain and the user entered compression points from each stage based on a single input power. For higher accuracy the Pin vs Pout should be created.

Input 2nd Order Intercept - Equation Based (EIIP2)

This is the traditional cascaded input 2nd order intercept measurement based on the ***user entered gain and 2nd order intercept values for the stages*** (*SGAIN* (sim) and *SOIP2* (sim)). This measurement can be used to compare with traditional calculations typically found in spreadsheets.

1 / EIIP2[n] = 1 / (SOIP2[0] / ECGAIN[0]) + 1 / (SOIP2[1] / ECGAIN[1]) + ... + 1 / (SOIP2[n] / ECGAIN[n]) (Linear), where n = stage number

Caution: This measurement excludes all frequency, VSWR, and compression effects. Also, equation based cascaded intermod calculations ignore that fact that the two tones used to create the intermod may actually be attenuated drastically like through and IF filter.

See *IIP2* (sim) as the general cascaded input 2nd order intercept measurement that includes all secondary effects.

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Channel Used: None

Types of Spectrums Used: None

Travel Direction: N/A

S Parameters
Equation based measurements use stage parameters directly entered by the user like gain, noise figure, intercept point, etc. If the model doesn't allow the user to enter one of these parameters, like the S parameter model, then equation based measurements will ignore these parameters.

Input 3rd Order Intercept - Equation Based (EIIP3)

This is the traditional cascaded input 3rd order intercept measurement based on the ***user entered gain and 3rd order intercept values for the stages*** (*SGAIN* (sim) and *SOIP3* (sim)). This measurement can be used to compare with traditional calculations typically found in spreadsheets.

1 / EIIP3[n] = 1 / (SOIP3[0] / ECGAIN[0]) + 1 / (SOIP3[1] / ECGAIN[1]) + ... + 1 / (SOIP3[n] / ECGAIN[n]) (Linear), where n = stage number

Caution: This measurement excludes all frequency, VSWR, and compression effects. Also, equation based cascaded intermod calculations ignore that fact that the two tones used to create the intermod may actually be attenuated drastically like through and IF filter.

See *IIP3* (sim) as the general cascaded input 3rd order intercept measurement that includes all secondary effects.

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Channel Used: None

Types of Spectrums Used: None

Travel Direction: N/A

S Parameters
Equation based measurements use stage parameters directly entered by the user like gain, noise figure, intercept point, etc. If the model doesn't allow the user to enter one of these parameters, like the S parameter model, then equation based measurements will ignore these parameters.

Input Intercept [All Orders] (IIP)

This measurement is the intercept point referenced to the path input. This is an in-band type of intermod measurement.

IIP[n] = *OIP* (sim)[n] - *CGAIN* (sim)[n] (dBm), where n = stage number

This measurement simple takes the computed 'Output Intercept' and references it to the input by subtracting the cascaded gain. The last IIP value for a cascaded chain will always be the actual input intercept for the entire chain.

Each column in this measurement is for a different intermod order up to the Maximum Order specified on the '*Calculate Tab* (sim)' of the System Analysis Dialog Box. The column number is the same as the order starting from the left with order 0.

See the 'Intermods Along a Path' section for information on how to configure these tests.

Remember intermod bandwidth is a function of the governing intermod equation. For example, if the intermod equation is 2F1 - F2 then the intermod bandwidth would be: 2BW1 + BW2. Note: Bandwidths never subtract and will always add. The channel bandwidth must be set wide enough to include the entire bandwidth of the intermod to achieve the expected results. The 'Automatic Intermod Mode' will set the bandwidth appropriately.

Caution: This method used to determine the intercept point is only valid for 2 tones with equal amplitude

Channel Used: Main Channel Frequency, Interferer Channel Frequency, and Channel Measurement Bandwidth

Types of Spectrums Used: Same as OIP and CGAIN

Travel Direction: Same as OIP and CGAIN

See the Intercept Measurements in the Lab and Cascaded Intermods and Spectrasys sections for additional information.

Input Intercept - Receiver [All Orders] (RX_IIP)

This measurement is the receiver input intercept point along the path. This is an out-of-band type of intermod measurement.

```
RX_IIP[n] = RX_OIP[n] - CGAIN[n] (dBm), where n = stage number
```

This measurement simple takes the computed 'Output Intercept' and references it to the input by subtracting the cascaded gain. The last IIP value for a cascaded chain will always be the actual input intercept for the entire chain.

Each column in this measurement is for a different intermod order up to the Maximum Order specified on the 'Calculate Tab' of the System Analysis Dialog Box. The column number is the same as the order starting from the left with order 0.

See the 'Intermods Along a Path' section for information on how to configure these tests.

Remember intermod bandwidth is a function of the governing intermod equation. For example, if the intermod equation is 2F1 - F2 then the intermod bandwidth would be: 2BW1 + BW2. Note: Bandwidths never subtract and will always add. The channel bandwidth must be set wide enough to include the entire bandwidth of the intermod to achieve the expected results. The 'Automatic Intermod Mode' will set the bandwidth appropriately.

Caution: This method used to determine the intercept point is only valid for 2 tones with equal amplitude

Channel Used: Main Channel Frequency, Interferer Channel Frequency, and Channel Measurement Bandwidth
Types of Spectrums Used: Same as RX_OIP and CGAIN
Travel Direction: Same as RX_OIP and CGAIN

See the Intercept Measurements in the Lab and Cascaded Intermods and Spectrasys sections for additional information.

Input Saturation Point (IPSAT)

This measurement is the system input saturation point referenced to the path input. The operating gain is used to refer the input saturation point of the individuals stages back to the input. The user specified gain will only equal the operating gain in a perfectly matched system operating under linear conditions.

1 / IPSAT = 1 / *SIPSAT* (sim)1 + 1 / (SIPSAT2 - *CGAIN* (sim)[1]) + 1 / (SIPSAT3 - CGAIN[2]) + ... + 1 / (SIPSATX - CGAIN[X-1]), where X is the nth stage dBm

See *EIPSAT* (sim) for an equation based measurement that uses the user specified gain instead of the actual operating gain.

Channel Used: Same as CGAIN

Types of Spectrums Used: Same as CGAIN

Travel Direction: Same as CGAIN

Input Saturation Power - Equation Based (EIPSAT)

This is the traditional cascaded input saturation measurement based on the ***user entered gain and saturation values for the stages*** (*SGAIN* (sim) and *SOPSAT* (sim)). This measurement can be used to compare with traditional calculations typically found in spreadsheets.

1 / EIPSAT[n] = 1 / (SOPSAT[0] / ECGAIN[0]) + 1 / (SOPSAT[1] / ECGAIN[1]) + ... + 1 / (SOPSAT[n] / ECGAIN[n]) (Linear), where n = stage number

Caution: This measurement excludes all frequency, VSWR, and compression effects.

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Channel Used: None

Types of Spectrums Used: None

Travel Direction: N/A

S Parameters
Equation based measurements use stage parameters directly entered by the user like gain, noise figure, intercept point, etc. If the model doesn't allow the user to enter one of these parameters, like the S parameter model, then equation based measurements will ignore these parameters.

Interferer Cascaded Gain (ICGAIN)

This measurement is the interferer cascaded gain of the main channel along the specified path. The 'Interferer Cascaded Gain' is the difference between the 'Interferer Channel Power' measurement at the nth stage minus the 'Interferer Channel Power' measurement at the input as shown by:

ICGAIN[n] = *ICP* (sim)[n] - ICP[0] (dB), where n = stage number

Channel Used: Main Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: Same as ICP

Travel Direction: Same as ICP

See the Intercept Measurements in the Lab and Cascaded Intermods and Spectrasys sections for additional information.

Interferer Channel Frequency (ICF)

This measurement is the frequency of the interferer used for intermod measurements such as: IIP, OIP, SFDR, etc.. The 'Interferer Channel Frequency' is determined set on the '*Calculate Tab* (sim)' of the System Analysis Dialog Box.

As with other frequency measurements Spectrasys is able to deal with frequency translation through mixers, frequency multipliers, etc.

Interferer Channel Power (ICP)

This measurement is the total integrated power in the interferer channel. This power is used for intermod measurements such as: IIP3, OIP3, SFDR, etc..

This measurement is simply a '*Desired Channel Power* (sim)' measurement at the ' *Interferer Channel Frequency* (sim)'.

Channel Used: Interferer Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: Same as *DCP* (sim)

Travel Direction: Same as DCP

See the Intercept Measurements in the Lab and Cascaded Intermods and Spectrasys sections for additional information.

Interferer Channel Voltage (ICV)

This measurement is the peak voltage in the interferer channel.

This measurement is simply a '*Desired Channel Voltage* (sim)' measurement at the ' *Interferer Channel Frequency* (sim)'.

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Channel Used: Interferer Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: Same as *DCV* (sim)

Travel Direction: Same as DCV

Interferer Gain (IGAIN)

This measurement is the gain of the interferer (tone) channel along the specified path. The 'Gain' is the difference between the 'Interferer Channel Power' output of the current stage minus the 'Interferer Channel Power' output of the prior stage as shown by:

IGAIN[n] = *ICP* (sim)[n] - ICP[n-1] (dB), where IGAIN[0] = 0 dB, n = stage number

Channel Used: Interferer (Tone) Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: Same as ICP

Travel Direction: Same as ICP

See the Intercept Measurements in the Lab and Cascaded Intermods and Spectrasys sections for additional information.

Minimum Detectable Signal (MDS)

This measurement is the minimum detectable (discernable) signal referred to the input and is equivalent to the input channel noise power plus the cascaded noise figure of the specified chain as shown by:

MDS[n] = *CNP* (sim)[0] + *CNF* (sim)[n] (dBm), where n = stage number

The MDS value at stage n represents the MDS of the entire system up to and including stage n. Consequently, the MDS of the entire system is the value indicated at the last stage in the path or chain. The minimum detectable signal is the equivalent noise power present on the input to a receiver that sets the limit on the smallest signal the receiver can detect.

For example, if the thermal noise power input to a receiver is -174 dBm/Hz and the channel bandwidth is 1 MHz (10 Log (1 MHz) = 60 dB) then the input channel power would be -114 dBm. For a cascaded noise figure of 5 dB the minimum detectable signal would be -109 dBm.

See the '*Channel Noise Power* (sim)' measurement to determine which types of signals are included or ignored in this measurement.

Channel Used: Main Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: Same as CNP and CNF

Travel Direction: Same as CNP and CNF

Minimum Detectable Signal - Equation Based (EMDS)

This is the traditional cascaded gain measurement based on the ***user entered gain values for the stages*** (*SGAIN* (sim)) and the equation based cascaded noise figure (*ECNF* (sim)). This measurement can be used to compare with traditional calculations typically found in spreadsheets.

EMDS[n] = Input thermal noise power + *ECNF* (sim)[n] (dBm), where n = stage number

The input thermal noise power = kTB, where k is Boltzman's constant, T is the temperature in Kelvin, and B is the channel bandwidth.

Caution: This measurement excludes all frequency, VSWR, and compression effects.

See *MDS* (sim) as the general minimum detectable signal measurement that includes all secondary effects.

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Channel Used: None

Types of Spectrums Used: None

Travel Direction: N/A

S Parameters
Equation based measurements use stage parameters directly entered by the user like gain, noise figure, intercept point, etc. If the model doesn't allow the user to enter one of these parameters, like the S parameter model, then equation based measurements will ignore these parameters.

Node Noise Voltage (NNV)

This measurement is the peak noise voltage at the node along the specified path. This includes all noise signals both in and out of the channel.

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Default Unit: dBV

Channel Used: No channel is used for this measurement

Types of Spectrums Used: *ONLY NOISE*

Travel Direction: Only spectrums traveling in the FORWARD path direction

Noise and Distortion Channel Power (NDCP)

This measurement is the integrated noise and distortion channel power in the main channel along the specified path. The Noise and Distortion Channel Power is the sum of the 'Channel Noise Power' plus the 'Total Intermod Channel Power' plus the 'Phase Noise

Channel Power' as shown by:

NDCP = *CNP* (sim)[n] + *TIMP* (sim)[n] + *PNCP* (sim)[n] (dB), where n = stage number

See the above measurements to determine which types of signals are included or ignored in this measurement.

Channel Used: Main Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: Same as CNP, TIMP, and PNCP

Travel Direction: Same as CNP, TIMP, and PNCP

Offset Channel Frequency (OCF)

The 'Offset Channel Frequency' and 'Offset Channel Power' are very useful measurements in Spectrasys. These measurements give the user the ability to create a user defined channel relative the main channel. The user specifies both the 'Offset Frequency' relative to the main 'Channel Frequency' and the 'Offset Channel Bandwidth'. As with the 'Channel Frequency' measurement Spectrasys automatically deals with the frequency translations of the 'Offset Channel Frequency' through frequency translations parts such as mixers and frequency multipliers. Both the 'Offset Frequency' and the 'Offset Channel Bandwidth' can be tuned by creating a variable for each of these parameters. This measurement simply returns the 'Offset Channel Frequency' for every node along the specified path.

Offset Channel Power (OCP)

The Offset Channel is a user defined channel relative to the main channel. The 'Offset Channel Frequency' and 'Offset Channel Bandwidth' are specified on the '*Options Tab* (sim)' of the System Analysis Dialog Box. As with the 'Channel Frequency' measurement Spectrasys automatically deals with the frequency translations of the 'Offset Channel Frequency' through frequency translation devices such as mixer and frequency multipliers.

For example, if the 'Channel Frequency' was 2140 MHz, 'Offset Channel Frequency' was 10 MHz, and the 'Offset Channel Bandwidth" was 1 MHz, then the OCP is the integrated power from 2149.5 to 2150.5 MHz.

This measurement is simply a '*Channel Power* (sim)' measurement at the '*Offset Channel Frequency* (sim)' using the '*Offset Channel Bandwidth* (sim)'.

Channel Used: Offset Channel Frequency and Offset Channel Bandwidth

Types of Spectrums Used: Same as *CP* (sim)

Travel Direction: Same as CP

Offset Channel Voltage (OCV)

The Offset Channel is a user defined channel relative to the main channel. The 'Offset Channel Frequency' and 'Offset Channel Bandwidth' are specified on the '*Options Tab* (sim)' of the System Analysis Dialog Box. As with the 'Channel Frequency' measurement Spectrasys automatically deals with the frequency translations of the 'Offset Channel Frequency' through frequency translation devices such as mixers and frequency multipliers.

For example, if the 'Channel Frequency' was 2140 MHz, 'Offset Channel Frequency' was 10 MHz, and the 'Offset Channel Bandwidth" was 1 MHz then the OCV is the average voltage from 2149.5 to 2150.5 MHz.

This measurement is simply a '*Channel Voltage* (sim)' measurement at the '*Offset Channel Frequency* (sim)' using the '*Offset Channel Bandwidth* (sim)'.

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Channel Used: Offset Channel Frequency and Offset Channel Bandwidth

Types of Spectrums Used: Same as *CV* (sim)

Travel Direction: Same as CV

Output 1 dB Compression - Equation Based (EOP1DB)

This is the traditional cascaded output 1 dB compression measurement based on the ***user entered gain and 1 dB compression values for the stages*** (*SGAIN* (sim) and *SOP1DB* (sim)). This measurement can be used to compare with traditional calculations typically found in spreadsheets.

1 / EOP1DB[n] = 1 / (SOP1DB[0] * ECGAIN[n] / ECGAIN[0]) + 1 / (SOP1DB[1] * ECGAIN[n] / ECGAIN[1]) + ... + 1 / SOP1DB[n] (Linear), where n = stage number

Caution: This measurement excludes all frequency, VSWR, and compression effects.

See *IP1DB* (sim) as the general cascaded gain measurement that includes all secondary effects.

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Channel Used: None

Types of Spectrums Used: None

Travel Direction: N/A

S Parameters
Equation based measurements use stage parameters directly entered by the user like gain, noise figure, intercept point, etc. If the model doesn't allow the user to enter one of these parameters, like the S parameter model, then equation based measurements will ignore these parameters.

Output 1 dB Compression Point (OP1DB)

This measurement is the output 1 dB compression point along the path. The gain is used

to refer the output 1 dB compression points of the individuals stages forward to the output. The user specified gain will only equal the operating gain in a perfectly matched system operating under linear conditions.

1 / OP1dB = 1 / (SOP1dB1 + Gain2 ... + GainX) + 1 / (SOP1dB2 + Gain3 ... + GainX) + ... + 1 / SOP1dBX dBm, where X is the nth stage

See *EOP1DB* (sim) for an equation based measurement that uses the user specified gain instead of the actual operating gain.

Channel Used: Same as CGAIN

Types of Spectrums Used: Same as CGAIN

Travel Direction: Same as CGAIN

NOTE
This measurement is an approximation based on the cascaded gain and the user entered compression points from each stage based on a single input power. For higher accuracy the Pin vs Pout should be created.

Output 2nd Order Intercept - Equation Based (EOIP2)

This is the traditional cascaded output 2nd order intermod measurement based on the ***user entered gain and 2nd order intercept values for the stages*** (*SGAIN* (sim) and *SOIP2* (sim)). This measurement can be used to compare with traditional calculations typically found in spreadsheets.

1 / EOIP2[n] = 1 / (SOIP2[0] * ECGAIN[n] / ECGAIN[0]) + 1 / (SOIP2[1] * ECGAIN[n] / ECGAIN[1]) + ... + 1 / SOIP2[n] (Linear), where n = stage number

Caution: This measurement excludes all frequency, VSWR, and compression effects. Also, equation based cascaded intermod calculations ignore that fact that the two tones used to create the intermod may actually be attenuated drastically like through and IF filter.

See *OIP2* (sim) as the general cascaded output 2nd order intercept measurement that includes all secondary effects.

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Channel Used: None

Types of Spectrums Used: None

Travel Direction: N/A

S Parameters
Equation based measurements use stage parameters directly entered by the user like gain, noise figure, intercept point, etc. If the model doesn't allow the user to enter one of these parameters, like the S parameter model, then equation based measurements will ignore these parameters.

Output 3rd Order Intercept - Equation Based (EOIP3)

This is the traditional cascaded output 3rd order intermod measurement based on the ***user entered gain and 3rd order intercept values for the stages*** (*SGAIN* (sim) and *SOIP3* (sim)). This measurement can be used to compare with traditional calculations typically found in spreadsheets.

1 / EOIP3[n] = 1 / (SOIP3[0] * ECGAIN[n] / ECGAIN[0]) + 1 / (SOIP3[1] * ECGAIN[n] / ECGAIN[1]) + ... + 1 / SOIP3[n] (Linear), where n = stage number

Caution: This measurement excludes all frequency, VSWR, and compression effects. Also, equation based cascaded intermod calculations ignore that fact that the two tones used to create the intermod may actually be attenuated drastically like through and IF filter.

See *OIP3* (sim) as the general cascaded output 3rd order intercept measurement that includes all secondary effects.

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Channel Used: None

Types of Spectrums Used: None

Travel Direction: N/A

S Parameters
Equation based measurements use stage parameters directly entered by the user like gain, noise figure, intercept point, etc. If the model doesn't allow the user to enter one of these parameters, like the S parameter model, then equation based measurements will ignore these parameters.

Output Intercept - All Orders [All Orders] (OIP)

This measurement is the output intercept point along the path. This is an in-band type of intermod measurement.

OIP[n] = ICP[n] + Delta[n] (dBm), where n = stage number and Order = order of the intermod

Delta[n] = (*ICP* (sim)[n] - *TIMCP* (sim)[n]) / (Order - 1) (dB)

Delta is the difference in dB between the 'Total Intermod Channel Power' in the main channel and the interfering signal present in the 'Interferer Channel' including the effects of the order.

In order to make this measurement a minimum of two signals (tones) must be present at the input.

- first interfering signal
- second interfering signal

The Channel Frequency must be set to the intermod frequency and the Interferer frequency must be set the first or second interfering frequency. See the 'Calculate Tab' on the System Analysis Dialog Box to set the Interfering Frequency. Furthermore, the spacing of the interfering tones needs to be such that intermods will actually fall into the main channel. If these conditions are not met then no intermod power will be measured in the main channel.

Each column in this measurement is for a different intermod order up to the Maximum Order specified on the '*Calculate Tab* (sim)' of the System Analysis Dialog Box. The column number is the same as the order starting from the left with order 0.

Remember intermod bandwidth is a function of the governing intermod equation. For example, if the intermod equation is 2F1 - F2 then the intermod bandwidth would be: 2BW1 + BW2. Note: Bandwidths never subtract and will always add. The channel bandwidth must be set wide enough to include the entire bandwidth of the intermod to achieve the expected results. The 'Automatic Intermod Mode' will set the bandwidth appropriately.

Note: Cascaded intermod equations are not used in Spectrasys.

Caution: This method used to determine the intercept point is only valid for 2 tones with equal amplitude

Channel Used: Interferer Channel Frequency, Main Channel Frequency, and Channel Measurement Bandwidth

Types of Spectrums Used: Same as ICP and TIMCP

Travel Direction: Same as ICP and TIMCP

See the Intercept Measurements in the Lab and Cascaded Intermods and Spectrasys sections for additional information.

Output Intercept - Receiver [All Orders] (RX_OIP)

This measurement is the receiver output intercept point along the path. This is an out-of-band type of intermod measurement.

```
RX_OIP[n] = VTCP[n] + RX_Delta[n] (dBm), where n = stage number and Order = order of the intermod
VTCP[n] = ICP[0] + CGAIN[n] (dBm)
RX_Delta[n] = ( VTCP[n] - TIMCP[n] ) / ( Order - 1) (dB)
```

Delta is the difference in dB between the 'Total Intermod Channel Power' in the main channel and the interfering signal present in the 'Interferer Channel' including the effects of the order. In order to correctly calculate OIP due to out-of-band interferers a Virtual Tone is created whose virtual power is that of an un-attenuated in-band tone. This power level is simply the 'Interferer Channel Power' at the input plus the 'Cascaded Gain'.

This Virtual Tone Channel Power is different than the 'Interferer Channel Power' measurement because the Virtual Tone Channel Power is not attenuated by out-of-band rejection whereas the 'Interferer Channel Power' can be. For in-band interferers the Virtual Tone Channel Power and the 'Interferer Channel Power' measurement will be identical.

In order to make this measurement a minimum of three signals (tones) must be present at the input.

- main channel signal (used for cascaded gain and total intermod channel power measurements)
- first interfering signal
- second interfering signal

The Channel Frequency must be set to the intermod frequency and the Interferer frequency must be set the first or second interfering frequency. See the 'Calculate Tab' on the System Analysis Dialog Box to set the Interfering Frequency. Furthermore, the spacing of the interfering tones needs to be such that intermods will actually fall into the main channel. If these conditions are not met then no intermod power will be measured in the main channel.

Each column in this measurement is for a different intermod order up to the Maximum Order specified on the 'Calculate Tab' of the System Analysis Dialog Box. The column number is the same as the order starting from the left with order 0.

See the 'Intermods Along a Path' section for information on how to configure these tests.

Remember intermod bandwidth is a function of the governing intermod equation. For example, if the intermod equation is 2F1 - F2 then the intermod bandwidth would be: 2BW1 + BW2. Note: Bandwidths never subtract and will always add. The channel bandwidth must be set wide enough to include the entire bandwidth of the intermod to achieve the expected results. The 'Automatic Intermod Mode' will set the bandwidth appropriately.

Cascaded intermod equations are not used in Spectrasys.

Caution: This method used to determine the intercept point is only valid for 2 tones with equal amplitude

Channel Used: Interferer Channel Frequency, Main Channel Frequency, and Channel Measurement Bandwidth
Types of Spectrums Used: Same as ICP, CGAIN, and TIMCP
Travel Direction: Same as ICP, CGAIN, and TIMCP

See the Intercept Measurements in the Lab and Cascaded Intermods and Spectrasys sections for additional information.

Output Saturation Point (OPSAT)

This measurement is the output 1 dB compression point along the path. The user specified gain will only equal the operating gain in a perfectly matched system operating under linear conditions.

1 / OPSAT = 1 / (SOPSAT1 + Gain2 ... + GainX) + 1 / (SOPSAT2 + Gain3 ... + GainX) + ... + 1 / SOPSATX dBm, where X is the nth stage

See *EOPSAT* (sim) for an equation based measurement that uses the user specified gain

instead of the actual operating gain.

Channel Used: Same as CGAIN

Types of Spectrums Used: Same as CGAIN

Travel Direction: Same as CGAIN

Output Saturation Power - Equation Based (EOPSAT)

This is the traditional cascaded output saturation measurement based on the ***user entered gain and saturation values for the stages*** (*SGAIN* (sim) and *SOPSAT* (sim)). This measurement can be used to compare with traditional calculations typically found in spreadsheets.

1 / EOPSAT[n] = 1 / (SOPSAT[0] * ECGAIN[n] / ECGAIN[0]) + 1 / (SOPSAT[1] * ECGAIN[n] / ECGAIN[1]) + ... + 1 / SOPSAT[n] (Linear), where n = stage number

Caution: This measurement excludes all frequency, VSWR, and compression effects.

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Channel Used: None

Types of Spectrums Used: None

Travel Direction: N/A

S Parameters
Equation based measurements use stage parameters directly entered by the user like gain, noise figure, intercept point, etc. If the model doesn't allow the user to enter one of these parameters, like the S parameter model, then equation based measurements will ignore these parameters.

Percent Intermods - All Orders (PRIM)

This routine calculates the Percent Intermod Contribution by each stage to the final Total Intermod Channel Power of the path.

IMREF - Equivalent Intermod Power Referenced to the Output

IMREF = *GIMCP* (sim)[n] + (CGAIN[nLastStage] - *CGAIN* (sim)[n])

PRIM[n] = IMREF[n] / *TIMCP* (sim)[nLastStage] (this is a ratio in Watts), Where PRIM[0] = 0, n is the current stage, and nLastStage is the last stage along the designated path

This measurement will help the user pinpoint all stages and their respective contribution to the total third order intermod power of the selected path.

This measurement is unit-less since the measurement is a percentage. There can cases where the percentage sum of all the stages in the path does not equal 100%. For instance, if the architecture contains parallel paths then each path would contribute to the total third order intermod power but only a single path is considered in this measurement. Another case would be where there are sufficient VSWR interactions between stages that effect the intermod levels. Reducing the architecture to the spreadsheet case will always yield the expected spreadsheet answers with respect to percentages. Sometimes this measurement can be greater than 100% if the equivalent intermod power referenced to the output is greater than the actual total intermod channel power. A good example of this would be an amplifier where intermods are cancelled at the amplifier output. In this case the generated intermod power alone may be much higher that the total intermod output power.

Channel Used: Main Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: Same as GIMCP, GAIN, and TIMCP

Travel Direction: Same as GIMCP, GAIN, and TIMCP

Percent Noise Figure (PRNF)

This routine calculates the Percent Noise Figure contribution by each stage to the final Cascaded Noise Figure of the path.

PRNF[n] = *AN* (sim)[n] / *CNF* (sim)[nLastStage] * 100 (this is a ratio of dB values), where PRNF[0] = 0, n is the current stage, and nLastStage is the last stage along the designated path.

This measurement will help the user pinpoint all stages and their respective contribution to the total cascaded noise figure of the selected path.

This measurement is unit-less since the measurement is a percentage. There can be a few cases where the percentage sum of all the stages in the path does not equal 100%. For instance, if the architecture contains parallel paths then each path would contribute to the total cascaded noise figure but only a single path is considered in the measurement. Another case would be where there is sufficient VSWR interactions between stages that effect the noise so it does not change linearly with the gain. Reducing the architecture to the spreadsheet case will always yield the expected spreadsheet answers with respect to percentages. See the '*Cascaded Noise Figure* (sim)' measurement for additional information.

Channel Used: Main Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: Same as AN and CNF

Travel Direction: Same as AN and CNF

Phase Noise Channel Power (PNCP)

This measurement is the integrated phase noise power in the main channel along the specified path. Phase noise is displayed on the graphs in dBm/Hz and the channel bandwidth is ignored while displaying phase noise. However, for channel measurements like this one the phase noise is scaled by the channel bandwidth before being integrated.

Channel Used: Main Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: *ONLY PHASE NOISE*

Travel Direction: Only spectrums traveling in the FORWARD path direction

Phase Noise Channel Voltage (PNCV)

This measurement is the peak phase noise voltage in the main channel along the specified path. Phase noise is displayed on the graphs in V/sqrt(Hz).

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Channel Used: Main Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: *ONLY PHASE NOISE*

Travel Direction: Only spectrums traveling in the FORWARD path direction

Gain (GAIN)

This measurement is the gain of the main channel along the specified path. The 'Gain' is the difference, in dB, between the 'Desired Channel Power' output of the current stage minus the 'Desired Channel Power' output of the prior stage as shown by:

GAIN[n] = *DCP* (sim)[n] - DCP[n-1] (dB), where GAIN[0] = 0 dB, n = stage number

Note: The DCP measurement includes the source mismatch loss. When the source impedance is mismatched to the system input impedance this mismatch loss will appear as an additional loss seen by the first stage.

See the 'Desired Channel Power' measurement to determine which types of signals are included or ignored in this measurement.

Channel Used: Main Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: Same as DCP

Travel Direction: Same as DCPthis

Source Mismatch Loss (MML)

This measurement shows the mismatch loss between the internal input source and the first stage of the path. This loss will be 0 dB when the source impedance is equivalent to the input impedance of the specified path. This loss will appear as a loss for the GAIN measurement of the first stage.

This measurement includes **ONLY DESIRED SIGNALS** on the beginning node of the path, traveling in the **FORWARD** path direction. All other intermods, harmonics, noise, and phase noise signals are ignored.

Channel Used: Main Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: *ONLY DESIRED SIGNALS*

Travel Direction: Only in the FORWARD direction

Spurious Free Dynamic Range (SFDR)

This measurement is the spurious free dynamic range along the specified path as shown by:

SFDR[n] = 2/3 [*IIP3* (sim)[n] - *MDS* (sim)[n]] (dB), where n = stage number

The 'Spurious Free Dynamic Range' is the range between the Minimum Detectable (Discernable) Signal (MDS) and the input power which would cause the third order intermods to be equal to the MDS. The MDS is the smallest signal that can be detected and will be equivalent to the receiver noise floor with a signal to noise ratio of 0 dB. In other words the MDS = -174 dBm/Hz + System Noise Figure + 10 Log(Channel Bandwidth).

See the '*Input Intercept* (sim)' and '*Channel Noise Power* (sim)' measurements to determine which types of signals are included or ignored in this measurement.

Channel Used: Main Channel Frequency, Interferer Channel Frequency, and Channel Measurement Bandwidth

Types of Spectrums Used: Same as *IIP* (sim) and MDS

Travel Direction: Same as IIP and MDS

Spurious Free Dynamic Range - Equation Based (ESFDR)

This is the traditional cascaded spurious free dynamic range measurement based on the ***user entered gain, noise, and intercept values for the stages*** (*SGAIN* (sim), *SNF* (sim), *SOIP3* (sim)). This measurement can be used to compare with traditional calculations typically found in spreadsheets.

ESFDR[n] = 2/3 [*EIIP3* (sim)[n] - *EMDS* (sim)[n]] (dB), where n = stage number

The 'Spurious Free Dyanmic Range' is the range between the Minimum Detectable (Discernable) Signal (MDS) and the input power which would cause the third order intermods to be equal to the MDS. The MDS is the smallest signal that can be detected and will be equivalent to the receiver noise floor with a signal to noise ratio of 0 dB.

Caution: This measurement excludes all frequency, VSWR, and compression effects.

See *SFDR* (sim) as the general spurious free dynamic range measurement that includes all secondary effects.

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Channel Used: None

Types of Spectrums Used: None

Travel Direction: N/A

S Parameters
Equation based measurements use stage parameters directly entered by the user like gain, noise figure, intercept point, etc. If the model doesn't allow the user to enter one of these parameters, like the S parameter model, then equation based measurements will ignore these parameters.

Spurious Free Dynamic Range - Receiver (RX_SFDR)

This measurement is the spurious free dynamic range along the specified path as shown by:

RX_SFDR[n] = 2/3 [*RX_IIP3* (sim)[n] - *MDS* (sim)[n]] (dB), where n = stage number

The 'Spurious Free Dynamic Range' is the range between the Minimum Detectable (Discernable) Signal (MDS) and the input power which would cause the third order intermods to be equal to the MDS. The MDS is the smallest signal that can be detected and will be equivalent to the receiver noise floor with a signal to noise ratio of 0 dB. In other words the MDS = -174 dBm/Hz + System Noise Figure + 10 Log(Channel Bandwidth).

See the '*Input Intercept (Receiver)* (sim)' and '*Channel Noise Power* (sim)' measurements to determine which types of signals are included or ignored in this measurement.

Channel Used: Main Channel Frequency, Interferer Channel Frequency, and Channel Measurement Bandwidth

Types of Spectrums Used: Same as RX_IIP and MDS

Travel Direction: Same as RX_IIP and MDS

See the Intercept Measurements in the Lab and Cascaded Intermods and Spectrasys sections for additional information.

Stage Dynamic Range (SDR)

This measurement along the specified path as shown by:

SDR[n] = *SOP1DB* (sim)[n] - TNP[n] (dB), where n = stage number

This simple measurement shows the difference between the 1 dB compression point of the stage entered by the user and the 'Total Node Power' at the stage output. This measurement is extremely useful when trying to optimize each stage dynamic range and determine which stage that will go into compression first.

See the '*Stage Output 1 dB Compression Point* (sim)' and "Total Node Power' measurements to determine which types of signals are included or ignored in this measurement

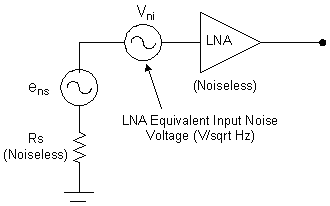
Channel Used: Main Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: Same as TNP

Travel Direction: Same as TNP

Stage Equivalent Input Noise Voltage (SVNI)

This measurement is the stage equivalent input noise voltage entered by the user. Stage noise figure, source resistance, and temperature can be converted to stage equivalent input noise voltage using the following equations.



Source Noise Voltage (V/sqrt Hz): ens(Rs) = sqrt(4 k T Rs) for example ens(50) = 0.895 nV

Stage Equivalent Input Noise Voltage: Vni = sqrt(F - 1) * ens

Stage Noise Factor: F = (Vni / ens)^2 + 1

Stage Noise Figure (dB): NF = 10 Log(F)

where Rs is the input source resistance to the stage

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Default Unit: nV / sqrt(Hz)

Channel Used: No channel is used for this measurement

Types of Spectrums Used: None

Travel Direction: N/A

Stage Gain (SGAIN)

This measurement is the stage gain entered by the user. For behavioral passive models the insertion loss parameter is used. When a stage doesn't have either a gain or insertion

loss parameter 0 dB is used. This measurement is not dependent on the path direction through the model. For example, if the path was defined through backwards through an amplifier the forward path gain would be reported not the reverse isolation of the amplifier.

Channel Used: No channel is used for this measurement

Types of Spectrums Used: None

Travel Direction: N/A

Stage Input 1 dB Compression Point (SIP1DB)

This measurement is the stage 1 dB compression point calculated by using the Stage Output 1 dB Compression Point and the Stage Gain. When a stage doesn't have this parameter +100 dBm is used.

SIP1DB[n] = *SOP1DB* (sim)[n] - *SGAIN* (sim)[n] (dBm), where n = stage number

Channel Used: No channel is used for this measurement

Types of Spectrums Used: None

Travel Direction: N/A

X-Parameters
This measurement for an X-parameter device will be based on the operating input and output impedance's found during the simulation.

Stage Input 1 dB Compression Voltage (SIV1DB)

This measurement is the stage 1 dB compression voltage point calculated by using the Stage Output 1 dB Compression Voltage Point and the Stage Voltage Gain. When a stage does not have this parameter 100 kV is used.

SIV1DB[n] = *SOV1DB* (sim)[n] - *SVGAIN* (sim)[n] (dBV), where n = stage number

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Channel Used: No channel is used for this measurement

Types of Spectrums Used: None

Travel Direction: N/A

Stage Input Impedance (SZIN)

This measurement is the stage input impedance entered by the user. When a stage does not have this parameter 50 ohms is used.

Channel Used: No channel is used for this measurement

Types of Spectrums Used: None

Travel Direction: N/A

Stage Input Intercept - All Orders (SIIP)

This measurement is the stage input intercept point calculated by using the Stage Output Intercept Point and the Stage Gain. When a stage doesn't have this parameter +100 dBm is used.

SIIP[n] = *SOIP* (sim)[n] - *SGAIN* (sim)[n] (dBm), where n = stage number

Each column in this measurement is for a different intermod order up to the Maximum Order specified on the '*Calculate Tab* (sim)' of the System Analysis Dialog Box. The column number is the same as the order starting from the left with order 0.

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Channel Used: No channel is used for this measurement

Types of Spectrums Used: None

Travel Direction: N/A

Stage Input Intercept Voltage - All Orders (SIIV)

This measurement is the stage input intercept voltage point calculated by using the Stage Output Intercept Voltage and the Stage Voltage Gain. When a stage doesn't have this parameter 100 kV is used.

SIIV[n] = *SOIV* (sim)[n] - *SVGAIN* (sim)[n] (dBV), where n = stage number

Each column in this measurement is for a different intermod order up to the Maximum Order specified on the '*Calculate Tab* (sim)' of the System Analysis Dialog Box. The column number is the same as the order starting from the left with order 0.

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Channel Used: No channel is used for this measurement

Types of Spectrums Used: None

Travel Direction: N/A

Stage Input Saturation Power (SIPSAT)

This measurement is the stage input saturation power calculated by using the Stage Output Saturation Power and the Stage Gain. When a stage doesn't have this parameter +100 dBm is used.

SIPSAT[n] = *SOPSAT* (sim)[n] - *SGAIN* (sim)[n] (dBm), where n = stage number

Channel Used: No channel is used for this measurement

Types of Spectrums Used: None

Travel Direction: N/A

X-Parameters
This measurement for an X-parameter device will be based on the operating input and output impedance's found during the simulation.

Stage Input Saturation Voltage (SIVSAT)

This measurement is the stage input saturation voltage calculated by using the Stage Output Saturation Voltage and the Stage Voltage Gain. When a stage does not have this parameter 100 kV is used.

SIVSAT[n] = *SOVSAT* (sim)[n] - *SVGAIN* (sim)[n] (dBV), where n = stage number

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Channel Used: No channel is used for this measurement

Types of Spectrums Used: None

Travel Direction: N/A

Stage Noise Figure (SNF)

This measurement is the stage noise figure entered by the user. For behavioral passive models the insertion loss parameter is used. When a stage doesn't have either a noise figure or insertion loss parameter 0 dB is used. This measurement is not dependent on the path direction through the model. For example, if the path was defined through the coupled port of a coupler the insertion loss of the coupler would be reported and not the coupled loss.

Channel Used: No channel is used for this measurement

Types of Spectrums Used: None

Travel Direction: N/A

Stage Output 1 dB Compression Point (SOP1DB)

This measurement is the stage 1 dB compression point entered by the user. When a stage does not have this parameter +100 dBm is used.

Channel Used: No channel is used for this measurement

Types of Spectrums Used: None

Travel Direction: N/A

X-Parameters
This measurement for an X-parameter device will be based on the operating input and output impedance's found during the simulation.

Stage Output 1 dB Voltage Compression Point (SOV1DB)

This measurement is the stage 1 dB voltage compression point entered by the user. When a stage does not have this parameter 100 kV is used.

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Channel Used: No channel is used for this measurement

Types of Spectrums Used: None

Travel Direction: N/A

Stage Output Impedance (SZOUT)

This measurement is the stage output impedance entered by the user. When a stage does not have this parameter 50 ohms is used.

Channel Used: No channel is used for this measurement

Types of Spectrums Used: None

Travel Direction: N/A

Stage Output Intercept - All Orders (SOIP)

This measurement is the stage output intercept point entered by the user. When a stage does not have this parameter +100 dBm is used.

Each column in this measurement is for a different intermod order up to the Maximum Order specified on the '*Calculate Tab* (sim)' of the System Analysis Dialog Box. The column number is the same as the order starting from the left with order 0.

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Channel Used: No channel is used for this measurement

Types of Spectrums Used: None

Travel Direction: N/A

Stage Output Intercept Voltage - All Orders (SOIV)

This measurement is the stage output intercept voltage point entered by the user. When a

stage does not have this parameter 100 kV is used.

Each column in this measurement is for a different intermod order up to the Maximum Order specified on the '*Calculate Tab* (sim)' of the System Analysis Dialog Box. The column number is the same as the order starting from the left with order 0.

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Channel Used: No channel is used for this measurement

Types of Spectrums Used: None

Travel Direction: N/A

Stage Output Saturation Power (SOPSAT)

This measurement is the stage saturation point entered by the user. When a stage does not have this parameter +100 dBm is used.

Channel Used: No channel is used for this measurement

Types of Spectrums Used: None

Travel Direction: N/A

X-Parameters
This measurement for an X-parameter device will be based on the operating input and output impedance's found during the simulation.

Stage Output Saturation Voltage (SOVSAT)

This measurement is the stage output voltage saturation point entered by the user. When a stage does not have this parameter 100 kV is used.

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Channel Used: No channel is used for this measurement

Types of Spectrums Used: None

Travel Direction: N/A

Stage Voltage Gain (SVGAIN)

This measurement is the stage voltage gain entered by the user. For behavioral passive models the insertion loss parameter is used. When a stage doesn't have either a gain or insertion loss parameter 0 dB20 is used. This measurement is not dependent on the path direction through the model. For example, if the path was defined through backwards through an amplifier the forward path gain would be reported not the reverse isolation of the amplifier.

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Channel Used: No channel is used for this measurement

Types of Spectrums Used: None

Travel Direction: N/A

Total Intermod Channel Power [All Orders] (TIMCP)

This measurement is the integrated total intermod power conducted from the prior stage plus the intermod power generated by the current stage.

In equation form the conducted third order intermod power is:

TIMCP[n] = integration of the total intermod spectrum at stage n across the main channel

Each column in this measurement is for a different intermod order up to the Maximum Order specified on the '*Calculate Tab* (sim)' of the System Analysis Dialog Box. The column number is the same as the order starting from the left with order 0.

Remember intermod bandwidth is a function of the governing intermod equation. For example, if the intermod equation is 2F1 - F2 then the intermod bandwidth would be: 2BW1 + BW2. Note: Bandwidths never subtract and will always add. The channel bandwidth must be set wide enough to include the entire bandwidth of the intermod to achieve the expected results. The 'Automatic Intermod Mode' will set the bandwidth appropriately.

Channel Used: Main Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: *ONLY INTERMODS and HARMONICS (separated according to their order)*

Travel Direction: Only in the FORWARD direction

Total Intermod Power (TIMP)

This measurement is the total integrated power of all intermod orders in the main channel along the path. This measurement differs from the 'Total Intermod Channel Power' in that it is a sum of all the orders of intermods whereas the 'Total Intermod Channel Power' is separated by order.

Channel Used: Main Channel Frequency and Channel Measurement Bandwidth

Types of Spectrums Used: *ONLY INTERMODS and HARMONICS*

Travel Direction: Only in the FORWARD direction

Total Node Power (TNP)

This measurement is the total integrated power of all spectrum types at the node. These spectrum types are: signals, harmonics, intermods, thermal noise, and phase noise.

Channel Used: None

Types of Spectrums Used: *All SIGNALS, INTERMODS, HARMONICS, THERMAL NOISE, and PHASE NOISE*

Travel Direction: All directions through the node

Total Node Voltage (TNV)

This measurement is the peak voltage of the entire spectrum at the node.

This is an extremely useful measurement in determining the total voltage present at the input of a device. This measurement includes ALL SIGNALS, INTERMODS, HARMONICS, NOISE, and PHASE NOISE traveling in ALL directions through the node.

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Default Unit: dBV

Channel Used: No channel is used for this measurement

Types of Spectrums Used: *All SIGNALS, INTERMODS, HARMONICS, NOISE, and PHASE NOISE*

Travel Direction: All directions through the node

Total RF Power Entering a Part (RFPwrIn)

This measurement is automatically added to the main dataset of the system analysis. This measurement contains the total RF power entering the given part. All input signals on all part terminals are summed together to determine the total RF input power.

Virtual Tone Channel Power (VTCP)

This measurement is the virtual tone channel power. The virtual tone is the power level of an un-attenuated tone along that cascade of stages. This tone consists of the power level of one of the tones at the input of a cascade plus the in-channel cascaded gain.

VTCP[n] = *ICP* (sim)[0] + *CGAIN* (sim)[n] (dB), where n = stage number

Channel Used: Same as ICP and CGAIN

Types of Spectrums Used: Same as ICP and CGAIN

Travel Direction: Same as ICP and CGAIN

See the Intercept Measurements in the Lab and Cascaded Intermods and Spectrasys sections for additional information.

Voltage DC (VDC)

This measurement is the DC voltage along the specified path.

This measurement includes **ALL SIGNALS**, **INTERMODS**, **HARMONICS**, **NOISE**, and **PHASE NOISE** traveling in **ALL** directions through the node.

Channel Used: No channel is used for this measurement

Types of Spectrums Used: *All SIGNALS, INTERMODS, HARMONICS, NOISE, and PHASE NOISE*

Travel Direction: All directions through the node

Voltage Gain (GAINV)

This measurement is the voltage gain of the main channel along the specified path. The 'Voltage Gain' is the difference between the 'Desired Channel Voltage' output of the current stage minus the 'Desired Channel Voltage' output of the prior stage as shown by:

GAINV[n] = *DCV* (sim)[n] - DCV[n-1] (dB20), where GAINV[0] = 0 dB20, n = stage number

See the '*Desired Channel Voltage* (sim)' measurement to determine which types of signals are included or ignored in this measurement.

Note
This measurement is not currently supported for non-linear circuit models such as X parameters, etc.

Channel Used: Main Channel Frequency and Channel Measurement Bandwidth

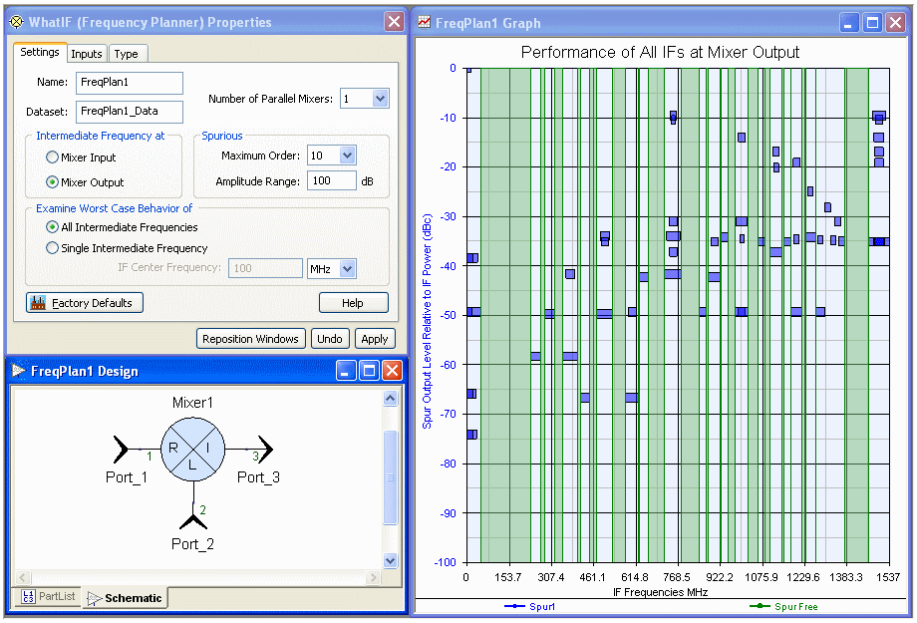
Types of Spectrums Used: Same as DCV

Travel Direction: Same as DCV

WhatIF Frequency Planner

A unique technique has been developed to reduce weeks and days spent doing spurious searches to hours and minutes. This technique is so powerful that spurious performance of all Intermediate Frequencies (IF) can be seen on a single graph. Spurious free regions are identified including multiple frequency band conversions to a common IF frequency. Users can see performance trade offs between IFs and can identify all spurious offenders giving them complete control over mixer requirements and specifications. This technique determines spurious responses and their amplitudes based on the characteristics of the mixers to be used, system frequencies and bandwidths, and desired IF bandwidths. Simulation speed is fast since conventional sweep analysis has been eliminated.

The WhatIF synthesis consists of the dialog box, schematic representation of the conversion process, and output graph.



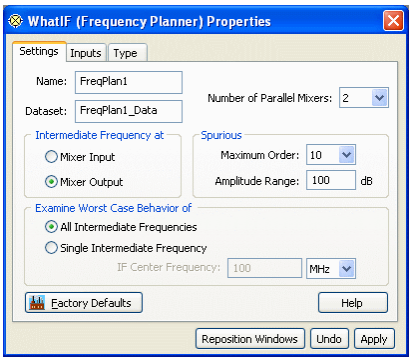
Note
The schematic is not used in the simulation process. No analysis is run on the mixer(s) in the schematic. They are present only as a visualization of the selected conversion scheme. The mixer in the schematic is generic and its parameters do not correspond to values in WhatIF. Only the mixer orientation and number of mixers are used in the schematic.

WhatIF Walkthrough

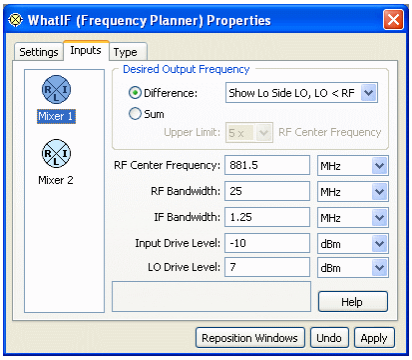
As an example let's suppose that we have a dual band receiver operating in the 869 to 894 and 1930 to 1990 MHz bands. A single IF frequency is desired to minimize cost of downstream components. An IF should be selected to minimize the potential for self inflicted interference. Let's also assume that spurs 100 dB below the desired IF output level is our definition of spurious free. Other important design criteria are IF bandwidth of 1.25 MHz and maximum RF input mixer drive level of -10 dBm with a mixer LO level of +7 dBm for the 800 MHz mixer and 0 dBm input drive level with a mixer LO level of +10 dBm for the 1900 MHz mixer. It will also be assumed that a double balance mixer will be used to predict spurious amplitudes. Default values for the double balanced mixer is assumed.

DESIGN STEPS

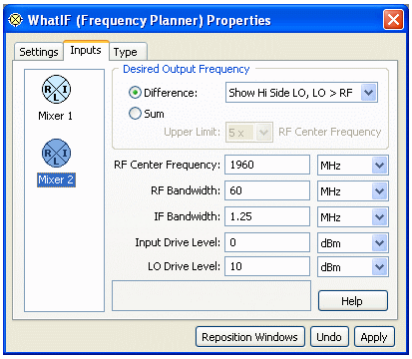
1. Create a new frequency planner synthesis by clicking the 'New' button. Find the 'Synthesis' submenu and then select the 'Add Frequency Planner' submenu item.
2. Change the number of parallel mixers to 2.



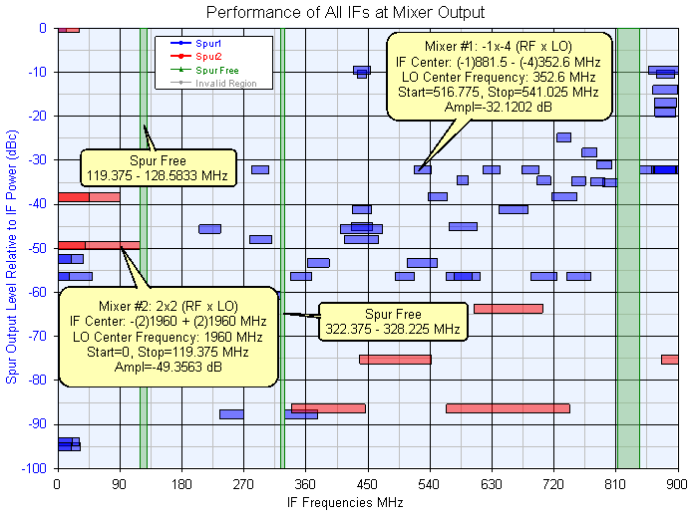
1. Click the 'Inputs' tab to specify the parameters for the parameters for our conversion scheme.
2. Click 'Mixer1'.



1. Select a Difference mixer with Low Side LO.
2. Enter the RF center frequency of 881.5 MHz.
3. Enter the RF bandwidth of 25 MHz (869 - 894 MHz).
4. Enter the IF bandwidth of 1.25 MHz.
5. Enter the Input drive level as -10 dBm.
6. Enter the LO drive level of +7 dBm.
7. Click 'Mixer2'.



1. Select a Difference mixer with High Side LO.
2. Enter the RF center frequency of 1960 MHz.
3. Enter the RF bandwidth of 60 MHz (1990 - 1930 MHz).
4. Enter the IF bandwidth of 1.25 MHz.
5. Enter the Input drive level of 0 dBm.
6. Enter the LO drive level of +10 dBm.
7. Click the 'Apply' button since defaults for all remaining parameters will be used.



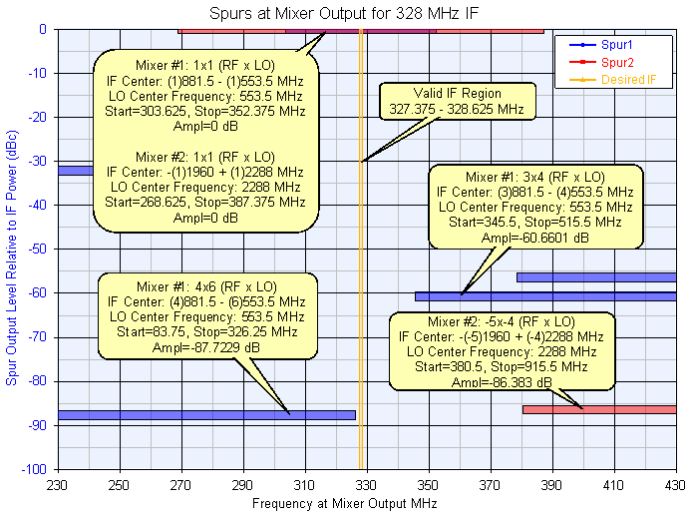
The figure shows results in an easy to interpret format. The performance of every valid IF frequency that meets the requirements for both RF bands appears on a single graph. Mouse fly-over text is used to identify each spur and spur free region. For the first time in the industry, IF frequency selection is optimized since performance trade offs are quickly made despite complications from multiband operation. Notice in Figure 1 that the half-IF spur (2x2 from 0 to 119.375 MHz) is easily identified on the left of the graph and well as all spurious free regions shaded in green.

To demonstrate the analysis capability of WhatIF a spur free IF frequency of 328 MHz will be selected and analyzed. This spur free region can be identified in preceding figure.

On the '**Settings**' tab:

- Select **Single Intermediate Frequency**
- Set **IF Center Frequency** to 328 MHz

When this IF frequency is specified an LO for each RF band is created internally. This LO along with the specified RF bands will be analyzed using the appropriate equation (i.e. FIF = m x FRF ± n x FLO). The mixer output is as shown.



This display is equivalent to looking at the down converted single IF spectrum output of the dual band conversion scheme on a spectrum analyzer. If the RF and LO frequencies where varied across their ranges the respective spurs would trace the given ranges.

Notice that the valid IF region is also shown for convenience.

Frequency Planning Fundamentals

Frequency planning is one the first steps performed during the RF architecture design phase. During this phase several frequency schemes are considered to ensure performance reliability internally as well as externally. External performance is typically controlled by a regulatory agency (i.e. Federal Communications Commission FCC or European Telecommunications Standards Institute ETSI) and the designer must comply with requirements before products are shipped and revenue collected. Spurious integrity for RF design is an age-old problem. Spurious responses, which occur naturally in the frequency conversion process, can render a particular design completely useless and non-compliant.

Spur Analysis Basics

The frequency conversion equation is:

FIF = m x FRF ± n x FLO or **FRF** = m x FIF ± n x FLO

m = 0, 1, 2, ...

n = 0, 1, 2, ...

FIF = IF frequency

FRF = any frequency in the RF band

FLO = any frequency in the LO band

From this equation all spurious products can be determined. It's easy to analyze frequencies that fall in a given IF band. Determining the amplitude of these spurious products is more challenging. But, selecting the best IF frequency can be difficult at best because of the time needed to create and analyze the hosts of data sets representing all of the possible spurious combinations for a single IF frequency. The process is further complicated for multiple band operation where a common IF is desired for all bands. Also, locating an input IF frequency while trying to maintain spectral purity across a wide output band is troublesome.

Spur (Intermod) Tables

In an ideal world on the **sum** and **difference** product would come out of a mixer. However, in the real world all the products governed by the following equation come out of the mixer.

FIF = m x FRF ± n x FLO or **FRF** = m x FIF ± n x FLO

m = 0, 1, 2, …
n = 0, 1, 2, …

FIF = IF frequency

FRF = any frequency in the RF band

FLO = any frequency in the LO band

One way to characterize the spurious performance of a mixer is to use a mixer spur table. This table shows the amplitude relationships of each harmonic combination of mixer input (RF/IF) and LO frequencies to the desired mixer output reference level.

The power of the spurious products on the mixer output is a strong function of the power levels of the RF and LO signals. A spur table example is shown below.

Table Characterization Parameters:

- F_{RF} = 500 MHz at -2 dBm
- F_{LO} = 470 MHz at 10 dBm
- F_{IF} = 30 MHz, measured to be -10 dBm

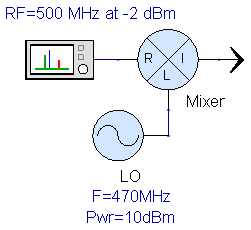
		LO			Harmonics					
		0	1	2	3	4	5	6	7	8	9	10
RF	0	X	14	29	23	42	25	43	53	57	65	72
	1	20	0	29	12	34	25	47	35	42	57	57
	2	52	40	58	40	58	41	50	48	66	53	68
Harmonics	3	46	49	50	49	53	49	52	48	58	57	51
	4	73	73	65	62	66	59	66	55	65	65	70
	5	77	76	84	63	64	60	59	60	59	68	71
	6	78	79	78	82	79	79	76	75	75	74	79
	7	79	78	77	79	82	80	81	80	80	79	78
	8	79	80	79	78	78	84	84	82	82	81	82
	9	79	79	80	79	78	79	84	84	82	83	82
	10	79	79	79	79	80	79	78	84	84	82	83

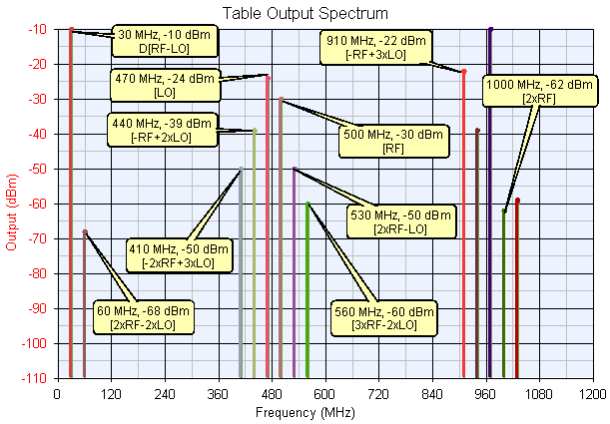
The table contains the *m* harmonics of the RF and *n* harmonics of the LO. All values in the table are relative to the desired output and are expressed in dBc. The desired output is the 1 x 1 entry in the table should always be 0 since this is the reference point. Even though no negative signs are shown all values are assumed to be below the desired output level of the 1 x 1 product. Some vendors may show a '+' sign next to the number indicating the value is above the reference level. The first column contains harmonics of RF (*n* = 0) and the first row contains harmonics of the LO (*m* = 0).

Spur Table Example

For example, the 1 x 3 product (1 x F_{RF} + 3 x F_{LO}) would occur at two frequencies: the sum at 1910 MHz and the difference at 910 MHz. The 1 x 3 entry in the table is **12**. This means that the absolute power level of these two frequencies is 12 dB below the desired output at 30 MHz. Since the table was characterized with an output at -10 dBm the spurious level of the 1 x 3 would be at -22 dBm.

The following figures show the setup use to measure the spur table shown above and the results for some of the lower orders.





Input Power Rolloff

Note
Theoretically, the 'm x FRF' or 'm x FIF' products will decrease (m-1) dB for each dB the input power (RF or IF) is decreased below the RF (IF) characterization level of the spur table.

The following table shows how the **relative** power changes at the mixer output when the input power is dropped by 1 dB.

		LO		Harmonic							
		0	1	2	3	4	5	6	7	8	9	10
RF	0	X	0	0	0	0	0	0	0	0	0	0
	1	0	0	0	0	0	0	0	0	0	0	0
	2	1	1	1	1	1	1	1	1	1	1	1
H	3	2	2	2	2	2	2	2	2	2	2	2
a	4	3	3	3	3	3	3	3	3	3	3	3
r	5	4	4	4	4	4	4	4	4	4	4	4
m	6	5	5	5	5	5	5	5	5	5	5	5
o	7	6	6	6	6	6	6	6	6	6	6	6
n	8	7	7	7	7	7	7	7	7	7	7	7
i	9	8	8	8	8	8	8	8	8	8	8	8
c	10	9	9	9	9	9	9	9	9	9	9	9

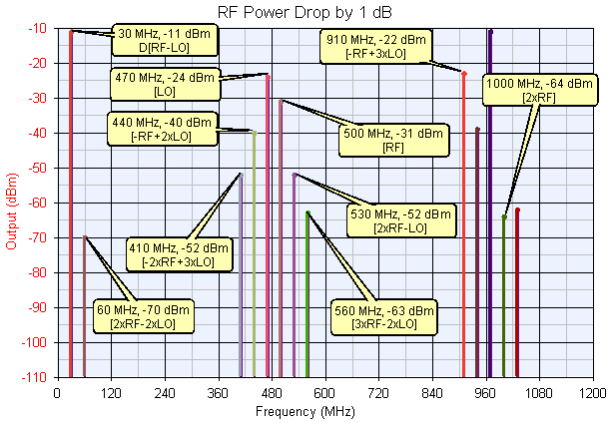
Harmonics of the LO are independent of the RF (IF) drive level. These spurious products will not change as the RF (IF) drive level changes. However, LO harmonics will drop 1 dB for every dB that the actual LO is below the LO characterization level.

Tip
When verifying a spur table set LO and RF levels to the same levels used to characterize the spur table. This will avoid additional calculations dealing with RF and LO power levels that may be different than table characterization levels.

Spur Table Example Rolloff

For example, all the 2 x F_{RF} +/- n x F_{LO} products shown in the prior graph will drop their relative values by 1 dB. All the 3 x F_{RF} +/- n x F_{LO} products will drop by 2 dB.

The following figure shows the output spectrum of the same mixer used in the prior graph when the input power level has dropped by 1 dB.



Remember
The desired about of the mixer also dropped by 1 dB. The new reference level must be used to determine spur table entries.

Orders Outside of Spur Table

In cases where the **maximum order** is set higher than orders specified in tables a **default suppression value** is used. Each spur table has an associated default suppression value.

Spur Table Limitations

Spur tables are limited in the following ways:

- They do not distinguish between sum or difference frequencies. The frequency response is assumed to be flat and the part perfect so the amplitudes are the same regardless of whether the output was a sum or a difference.
- They are only valid under the conditions the table was characterized under. Compression effects are only considered at the characterization state. If mixer input drive changes it is assumed the spur tables scales accordingly.

Mixer spur levels are affected by the load impedance they see at the mixer output. An accurate spur table would be characterized with the same load impedances as in the real system.

Traditional Approaches

Traditional tools include spreadsheets, spur charts, and many other types of custom tools.

The following weaknesses of these tools are readily apparent:

1. Interpretation of the results can be complicated and confusing
2. Assume the LO band is independent of the RF band
3. Don't account for IF bandwidth
4. Don't account for spurious amplitude

Interpretation of Results

Spurious searches typically involve the interpretation of large amounts of data. Charting techniques typically require normalized data and become very complicated when bandwidths are taken into account. Custom tools are difficult to use for engineering groups since many RF engineers have their own sets of custom tools they like to use. This makes it very difficult to pass the design from one engineer to another.

Dependence of LO and RF Bandwidths

This is one of the most glaring problems with spur search tools. In order to truly determine the performance of a given IF the LO band and RF band are always dependent and cannot be separated. The relationship between the RF, LO, and IF bands are as follows:

$$\mathbf{FRF_{BW} = FLO_{BW} + FIF_{BW}}$$

As seen the LO bandwidth must always be smaller than the RF bandwidth by the bandwidth of the IF. Violation of this rule falsely predicts spurs appearing across wider frequency ranges. True characterization of every IF frequency can only be obtained by preserving this bandwidth relationship for every case. However, this becomes tedious and time consuming with traditional approaches and is generally ignored.

Don't Account for IF Bandwidth

IF bandwidths are constantly increasingly causing additional difficulties during IF selection. As IF frequencies become wider the chances increase that a spur will fall in-band. However, as the IF bandwidth increases, the required LO bandwidth decreases yielding spurious combination's that cover smaller frequency ranges. The designer must account for these differences to minimize design time and component over or under specification.

Don't Account for Spurious Amplitude

Most spur searches are tedious and time consuming. Accounting for amplitude is yet another layer of complexity. Depending on the spurious combination and mixer input drive level, legitimate IF frequencies can be selected even though spurs may appear in-band if their amplitude is low enough. Knowing spurious amplitudes is very helpful when making trade-offs during the frequency planning process.

WhatIF

WhatIF is unique because it is a synthesis technique that accounts for the bandwidth of the RF, LO, and IF signals. This synthesis technique is fast and shows the entire frequency performance for every possible IF on a single easy to read graph.

WhatIF Setup

Maximum Order

Maximum Order = $m + n$

The maximum order determines the upper limit of spurious combinations. For certain mixer configurations the maximum order will affect the maximum IF or spurious response seen on the display.

If the maximum order is set higher than the orders contained in the spur table a default amplitude value will be used.

Amplitude Range

This range determines the limit on which spurious responses get displayed or thrown away. This range is in dB relative to the desired mixer output of the 1 x 1 product.

This range may also affects the upper limit of the spurious or IF frequencies being display. Generally, higher order products have lower amplitudes. These may fall outside the given amplitude range and will not be displayed.

Mixer Configuration Equations

In order to determine difference between **wanted** and **unwanted** IF frequencies the desired mixer configuration needs to be known. The following configurations can be selected by selecting the IF location shown in the **'Intermediate Frequency at'** group on the **Settings Tab** and by selecting parameters in the **'Desired Output Frequency'** group located on the **Inputs Tab**.

IF at the Mixer Output:

- Difference, Low Side LO: **IF = RF - LO**.
- Difference, High Side LO: **IF = LO - RF**.
- Sum: **IF = RF + LO**.

IF at the Mixer Input:

- Difference, IF < LO: **RF = IF - LO**.
- Difference, LO > IF: **RF = LO - IF**.
- Sum: **RF = IF + LO**.

Note
Some configurations require the user to set an upper limit of the desired output. This limit is use to restrict the range of spurious calculations.

Mixer Drive Levels

Mixer input (RF/IF) drive levels are used to determine the amplitude of the spurious responses. Harmonics of the mixer input (RF/IF) are mixed with the harmonics of the LO as shown by the following equation.

FIF = m x FRF ± n x FLO or **FRF** = m x FIF ± n x FLO

m = 0, 1, 2, …
n = 0, 1, 2, …

FIF = IF frequency

FRF = any frequency in the RF band

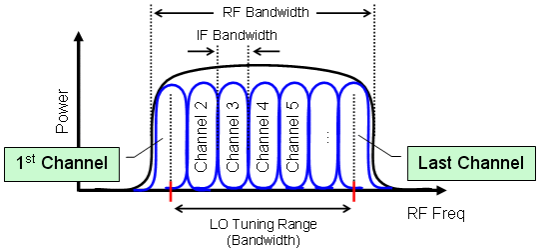
FLO = any frequency in the LO band

Note
Theoretically, the 'm x FRF' or 'm x FIF' products will decrease (m-1) dB for each dB the input power (RF or IF) is decreased. Obviously, the lower the input drive level to the mixer the more linear it will be and the lower the spurious responses are. See Input Power Rolloff for additional information.

Note
These levels have no effect on the widths (frequency ranges) of the spurious bands that are generated by WhatIF. The widths of these spurious bands are controlled by the RF Center Frequency, RF Bandwidth, IF Bandwidth, and mixer configuration (sum, difference low side LO, or difference high side LO).

Bandwidth Relationships

The RF, IF, and LO bandwidths are all related according to the following figure:



From this figure we see the following equation must be true:

$FRF_{BW} = FLO_{BW} + FIF_{BW}$

How it works

WhatIF uses closed form equations to generate all spurious responses. Mixer setup information is used to provide needed input to the calculation functions and restrict the output.

Note
The schematic is not used in the simulation process. No analysis is run on the mixer(s) in the schematic. They are present only as a visualization of the selected conversion scheme. The mixer in the schematic is generic and its parameters do not correspond to values in WhatIF. Only the mixer orientation and number of mixers are used in the schematic.

All Intermediate Frequencies

The mixer configuration information is used to determine the LO center frequency. This LO center frequency will change for every valid IF frequency. The IF frequency range will cover every possible IF that produces a valid IF for the given mixer configuration. See Valid IF Frequencies for additional information. The LO bandwidth remains constant for each IF frequency and is calculated according to the formula:

$FLO_{BW} = FRF_{BW} - FIF_{BW}$

The values of *m* and *n* are passed to an internal proprietary function which returns the range of intermediate frequencies over which the given spurious combination will be present. This range of IF's for the given combination of *m* and *n* is displayed on the graph if the amplitudes fall within the amplitude range.

Note
The upper IF (and LO) limit is a function of the **Mixer Configuration**, **RF Center Frequency** and corresponding **Bandwidth**, **Maximum Order**, and **Amplitude Range**. The **Mixer Configuration** will determine the location of the valid IF region. As the **RF Center Frequency** increases so will the needed LO. Larger values of the **Maximum Order** will cause a need for higher LO frequencies. If a spurious amplitude falls below the amplitude range then this spurious combination of *m* and *n* will be ignored.

Single Intermediate Frequency

When the user specifies the **IF Center Frequency** the exact LO frequency is determined from the selected mixer configuration. The LO bandwidth is determined from the RF and IF bandwidths according to the following equation:

$FLO_{BW} = FRF_{BW} - FIF_{BW}$

The LO range is displayed in the info window near the bottom of the **Inputs Tab**. *m* combinations of the input (RF / IF) and *n* combinations of the entire LO range is calculated and displayed on the graph.

WhatIF Output

The output of WhatIF is displayed by horizontal bars or bands. The width of the band corresponds to spur frequency range. The width of these bands is a function of the RF bandwidth, LO bandwidth, and the combination order. The LO bandwidth is a function of the IF bandwidth.

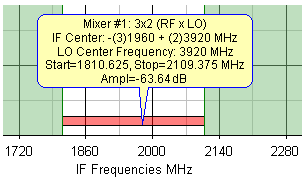
Each parallel frequency conversion (or mixer) is represented by a different color on the graph. The user can change these colors on the graphs properties page.

Flyover Help - The user can place the mouse cursor over any object in the graph to get information about the given spurious band or region. If you want the fly over help to become part of the graph it can be converted into an **'Info Balloon'** by right clicking on the trace and selecting the menu option **'Create Info Balloon'**.

All Intermediate Frequencies

The following figure shows and example response for all 3 x 2 spurs (RF x LO). The given

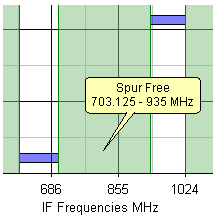
mixer configuration is a high side LO in a difference configuration. The flyover help show the particular *m* and *n* combination in this case a 3 x 2 spur. The next line show the equation how the center of the IF band is calculated. The LO center frequency is shown next. The **range of IF frequencies** that will have this 3 x 2 spurs is shown by the **Start** and **Stop** frequencies. The predicted amplitude is in dB below the 1 x 1 product is also shown.



Note
The bandwidth of the LO is not shown in the flyover help. The LO bandwidth is given by the equation: **$FLO_{BW} = FRF_{BW} - FIF_{BW}$** and is easily calculated by the user.

Note
This is not a spectrum analyzer type of graph. The horizontal axis is IF frequency. To determine the performance of any particular IF simply imagine a vertical line at that IF frequency. The performance can easily be determined by observing any horizontal spurious products crossing this vertical line.

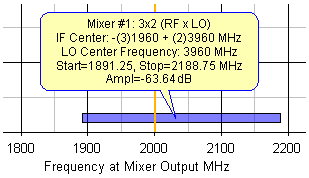
Spurious free regions across the given mixer dynamic range are shown in their own color. Spurious free ranges are post processing artifacts of the spurious calculations. IF frequencies that have no horizontal spur bands crossing them over the specified **amplitude range** are identified and shown as spur free regions in the graphs.



Note
Spur free ranges will generally change as the **Maximum Order** and **Amplitude Range** are changed.

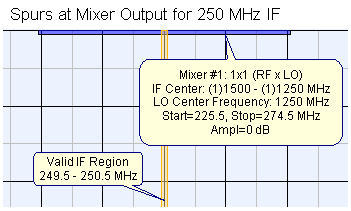
Single Intermediate Frequency

The output of the single intermediate frequency mode is much like a spectrum analyzer type of plot. The user can think of this plot as imaginary spectrum plot where we could place peak tracing markers on each spurious product as the RF and LO frequencies are moved across their entire ranges. The spurious products would trace these horizontal frequency bands.



The valid output region across the given mixer dynamic range is also shown in its own color. The graph shows a spectrum analyzer view showing the given mixer output frequency with its bandwidth along with any spurious frequencies that may appear in the output spectrum. One way to think of the spurious responses for this graph is that if the mixer input signal is swept across its input range and the LO is swept across its range the spurious signals would track the shown frequencies ranges.

At the top of the graph a 1x1 spur can be seen.

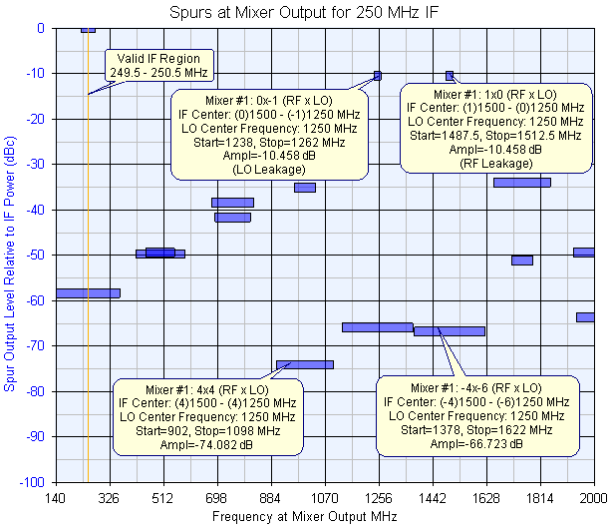


Remember
The desired output (valid region) will only occur at the single output frequency plus its bandwidth. However, if the input were to be swept across its entire range and the LO was swept across its entire range this would produce a 1x1 output, which covers a much larger range than the single IF frequency as shown on the graph.

In the above figure the RF bandwidth is 25 MHz and the IF bandwidth is 1 MHz. Using equations given earlier the LO bandwidth is 24 MHz. The 1x1 product always has the following bandwidth:

$$1x1_{BW} = FRF_{BW} + FLO_{BW}$$

The following figure is an example of an output for a 250 MHz IF appearing at the mixer output.



Note
THE GRAPHICAL OUTPUT IS ONLY FROM THE SELECTED MIXER CONFIGURATION. i.e. Sum or Difference. If a difference then only the information for the high or low side LO injection is shown. Since mixers produce products from all configurations the user should examine all configurations with the selected LO to determine the final spurious performance.

Valid IF Frequencies

Valid IF frequencies are based on mixer configuration equations. Valid IF frequencies are those frequencies whose equations give a **positive frequency result**. For example, if we were trying to find an IF for a sum mixer with the IF at the input (RF = IF + LO) then by definition the IF frequency must be below the RF band. No valid IF frequencies could exist for a sum output where the IF frequency was greater than the RF frequency. If the user wants an IF frequency greater than the RF band then a difference mixer must be selected. In the above example (RF = IF + LO) the maximum IF could be equal to the RF frequency when the LO is at 0 Hz. The minimum IF frequency is 0 Hz when the LO equals the RF frequency. Consequently, valid IF frequencies for this configuration would be between 0 Hz and the lowest RF frequency.

A summary of the valid ranges of the IF for given mixer configurations is shown in the following table.

Mixer Configuration	Valid IF Range
IF = LO - RF (High Side LO)	IF < LO
IF = RF - LO (Low Side LO)	IF < RF
IF = RF + LO	IF > RF
RF = LO - IF (High Side LO)	IF < LO
RF = IF - LO (Low Side LO)	IF > RF
RF = IF + LO	IF < RF

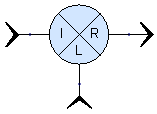
Remember
The LO injection side is determined from the relationship of the **LO Frequency** to the mixer **Input Frequency** not necessarily the RF frequency. When the IF frequency is at the mixer input then the injection side is determined by whether the LO frequency is higher or lower than the IF frequency.

Considerations when the IF is at the Mixer Input

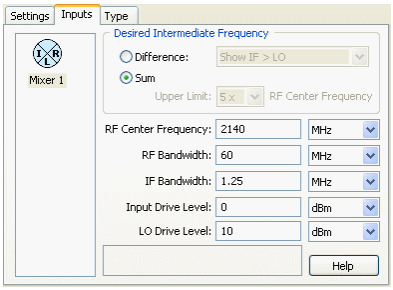
The graphical output results when the IF is at the mixer input are not the same as when the IF is at the mixer output. When the Worst Case Behavior of **'All Intermediate Frequencies'** is being examined for the IF input case the graphical output shows **IF frequencies that appear at the mixer input**. Once again this type of graph is showing what spurious combinations will appear in the **RF output band** after a given IF is selected.

The following figures show an example of the IF at the mixer input.

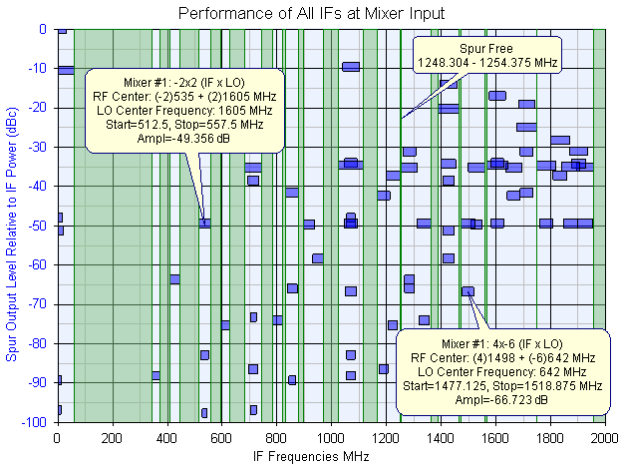
The visual representation:



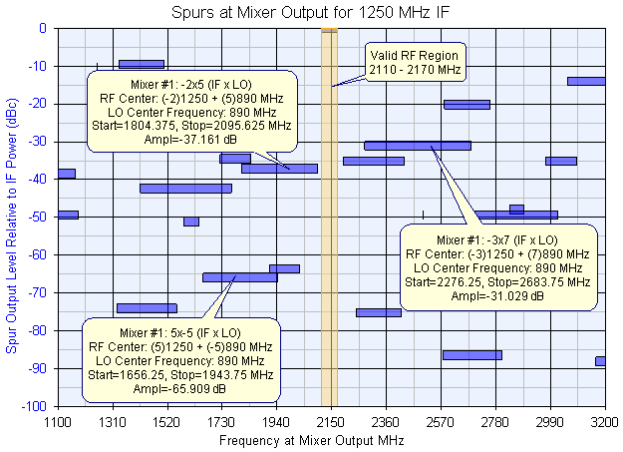
The mixer configuration:



Worst case behavior for all IF frequencies:



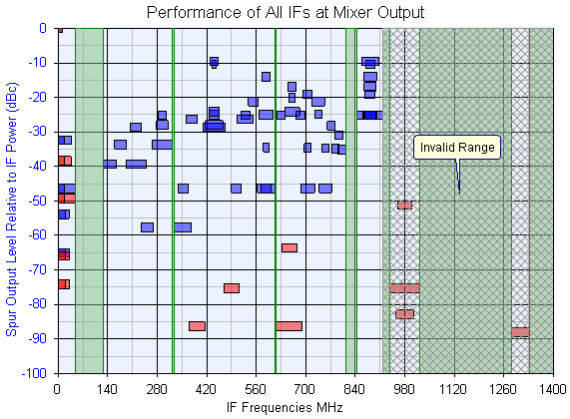
If a spur free IF is selected at **1250 MHz** the graphically output now shows the spurious performance of the **RF band at the mixer output**.



This graphs shows users the RF spurious performance at the mixer output without filtering. This output aids the designer in determining filtering requirements.

Invalid Ranges (Parallel Mixers Only)

In the parallel mixer case the intersection of valid frequencies for all mixers is shown on the graph. For example, one mixer could have a sum configuration and another could have a difference configuration. Only the IF frequencies that satisfy both of these conditions are valid. Frequencies outside the valid region will be marked as an **Invalid** since they don't have overlapping frequencies. Frequencies in this region may satisfy one or more of the parallel mixers configurations but not all of them.



In the above figure the gray hatched region shows an invalid range above about 900 MHz. The blue mixer has an RF center frequency of **881.5 MHz** configured as a difference mixer with **low side LO injection** with a **25 MHz** RF bandwidth. The red mixer has an RF center frequency of **1960 MHz** configured as a difference mixer with **high side LO injection** with a **60 MHz** RF bandwidth. Both mixers use a **1.25 MHz** IF bandwidth. The **maximum order** is 10.

Note
The user should quickly recognize how a common IF could **not** be selected for frequencies higher than 881.5 MHz since valid IF frequencies only exist below the RF frequency for this mixer configuration.

Degenerate Spurs

A degenerate spurious product is a spur that will be produced regardless of the LO frequency due to the relationship with the RF carrier frequency and the RF and IF bandwidths.

Degenerate spurious products occur under the following conditions for the specified configurations:

Intermediate Frequency at the Mixer Output

Spurious products will be created for any LO frequency when the ***following equation is true and the harmonic of the LO = 1*** :

$F_{RF} <= |(|M| \times BW_{RF} + BW_{LO} + BW_{IF}) / (2 (1 - |M|))|$

Where:

F_{RF} - RF Center Frequency

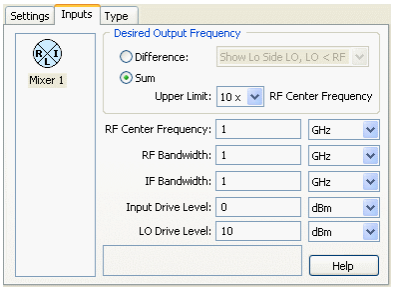
BW_{RF} - RF Bandwidth

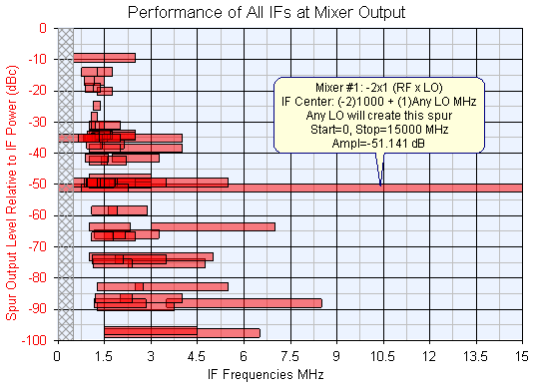
BW_{LO} - LO Bandwidth

BW_{IF} - IF Bandwidth

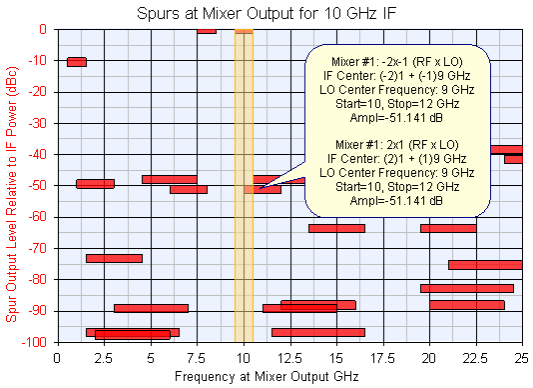
M - Harmonic of the RF

The following figures depict this degenerate case:





If a 10 GHz IF is picked the 2 x 1 degenerate spur will appear in the IF band as shown below.



Intermediate Frequency at the Mixer Input

Spurious products will be created for any LO frequency when the ***following equation is true and the harmonic of the LO = harmonic of the IF (M=N)*** :

$$\mathbf{F_{RF}} <= |(BW_{RF} + |M| \times BW_{LO} + |M| \times BW_{IF}) / (2 (1 - |M|))|$$

Where:

$\mathbf{F_{RF}}$ - RF Center Frequency

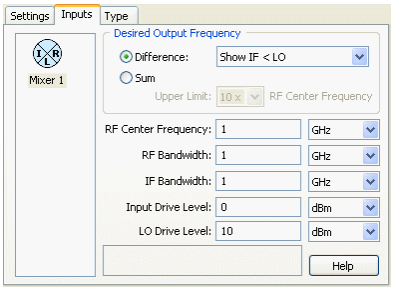
$\mathbf{BW_{RF}}$ - RF Bandwidth

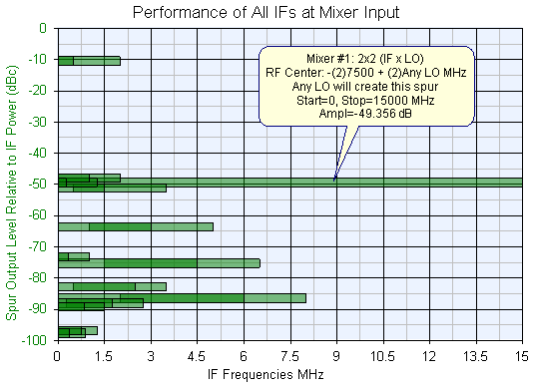
$\mathbf{BW_{LO}}$ - LO Bandwidth

$\mathbf{BW_{IF}}$ - IF Bandwidth

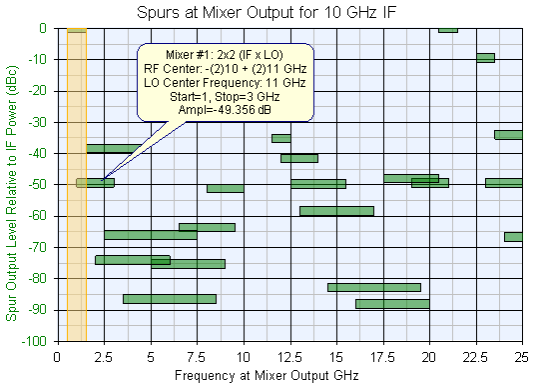
M - Harmonic of the IF

The following figures depict a degenerate case:





If a 10 GHz IF is picked the 2 x 2 degenerate spur will appear in the IF band as shown below.



Limitations

WhatIF only supports the agile conversion stage. However, multiple parallel agile conversion stages to a single IF frequency is supported. For multiple conversion stages it is recommended that Spectrasys be used to determine the performance of the entire system.

Creating an Intermod Table

To create an intermod table to use with WhatIF follow these steps:

Location
C:\Program Files\SystemVueXXXX.XX\Examples\RF Architecture Design\RF Design Kit\ **Simple Table Mixer.wsv**

1. Load the shipping example **'Simple Table Mixer'**.
2. Set the **'Input'** and **'LO'** frequencies of the sources in the schematic to those desired for characterization.
   Set the **'Input'** and **'LO'** power level of the soureces in the schematic to those desired for characterization. Set the **'RFTableInPwr'** and **'RFTableLOPwr'** parameters of the table mixer to these same power levels.
3. Set the **'RFTableDefSup'** of the table mixer to the desired default suppression value.
4. Adjust the **intermod table** values in the **equation block** associated with the **schematic** until the desired response is achieved.
5. Set the **'Designator'** of the mixer to the desired name that will appear in the part library.
6. Copy the table from the equation block and paste it into the **'RFTableData'** parameter of the table mixer. i.e. from **[** to **]**.

   Note
   WhatIF does not support equations in the intermod table. All variables used in a table need to be removed before the mixer is added to a part library.
7. Right click on the table mixer in the schematic and select **'Copy to Library'**.
8. Select a **new** or **existing** library.
9. Launch **WhatIF**.
10. Browse to the library where the table mixer was saved.

    Note
    If you copied the mixer to a new library you may have to add this library in the library manager.
11. Select this new table mixer.

Note
If the IF is at the mixer input then all of the **'IFTable...'** parameters must be updated.

Dialog Box Reference

General

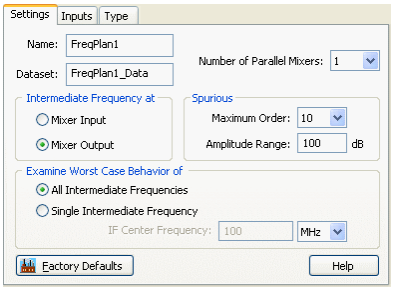
There **Reposition**, **Undo**, and **Apply** buttons that are available on every tab.

- **Reposition** - The Reposition Button (Reposition Windows)will cause the graph and schematic windows to be resized to fit within the Genesys environment.

- **Undo** - The Undo Button (Undo) will revert WhatIF to the state when the 'Apply' button was last clicked.
- **Apply** - The Apply Button (Apply) causes the spurious performance to be calculated and displayed.

Settings Tab

This tab is used to specify the type of spurious analysis that will be performed.



Name - Name of the frequency planner (WhatIF) that appears in the workspace tree.

Dataset - Name of the dataset where the WhatIF data will be stored when the apply button is clicked.

Intermediate Frequency at

Mixer Input - In this configuration WhatIF will determine a spurious combination of the mixer IF input and LO that will cause spurious signals to appear in the given RF band.

Mixer Output - In this configuration WhatIF will determine a spurious combination of the mixer RF input and LO that will cause spurious signals to appear in the IF band.

Examine Worst Case Behavior of

All Intermediate Frequencies - Shows graphically the performance of every valid IF for the given configuration. Each frequency conversion (or mixer) is represented by a different color on the graph.

Single IF Frequency - Specify an IF frequency that will be used to determine the exact LO frequencies for each mixer. Shows graphically output spectrum like what would be seen on a spectrum analyzer.

Number of Parallel Mixers - This is the number of RF bands that will be converted to / from a common IF. The schematic shows this representation when the 'Apply' button has been clicked. With the IF at the mixer input parallel mixers represent the numbers of RF bands which a simulcast will occur from a common IF. When the IF is at the mixer output then parallel mixers represent the number of RF bands being converted to a common IF.

Spurious

Maximum Order - Maximum order used for calculating spurious products

Amplitude Range - Only spurious products falling within this range of the desired 1x1 IF signal will be retained in the dataset and shown on the graph. The spurious free range is based on this amplitude range.

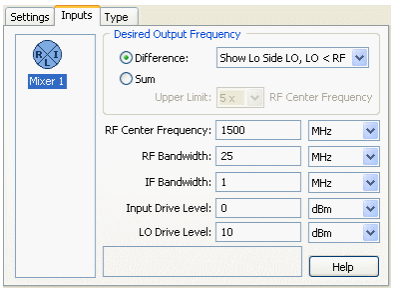
Apply - Clicking the button will cause the WhatIF frequency plan to be evaluated, the schematic to be updated, and the data to be saved in the dataset and displayed on the graph.

Undo - Clicking this button will undo all changes up to the last 'Apply'.

Factory Defaults - When clicked will reset all parameters on all tabs to their factory default values.

Inputs Tab

This tab is used to specify the characteristics of each RF band convert to or from an IF. Drive levels can also be specified for each of the mixers used in the conversion process.



The number of parallel mixers is shown in the list at the left. All other information on this page is based on the selected mixer from this list. Each mixer or conversion block can have its own parameters. However, only valid outputs will be shown on the graph. It is possible that the users can configure multiple mixers in such a way to prohibit a common IF.

Desired Intermediate Frequency

The user can select either whether they want to find an IF for either a sum or difference configuration. A difference configuration can be achieved through either a High Side LO (LO frequency greater than the RF/IF frequency) or a Low Side LO (LO frequency lower than the RF/IF frequency). For the sum configuration the user can specify how high in frequency they want to view the results.

RF Center Frequency Center frequency of the RF band.

RF Bandwidth - Bandwidth of the RF band.

IF Bandwidth - Bandwidth of the IF frequency or IF filter. The IF frequency is generally the bandwidth of the modulated signal. However, since brick wall filters are very difficult to build, this bandwidth can be increased to account for filtering imperfections. During the analysis, when examining all IF frequencies any spurious signals outside this bandwidth are ignored since brick wall filtering is assumed.

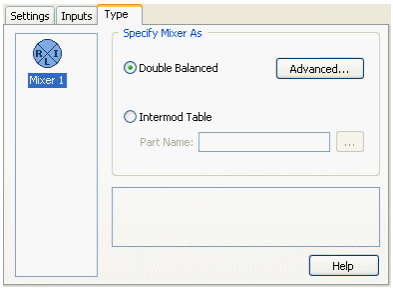
Input Drive Level - This is the input drive level to the mixer and has no bearing on the width of the spurious bands. This level is used in conjunction with the LO drive and the 'Mixer Type' to determine the amplitude of the spurious responses.

LO Drive Level - This is the LO drive level to the mixer and has no bearing on the width of the spurious bands. This level is used in conjunction with the input drive and 'Mixer Type' to determine the amplitude of the spurious responses.

Note
The RF Center Frequency, RF Bandwidth, and IF Bandwidth determine the width of the spurious bands. Internally, during the simulation process an LO center frequency and bandwidth is determined that is used for spurious calculations. Even though the mixer Input and LO drive levels don't affect the width of the spurious bands they can affect the spurious free bands since spurious free regions are determined within the specified amplitude dynamic range. To compensate for spurious amplitude inaccuracies the user can either specify a larger amplitude range or use the intermod table to specify more accurate mixer amplitude characteristics.

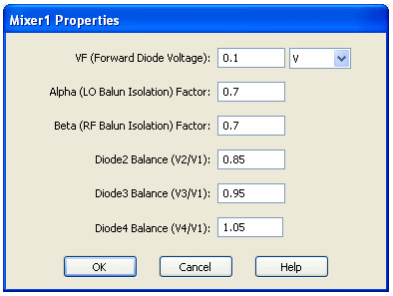
Type Tab

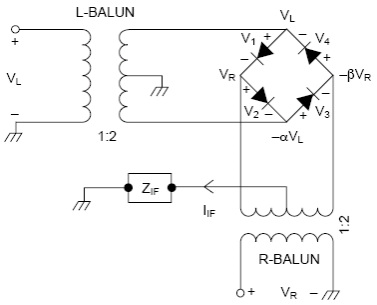
This tab is used to specify the amplitude performance of each mixer. The user has the choice of either a double balanced mixer or a user defined intermod table.



The number of parallel mixers is shown in the list at the left. All other information on this page is based on the selected mixer from this list. Each mixer or conversion block can have its own amplitude specifications. There are two types to choose from.

Double Balance - Parameters for the double balanced model can be changed by clicking the 'Advanced' button. This model is based on the work of Bert Henderson at Watkins Johnson. The name of the application note discussing this is, "Predicting Intermodulation Suppression in Double-Balanced Mixers ".

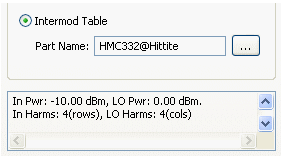




Parameters	Description
VF	Forward Diode Voltage
LO Balun Isolation factor (Alpha)	Isolation of the LO balun where isolation = 20 Log (1 - Alpha) dB. This isolation will be used for LO to RF and LO to IF.
RF Balun Isolation factor (Beta)	Isolation of the RF balun where isolation = 20 Log (1 - Beta) dB. This isolation will be used for RF to IF or IF to RF.
Diode2 Balance	Voltage balance between Diodes 2 and 1
Diode3 Balance	Voltage balance between Diodes 3 and 1
Diode4 Balance	Voltage balance between Diodes 4 and 1

Intermod Table - This is the part name and library of the table mixer. Click the ellipsis button (!ellipsis_button.gif!) to bring up the part selector. Select the 'Library' and 'Category' of the table mixers. At noted, this field uses actual mixer intermod table parts. Not just an numerically defined intermod table. Drive levels and directions through the mixer are read from this model. The corresponding drive levels for those tables will also be used.

The extracted table parameters will appear in the **info window** near the bottom of the tab.



Remember
If the IF is at the **mixer output** then the **RFTable...** will be used. If the IF is at the **mixer input** then the *IFTable...' will be used.

Note
The default table mixer in SystemVue can be used here.