Advanced Design System

Advanced Design System 2020 Update Release Notes

KEYSIGHT TECHNOLOGIES

Generated on : Sep 25, 2019

Notices

Restricted Rights Legend

Contents

ADS 2020 Update 0.2 Release Notes

Release: September 26, 2019

ADS 2020 Update 0.2 (minor update release) is a cumulative minor update release installed on ADS 2020 (base release). You can upgrade your existing ADS installation (ADS 2020) to ADS 2020 Update 0.2 without uninstalling any previous minor updates.

Version

510.update0.2

Platform Support

- **Supported Platforms**: Windows and Linux 64-bit.

Enhancements

ADS 2020 Update 0.2 includes enhancements in Design Editing.

Design Editing

- Added AEL functions to get end or corner for line segment of an interconnect.
- Added db_rotate_selected_objects() function to rotate all the selected objects in the given context.

For details on above AEL functions see, Design Editing AEL Functions.

Issues Addressed

ADS 2020 Update 0.2 addresses issues related to Design and Technology Management, Design Editing, ElectroThermal, Examples & Design Guides, FEM, SIPro, PIPro, and RFPro.

Design & Tech Management

- Fixed the crash when workspace changes during a library rename.

Design Editing

- Added AEL functions to use in macro recording when user places an unconstrained via.
- Fixed the Constraint Editor GUI crash issue.

For details on above AEL functions see, Design Editing AEL Functions.

ElectroThermal

- FloorPlanner results are now accurate when power specified is in [mW/uW/pW].

Examples & Design Guides

- Fixed the examples and added missing datasets to avoid simulation failure.

EM Simulation

- Fixed the complex bsub -R resource strings problem in an RFPro or SIPro simulation on a cluster queue.
- Fixed the SIPro simulation failure issue with "x_mode error" of "more than 10 minutes passed and file still not available".
- Fixed an issue in RFPro where pins in a flipped multi-tech design were positioned at the wrong height.

ADS 2020 Update 0.1 Release Notes

Release: September 4, 2019

ADS 2020 Update 0.1 (minor update release) is a cumulative minor update release installed on ADS 2020 (base release). You can upgrade your existing ADS installation (ADS 2020) to ADS 2020 Update 0.1 without uninstalling the existing version.

Version

510.update0.1

Platform Support

- **Supported Platforms**: Windows and Linux 64-bit.

Enhancements

ADS 2020 Update 0.1 includes enhancements in Circuit Simulation and Design Editing.

Circuit Simulation

- Added support for SiMKit 5.1.2 to ADS.

Design Editing

- Now can draw a Trace (with AEL commands) using a Line Type.
- Added tech_get_padstack_def_names() AEL function.

- Added AEL functions to extract Keepout information.
- Added AEL functions to get ground plane outline points.
- Added AEL functions to access via name, template name, start stop layer of a PCB Via instance.
- Added AEL iteration inside interconnect, get layer, via information, and net.

Issues Addressed

ADS 2020 Update 0.1 addresses issues related to Circuit Simulation, Design and Technology Management, DRC and LVS, EM Integration, Power Electronics, and RFPro.

Circuit Simulation

- GaN model can now be tuned.
- ADS Simulation is working fine for Capacitor devices.
- Fixed the GF 12 nm "slvtnfetrfe" device simulation issue.
- Using GF 22FDX kit with ADS now does not give any error.

Design and Technology

- Fixed possible crash editing Instance Parameters.
- Fixed possible crash in Design Search and Show Reference.

DRC and LVS

- Assura DRC: Added support for GDS datatype while loading Assura DRC output files.

EM Integration

- Customizing the substrate database path now expands environment variable.
- Added LTD support using RFPro in ADS
- RFPro generated sub circuit now preserve cdf component parameter values.
- Via layer now exists in the Virtuoso-RFPro.

Power Electronics

- Improved the Conducted EMI results data display for better spectral accuracy and noise level.
- Fixed the issue where the file name along with the file path was being used instead of just the file name in the NetlistInclude component's IncludeFile field, generated during netlist file import.
- Fixed the issue where the netlist file import failed on Windows operating systems when there was a space in the input file's path.

RFPro

- The .matdb file now is not read in case of an .ltd substrate.

- Fixed the issue with wrong net assignment to Virtual pin created.

SIPro

- Fixed the FEM distributed simulation failure issue.

Design Editing AEL Functions

Interconnect Functions

db_interconnect_get_line_endcap_type_before_index()

Returns the end cap type for the interconnect segment before index.

Syntax

```
decl endcapType = db_interconnect_get_line_endcap_type_before_index(object,
index);
```

where:

- object is interconnect composite object (multi segment trace).
- index is the point index before which line type is sought: 0 <= index <= Number of points.

The return value is one of the following 3 end type values:

- DB_SQUARE_CAP (1) = truncated at the end
- DB_ROUND_CAP (2) = rounded at the end (the extension radius is of half the width)
- DB_SQUARE_CAP_THAT_EXTENDS_HALF_WIDTH (3) = the end is extended by half the trace width

Example

```
decl context = de_get_current_design_context();
decl compositeIter = db_create_composite_iter(context);
    for (; db_composite_iter_is_valid(compositeIter); compositeIter =
db_composite_iter_get_next(compositeIter))
    {
        decl composite = db_composite_iter_get_composite_object(compositeIter);
        decl index = 0; // where 0 <= index <= number of points
        decl line_type = db_interconnect_get_line_type_before_index(composite,
index);
                                decl endType =
db_interconnect_get_line_endcap_type_before_index(composite, index);
    }
```

db_interconnect_get_line_corner_type_before_index()

Returns the corner (bend) type for the interconnect segment before index.

Syntax

```
decl cornerType =  db_interconnect_get_line_corner_type_before_index(object,
index);
```

where:

- object is interconnect composite object (multi segment trace).
- index is the point index before which line type is sought: 0 <= index <= Number of points.

The return value is one of the following 5 corner type values:

- DB_MITERED_CORNER (1) = mitered
- DB_SQUARE_CORNER (2) = square
- DB_CURVED_CORNER (3) = curved
- DB_ADAPTIVE_MITER_CORNER (4) = adaptive miter
- DB_ROUNDED_CORNER (5) = rounded

Example

```
decl context = de_get_current_design_context();
decl compositeIter = db_create_composite_iter(context);
    for (; db_composite_iter_is_valid(compositeIter); compositeIter =
db_composite_iter_get_next(compositeIter))
    {
        decl composite = db_composite_iter_get_composite_object(compositeIter);
        decl index = 0; // where 0 <= index <= number of points
        decl line_type = db_interconnect_get_line_type_before_index(composite,
index);
                                decl cornerType =
db_interconnect_get_line_corner_type_before_index(composite, index);
    }
```

db_interconnect_get_line_miter_radius_before_index()

Returns the miter cutoff percentage or curve radius for the interconnect segment before index mitered corner, or a curve radius in database units for a curved corner.

Syntax

```
decl miterRadiusPercnt =
db_interconnect_get_line_miter_radius_before_index(object, index);
```

where:

- object is interconnect composite object (multi segment trace).
- index is the point index before which line type is sought: 0 <= index <= Number of points.

Example

```
decl context = de_get_current_design_context();
decl compositeIter = db_create_composite_iter(context);
    for (; db_composite_iter_is_valid(compositeIter); compositeIter =
db_composite_iter_get_next(compositeIter))
    {
       decl composite = db_composite_iter_get_composite_object(compositeIter);
       decl index = 0; // where 0 <= index <= number of points
       decl line_type = db_interconnect_get_line_type_before_index(composite, index);
       decl miterRad = db_interconnect_get_line_miter_radius_before_index(composite,
index);
    }
```

Command Function

db_rotate_selected_objects()

Rotates all the selected objects in the given context.
Returns: TRUE or FALSE value.

Syntax

```
db_rotate_selected_objects(context, x, y, angle[, keepConnected[,
doAvoidanceRouting]]);
```

Where,

- *context* is a DesignContext returned from a function such as de_get_current_design_context().
- *x, y* is the point to rotate around.
- *angle* is the angle to rotate.
- *keepConnected* is an optional boolean TRUE or FALSE value that denotes if rotation should move wires to keep connectivity intact. This argument is FALSE by default if it is not provided.
- *doAvoidanceRouting* is an optional boolean TRUE or FALSE value that denotes if avoidance routing should be used when moving the object. This argument is FALSE by default if it is not provided.

Example

```
decl context = de_get_current_design_context();
db_select_all(context);
decl status = db_rotate_selected_objects(context, 0, 0, 45);
```

Via Functions

db_create_constraint_pcb_via()

Creates PCB Via from the constraint pcb via definition and place at location (x, y).

If pcb via definition does not have top/bottom layers set, means its an unconstrained pcb via, to create unconstrained pcb via use one of the following 3 functions: The constraint via definition must have a top layer and bottom layer constraint set.

- db_create_unconstrained_pcb_via_with_through_hole()
- db_create_unconstrained_pcb_via_with_top_bottom_layers()
- db_create_unconstrained_pcb_via_with_drill_layer()

Syntax

```
db_create_constraint_pcb_via(context, libraryName, viaName, x, y);
```

Where

- context of the design for the via or pad to be placed in.
- libraryName is the name of the library containing the padstack or via template.
- viaName is the name of PCB Via template that should be referenced when building a PCB via.
- x, y coordinates, in user units, where via should be placed in the design.

Example

```
 // Create a constraint pcb via from a viaM2M3 via definition with its defined top
and bottom layers specified in via definition at (0,0)
    decl context = de_get_current_design_context();
    decl via = db_create_constraint_pcb_via(context, "MyLib", "viaM2M3", 0, 0);
```

db_create_unconstrained_pcb_via_with_through_hole()

Creates a pcb via with a through hole from the unconstrained pcb via definition and place at location (x, y).

Syntax

```
db_create_unconstrained_pcb_via_with_through_hole(context, libraryName, viaName,
x, y);
```

Where,

- *context* of the design for the via or pad to be placed in.
- *libraryName* is the name of the library containing the via definition.
- *viaName* is the unconstrained pcb via definition name.
- *x, y* coordinates, in user units, where via should be placed in the design.

Example

```
// Create a through hole via from a via1 unconstrained via definition at (0,0)
    decl context = de_get_current_design_context();
    decl via = db_create_unconstrained_pcb_via_with_through_hole(target, "MyLib",
"via1", 0, 0);
```

db_create_unconstrained_pcb_via_with_top_bottom_layers()

Places a via going from top layer to bottom layer at location (x, y), using the unconstrained via definition between the respective layers.

Syntax

```
db_create_unconstrained_pcb_via_with_top_bottom_layers(context, libraryName,
viaName, topLayerID, bottomLayerID, x, y[, minimizeNumberOfDrills]);
```

Where,

- *context* of the design for the via to be placed in.
- *libraryName* is the name of the library containing the via definition.
- *viaName* is the unconstrained pcb via definition name.
- *topLayerID* is the top layer of via being created.
- *bottomLayerID* is the bottom layer of via being created.
- *minimizeNumberOfDrills* if TRUE, the pad will be created with the smallest number of drills that the substrate will allow. If FALSE, it will use the largest number of drills allowed.
- *x, y* coordinates, in user units, where via should be placed in the design.

Example

```
 // Create a via from a via1 unconstrained via definition between top(3) and
bottom(4) layers specified at (0,0)
     decl context = de_get_current_design_context();
     decl via = db_create_unconstrained_pcb_via_with_top_bottom_layers(context,
"MyLib", "via1", db_layerid(3), db_layerid(4), 0, 0);
```

db_create_unconstrained_pcb_via_with_drill_layer()

Places a via at location (x, y), using the drill layer passed in and using the unconstrained via definition.

Syntax

```
db_create_unconstrained_pcb_via_with_drill_layer(context, libraryName, viaName,
drillLayerID, x, y);
```

Where

- *context* of the design for the via to be placed in.
- *libraryName* is the name of the library containing the via definition.
- *viaName* is the unconstrained pcb via definition name.
- *drillLayerID* is the drill layer to place pad on.
- *x, y* coordinates, in user units, where via should be placed in the design.

Example

```
 // Create a via from a via1 unconstrained via definition with drill layer (8)
specified at (0,0)
decl context = de_get_current_design_context();
decl via = db_create_unconstrained_pcb_via_with_drill_layer(context, "MyLib", "via1",
db_layerid(8), 0, 0);
```

www.keysight.com