GenSys®
New Capabilities

Version O-2018.06, June 2018

SYNOPSYS®

Copyright Notice and Proprietary Information

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at http://www.synopsys.com/Company/Pages/Trademarks.aspx.
All other product or company names may be trademarks of their respective owners.

Free and Open-Source Software Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
690 E. Middlefield Road
Mountain View, CA 94043
www.synopsys.com

Contents

1

GenSys® M-2017.03-SP1 New Capabilities

GenSys® L-2016.06-SP1 has the following new capabilities:

- GenSys now enables you to modify the array-of-instances, generate if, and generate-for constructs such that the RTL abstractions of these constructs are not lost in the generated RTL.

  For details on this feature, see Editing generate Blocks and Array-Of-Instances.

- GenSys now generates a report of unsupported constructs present in an RTL. On identifying such constructs, you can take corrective actions, such as removing such constructs or stopping the modules containing such constructs. This ensures that the generated RTL is correct.

  For details, see The RTL Profiling Feature.

Editing generate Blocks and Array-Of-Instances

While modifying an RTL by using GenSys, the designer may need to replace third-party memories with an internal memory. The RTL for memories is usually present within the generate if, generate-for, or array-of-instances RTL statements. Each memory has a master associated.

To replace a third-party memory instance in these RTL statements with an internal memory, you need to update the master of that memory instance with a new master, which is the master of an internal memory. For details, see Updating the Master of Instances.

Once you update master, some additional pins get generated on the *generate* or array boundaries. You can tieoff such hanging pins or export them to the design boundary. For details, see Adding Tieoff Connections to Instances and Exporting Instance Pins.

You can also delete any unwanted connections from memory pins. For details, see Exporting Complex Multi-Instance Pins.

Prerequisites to Edit Generate Blocks and Array-Of-Instances

To allow editing of these structures, set the `-preserve_generate` argument of the `set_rtlimport_options` Tcl command to `TRUE`.

Multi-Instance

Within generate-for and array-of-instances structures, various instances are controlled by a loop variable. GenSys clubs all such instances in one group known as multi-instance.

When you make any change to a multi-instance, GenSys automatically applies that change to all the instances within that multi-instance. GenSys does not allow you to edit any individual instance within a multi-instance.

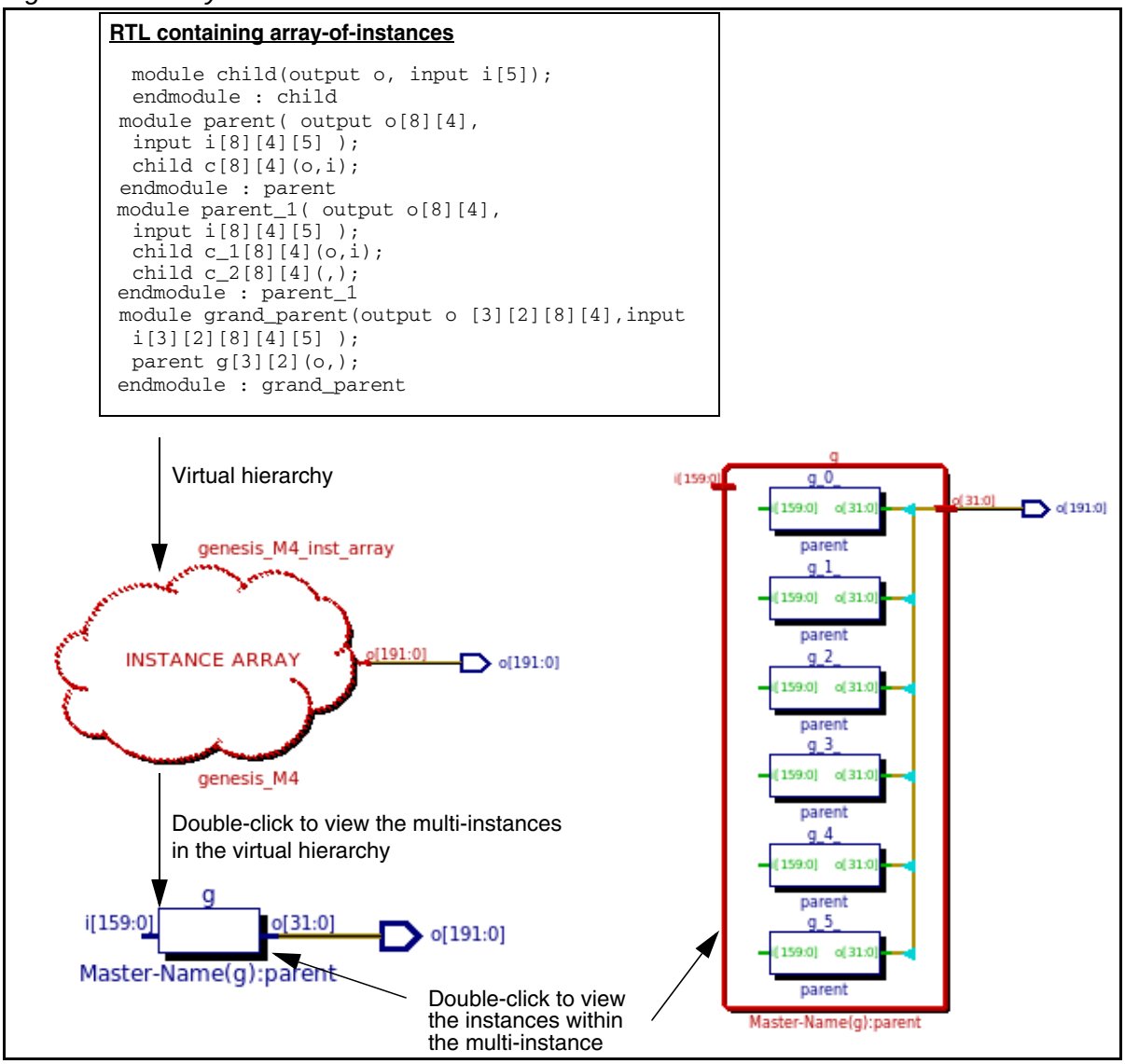
array-of-instances

An array-of-instances appears in a cloud-like virtual hierarchy. Double-click on the cloud to view the Multi-Instance, which is the consolidated view of all the instances inside the array. Double click the Multi-Instance to view all the instances in a multi-instance.

Any update made to a Multi-Instance automatically updates all the instances within that hierarchy. You cannot edit individual instances under a Multi-Instance.

See the following figure showing a multi-instance and its instances:

Figure 1-1 Array-of-instances

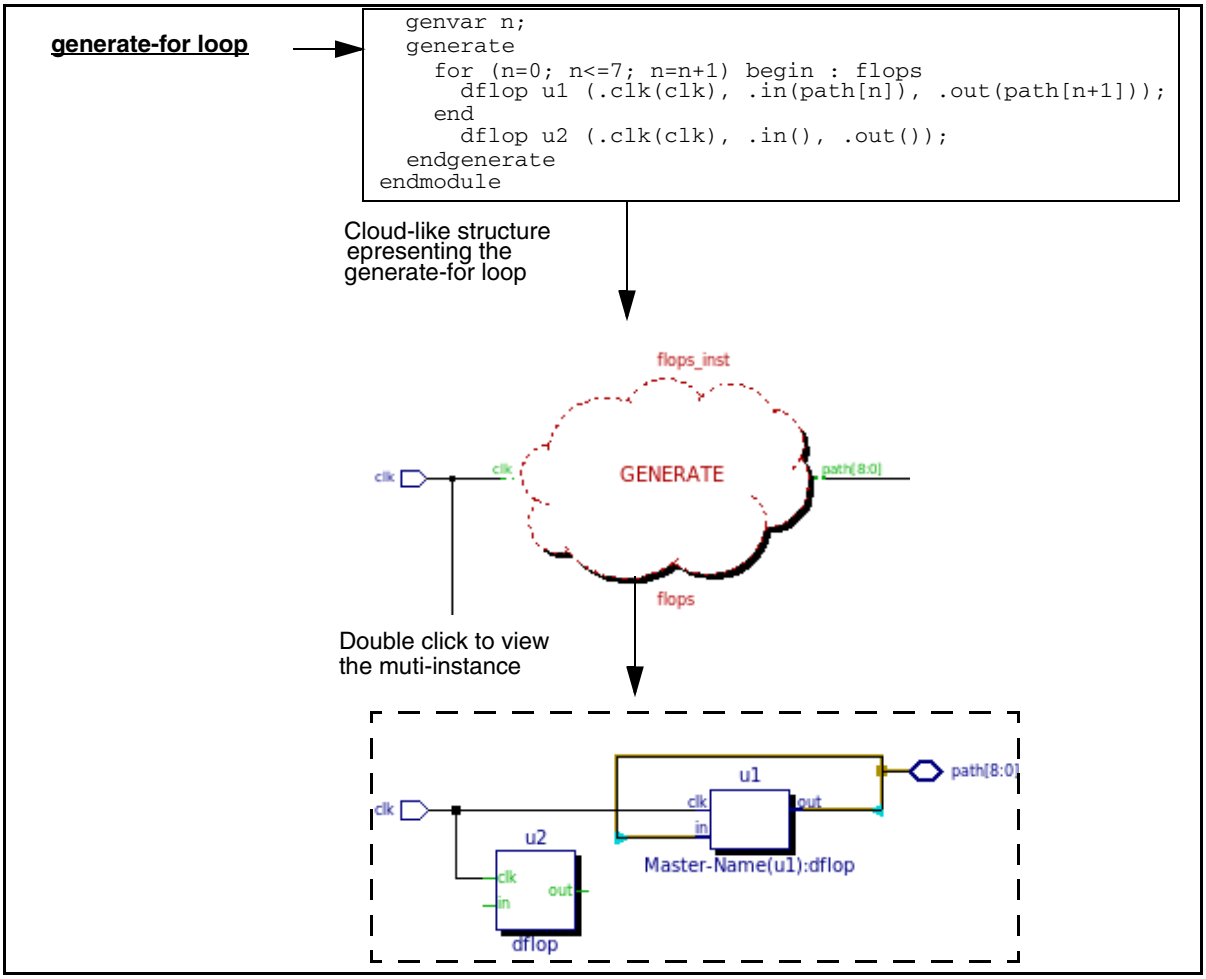


generate-for

Instances generated by a *generate-for* loop appear in a cloud-like virtual hierarchy. Double-click on this hierarchy to view the Multi-Instance, which is the consolidated view of all the instances generated by the *generate-for* loop.

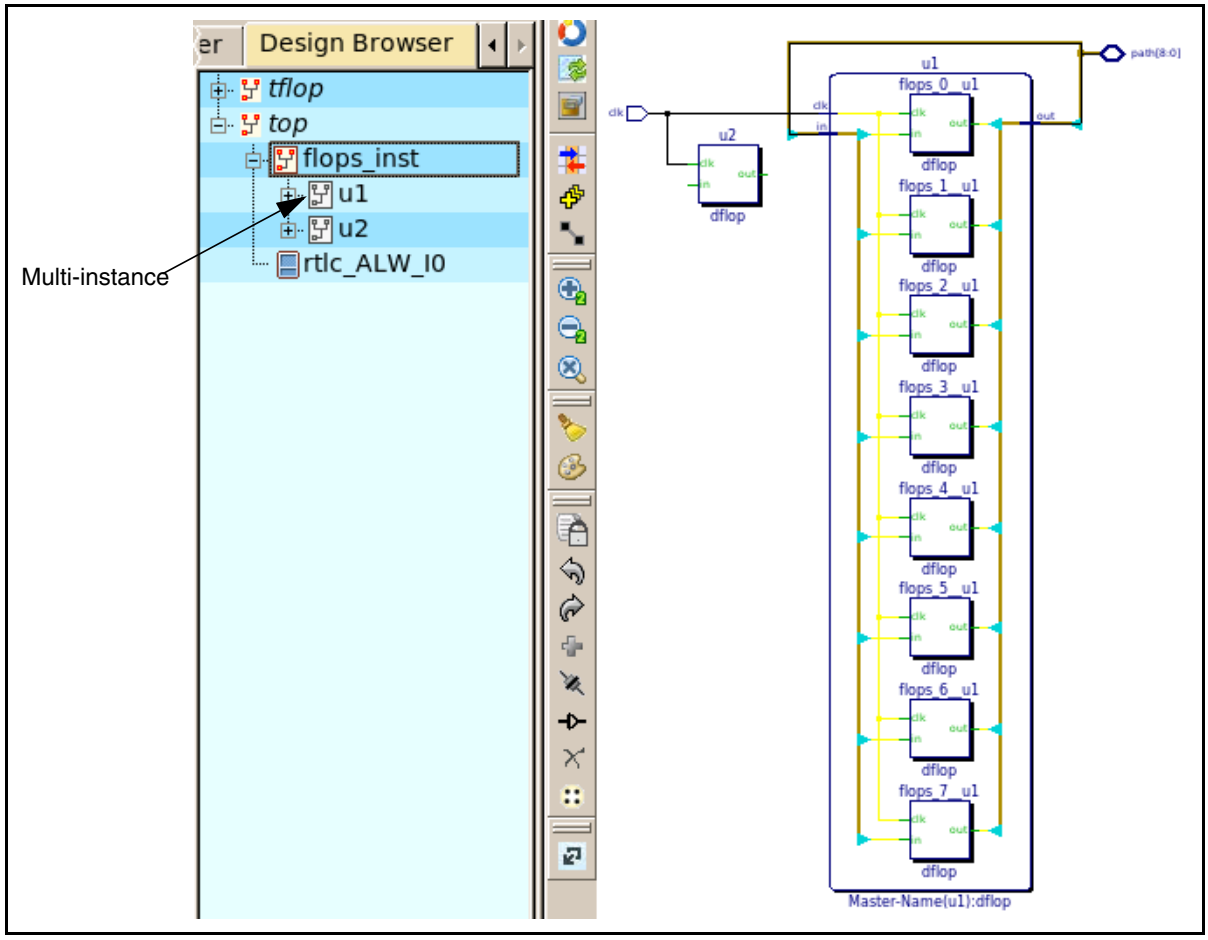
Any update made to a Multi-Instance automatically updates all the instances within that hierarchy. You cannot edit individual instances under a Multi-Instance. See the following figure showing a multi-instance:

Figure 1-2 generate-for loop



When you double-click on a Multi-Instance, GenSys shows all the instances within that Multi-Instance. This is shown in the following figure:

Figure 1-3 Expanded generate-for loop



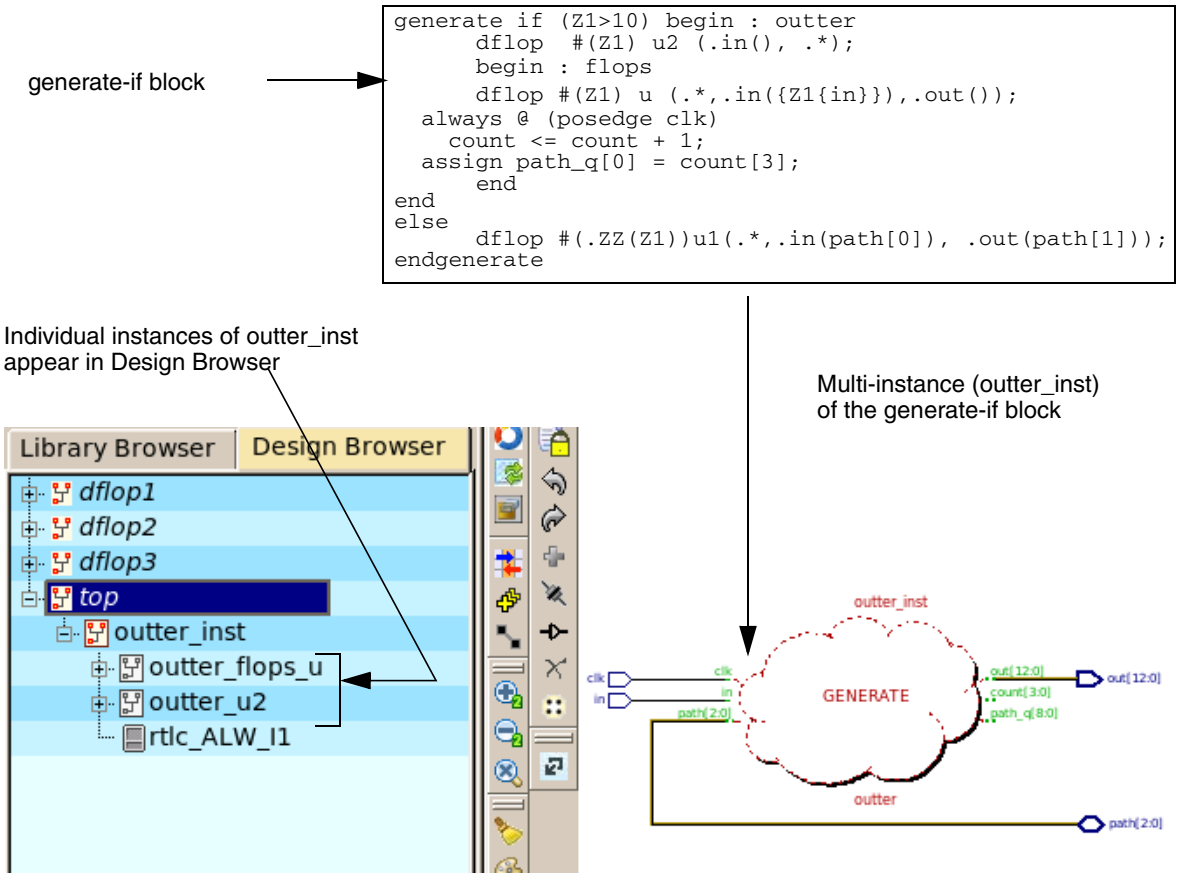
generate if

Instances generated by a *generate-if* loop appear in a cloud-like virtual hierarchy. When you double-click on this hierarchy, all the instances of that hierarchy appear in the schematic.

In addition, the *Design Browser* shows all the instances of a *generate-if* loop. The condition under which an instance is visible is shown in *the Generate-If Condition* row of the *Property Editor* pane.

The following figure shows the instances of a *generate-if* block:

Figure 1-4 generate-if block example



In the above example, if you double-click on the virtual hierarchy (`outter_inst`), all the instances within `outter_inst` appear that you can edit individually.

Updating the Master of Instances

The update master operation replaces the instance master with a new master. This operation is useful for memory swapping when you want to replace a third-party memory with a different memory.

For an array-of-instances and a generate-for loop, this operation updates the master of all the instances inside a Multi-Instance. You cannot perform this operation on individual instances.

However, for the generate if loop, you can update the master of individual instances.

You can update master in any of the following ways:

- Tcl

  Use the `update_master` Tcl command to specify the details of the new master.

- GUI

  Right-click on the instance and select the Update Master option from the shortcut menu. The Update Masters dialog appears in which you can specify the VNLV information of the new master.

  For information on this dialog, refer to the *GenSys User Guide*.

See Example of Updating the Master of a generate-for Multi-Instance.

Example of Updating the Master of a generate-for Multi-Instance

Consider the following RTL specified as input to GenSys:

```
module dflop(input clk,
                input in,
                output reg out);
always@(posedge clk)
        out<=in;
endmodule
module tflop(input clk,
                input in,
                output reg out);
always@(posedge clk)
        out<=~in;
endmodule
module top (clk, out);
  input         clk;
  reg     [3:0] count ;
  wire    [8:0] path;
  output        out;
  always @ (posedge clk)
    count <= count + 1;
  assign path[0] = count[3];
  assign out = path[8];
  genvar n;
  generate
    for (n=0; n<=7; n=n+1) begin : flops
      dflop u1 (.clk(clk), .in(path[n]), .out(path[n+1]));
    end
      dflop u2 (.clk(clk), .in(), .out());
  endgenerate
endmodule
```

In the above RTL, the *generate-for* loop creates the `u1` Multi-Instance that contains eight instances. To update the master of `u1`, perform the following steps:

1. Specify the following Tcl command:

   ```
   set_rtlimport_option -preserve_generate TRUE
   ```

2. Load the RTL in GenSys by specifying the following Tcl command:

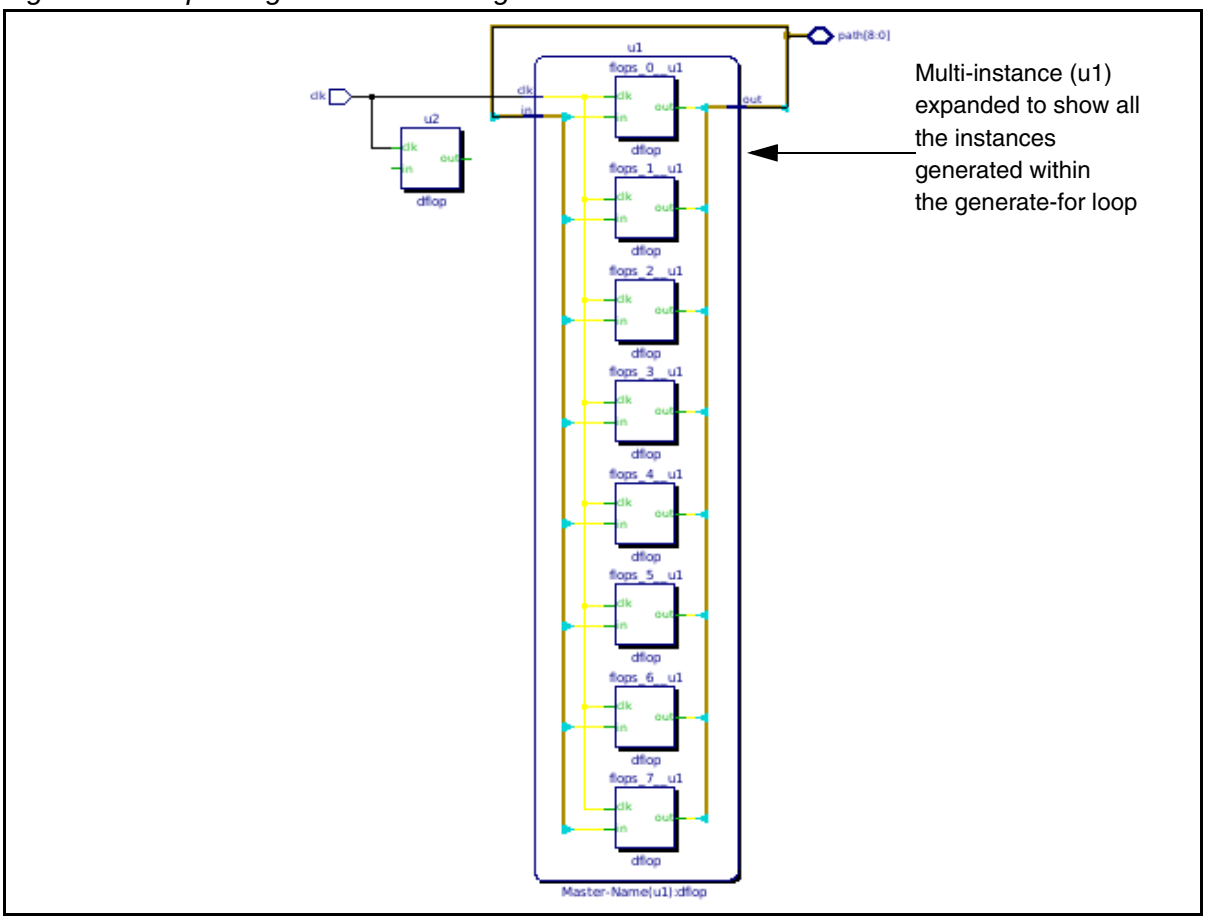
   ```
   load_rtl input.v
   ```

3. In the *Design Browser*, double-click on the `top` design to select this design. Alternatively, specify the following Tcl command:

   ```
   select_design top
   ```

   The following figure shows the expanded multi-instance (`u1`) in the `top` design:

Figure 1-5 Updating the master of a generate-for multi-instance



4. To update the master of the `u1` Multi-Instance, specify the following Tcl command:

   ```
   update_master -instance u1 -name tflop
   ```

   This step updates the master from `u1` to `tflop` for all the instances within `u1`.

5. Generate Verilog by specifying the following Tcl command:

   ```
   run GenerateVerilog
   ```

After performing the above steps, the generated Verilog contains the update master changes.

The generated Verilog is as follows:

```
module dflop(input clk,                input in,                output
reg out);always@(posedge clk)        out<=in;endmodulemodule tflop(input
clk,                input in,                output reg
out);always@(posedge clk)      out<=~in;endmodulemodule top (clk,
out);input           clk;output          out;//reg [3:0] count;wire [8:0]
path;////genvar n;////  generate   for (n=0; n<=7; n=n+1) begin : flops
tflop u1 (.clk(clk), .in(path[n]), .out(path[n+1]));    end      dflop u2
(.clk(clk), .in(), .out());  endgenerate////  always @ (posedge clk)
count <= count + 1;  assign path[0] = count[3];//endmodule
```

Adding Tieoff Connections to Instances

GenSys allows adding connections of the tieoff type for array-of-instances and instances in generate-for and generate if.

When you add a tieoff connection on a Multi-Instance pin, GenSys automatically creates that connection from all the instances present inside that Multi-Instance.

For an array-of-instances and a generate-for loop, you cannot add tieoff connections to individual instances. However, for a generate if loop, you can add tieoff connections on selected instances.

You can add tieoff connections in any of the following ways:

- Tcl

  Use the `add_connection` Tcl command specified with the `-tieoff` argument. If you specify any other connection type other than tieoff, GenSys reports an error.

  For details on this command, refer to *GenSys Tcl Commands Reference Guide*.

- GUI

  To add a tieoff connection in GUI, perform the following steps:

  1. Right-click on the instance pin, and select the Add Tie-off option from the shortcut menu.

The Add Tie-off dialog appears, as shown in the following figure:

Figure 1-6 The Add Tie-off dialog

2. In the Add Tie-off dialog, specify a tieoff value.
3. Click Add.

See Example of Adding a Tieoff Connection from a Multi-Instance.

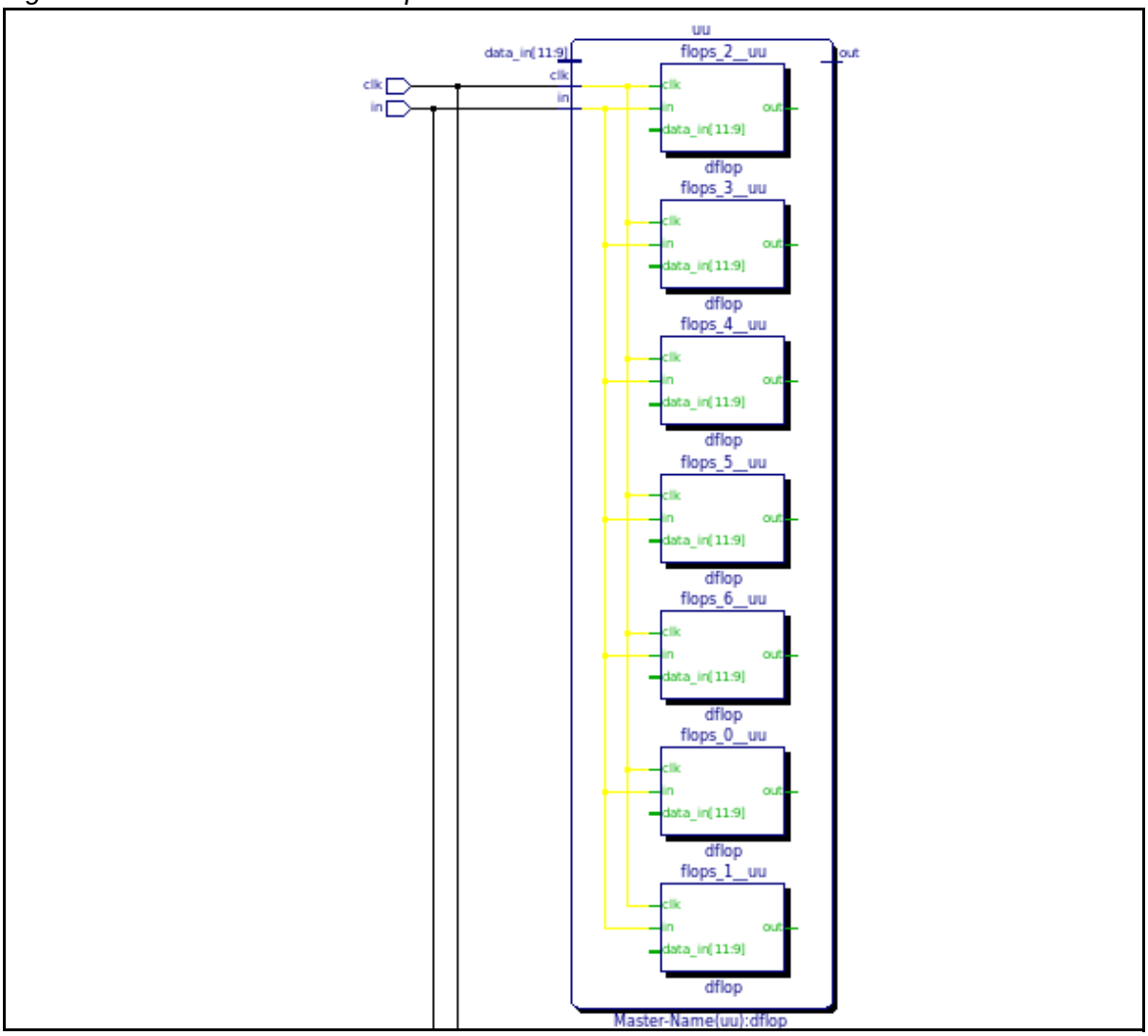
Example of Adding a Tieoff Connection from a Multi-Instance

Consider the following RTL specified as input to GenSys:

```
module top (clk, out,in);  input         clk,in;  reg     [3:0] count =
0;  wire    [8:0] path;  output        out;wire [7*2-1:0] data_in;
always @ (posedge clk)    count <= count + 1;  genvar n;  generate    for
(n=0; n<7; n=n+1) begin : flops      dflop u (.clk(clk), .in(in),
.out(path[n+1]),.data_in(data_in[2*n+1:2*n]));      dflop uu (.clk(clk),
.in(in), .out(),.data_in());    end  endgenerateendmodulemodule
dflop(input clk,  input in,  output reg out,input [11:9]data_in);
always@(posedge clk)  out<=in;endmodule
```

In the above RTL, the *generate-for* loop generates two Multi-Instances, u and uu. The following figure shows the uu Multi-Instance containing multiple instances generated by the *generate-for* loop:

Figure 1-7 Multi-instance example



In the above `uu` Multi-Instance, to create a tieoff connection from the `data_in` pin, perform the following steps:

1. Specify the following Tcl command:

   ```
   set_rtlimport_option -preserve_generate TRUE
   ```

2. Load the RTL in GenSys by specifying the following Tcl command:

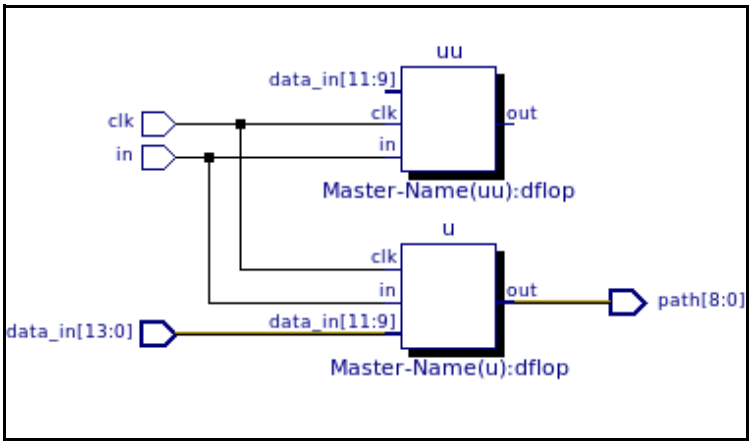
   ```
   load_rtl input.v
   ```

3. Specify the following Tcl commands to select the `flops` design containing the `uu` Multi-Instance:

   ```
   select_design top
   select_design flops
   ```

   The following schematic show the selected `flops` design in which the `u` and `uu` Multi-Instance is present:

Figure 1-8 Schematic of a multi-instance



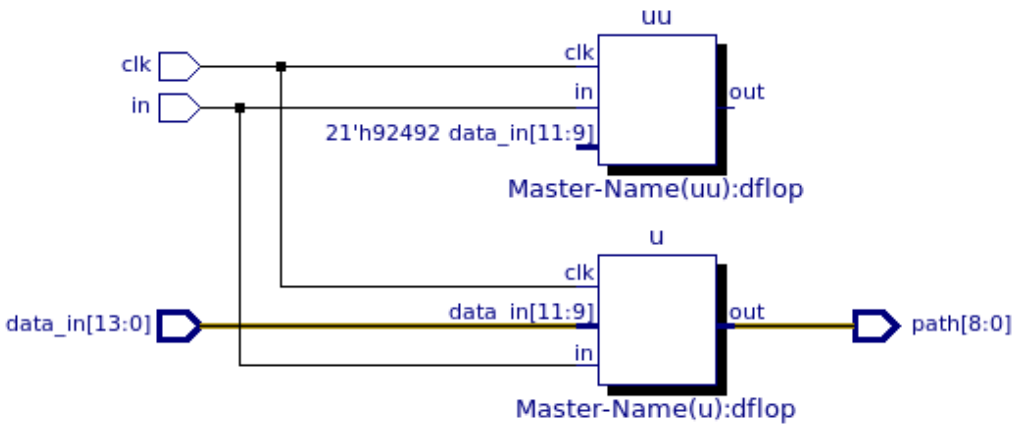
   The expanded view of `uu` is shown in Multi-instance exampleFigure 1-7.

4. Create a tieoff connection from the `uu` Multi-Instance by specifying the following Tcl command:

   ```
   add_connection -instance uu -pin {data_in} -tieoff 2
   ```

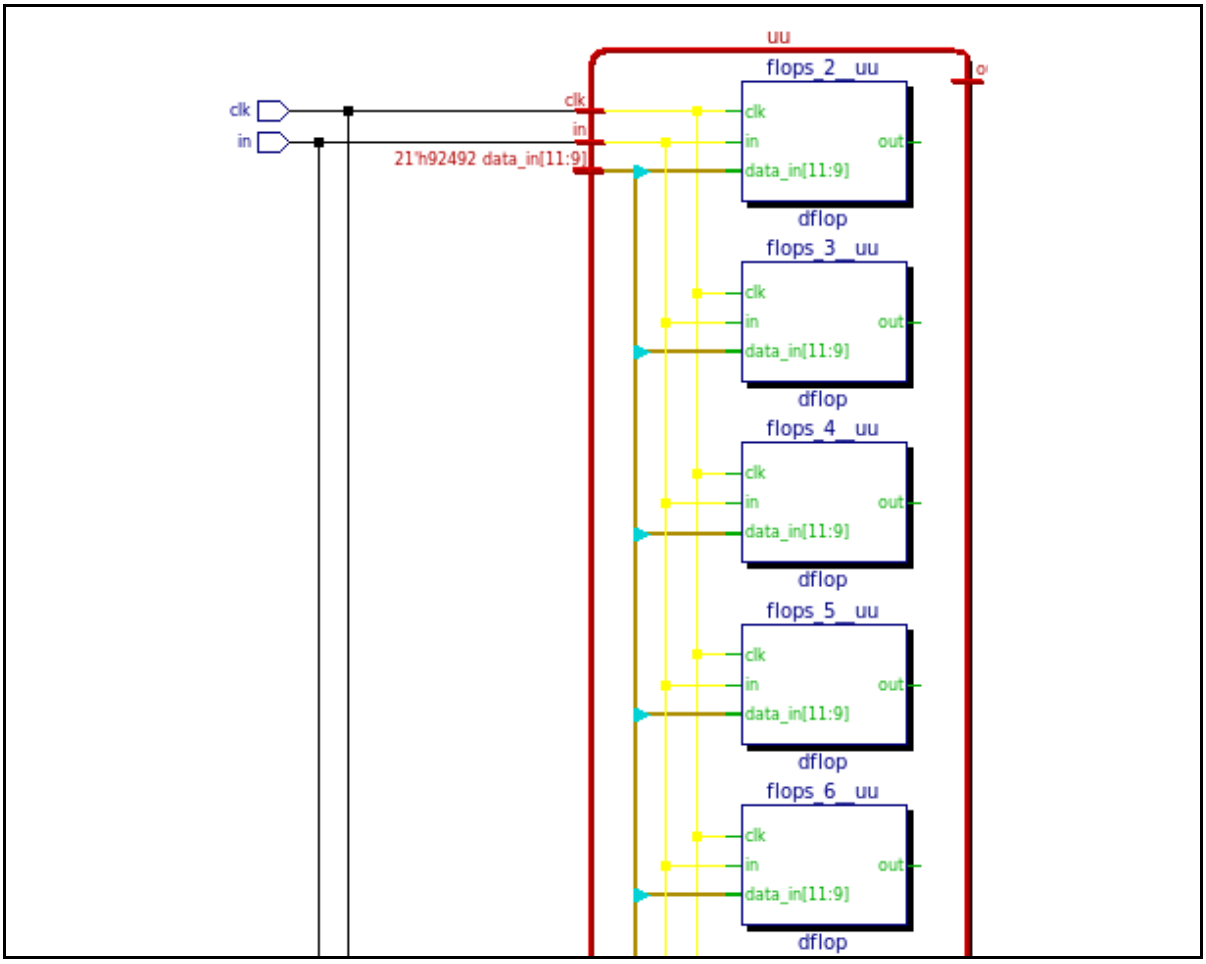
   After performing this step, a tieoff connection appears in the `data_in` pin, as shown in the following figure:

Figure 1-9 Tie-off added to the multi-instance



If you expand the uu group by double-clicking it in the schematic, you will notice that a tieoff connection is created from the `data_in` pins of all the instances present in the `uu` Multi-Instance. This is shown in the following figure:

Figure 1-10 Expanded multi-instance with tie-off connections



Deleting Connections from Instances

When you delete a connection from a Multi-Instance pin, GenSys automatically deletes that connection from all the instances present inside that Multi-Instance.

For an array-of-instances and a generate-for loop, you cannot delete connections from individual instances. However, for a generate-if loop, you can delete connections from selected instances.

You can delete connections in any of the following ways:

- Tcl

  Use the `delete_connection` Tcl command. For information on this command, refer to the *GenSys Tcl Commands Reference Guide*.

- GUI

  Right-click on an instance pin and select the Delete Connected Net option from the shortcut menu.

Exporting Instance Pins

To export a pin from a Multi-Instance to the design boundary, use the following Tcl command:

```
export_pin -instance <multi-instance-name> -pin <pin-name>-export_name
<exported-pin-name> -loop_expr <expression>
```

The arguments of the above command are explained as follows:

- *-loop_expr <expression>*: This is the new argument.

  Use this argument to export complex Multi-Instance pins. For details, see Exporting Complex Multi-Instance Pins.

- *-instance <multi-instance-name>*: Specifies the name of the Multi-Instance.
- *-pin <pin-name>*: Specifies the Multi-Instance pin to be exported.
- *-export_name <exported-pin-name>*: Specifies the name of the pin by which it should be exported.

Any argument other than the above-mentioned arguments are invalid for a Multi-Instance. If you specify any invalid argument, GenSys reports an error.

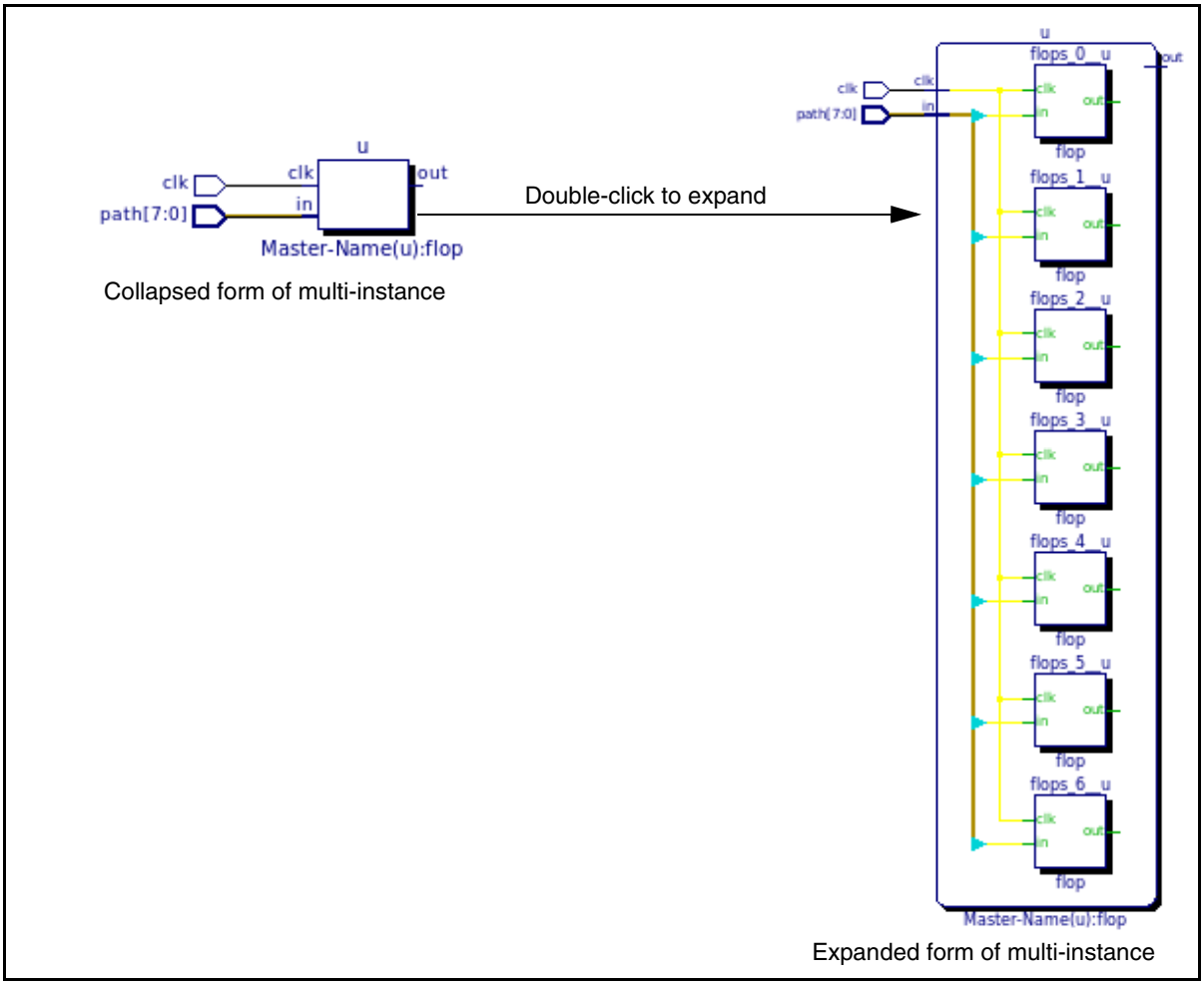
Example of Exporting a Multi-Instance Pin

Consider the following RTL specified as input to GenSys:

```
module top #(parameter Z0 = 0,parameter Z1 = 15,parameter Z2 = 5
)(clk,reset, out,in);  input         clk,reset;input [Z1:0] in;  output
logic [Z1*7-1:0] out [2] ;  genvar l,n,m;  wire [7:0] path;  generate
for (l=0; l<7; l=l+1) begin : flops      flop u (.clk(clk), .in(path[l]),
.out());    end  endgenerate  generate    for (n=0; n<7; n=n+1) begin :
dflops      dflop #(.Z0(Z0),.Z1(Z1))u1 (clk, in,
{out[0][Z1*n+7:Z1*n]});    end    for (m=0; m<7; m=m+1) begin : tflops
tflop #(.Z0(Z0),.Z1(Z1))u2 (clk,in,      out[1][Z1*m+7:Z1*m]);    end
endgenerateendmodulemodule dflop#(parameter Z0 = 0,parameter Z1 =
12)(input clk,input [Z1:Z0] in,output reg [Z1:Z0] out);always@(posedge
clk)out<=in;endmodulemodule tflop#(parameter Z0 = 0,parameter Z1 =
12)(input clk,input [Z1:Z0] in,output reg [Z1:Z0] out);always@(posedge
clk)out<=~in;endmodulemodule flop #(parameter Z1 = 12)(input clk,input
in,output reg out);always@(posedge clk)out<=in;endmodule
```

For the above example, the following figure shows the schematic of the u Multi-Instance and its expanded form when you double-click on u:

Figure 1-11 Multi-instance



In the above `example`, to export the `out` pin of the u Multi-Instance so that the `ex_output` pin appears on the design boundary, perform the following steps:

1. Specify the following Tcl command:

   ```
   set_rtlimport_option -preserve_generate TRUE
   ```

2. Load the RTL in GenSys by specifying the following Tcl command:

   ```
   load_rtl input.v -enableSV
   ```

3. Specify the following Tcl commands to select the `flops` design containing the u Multi-Instance:

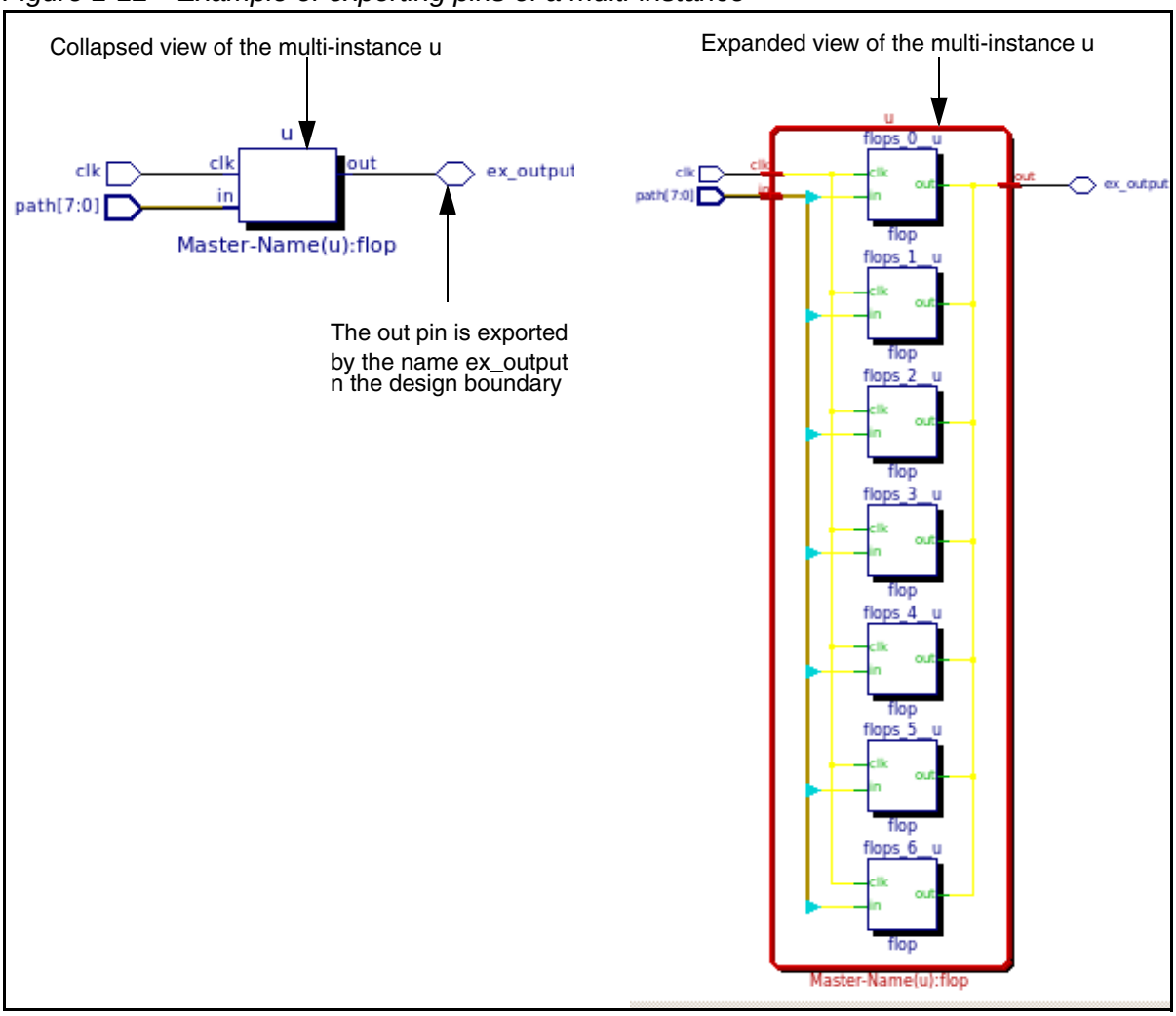
   ```
   select_design topselect_design flops
   ```

4. Export the `out` pin by specifying the following Tcl command:

   ```
   export_pin -instance u -pin out -export_name ex_output
   ```

   After performing this step, the out pin of the `u` Multi-Instance is exported by the name `ex_output` on the design boundary. This is shown in the following figure:

Figure 1-12 Example of exporting pins of a multi-instance



5. Generate Verilog by using the following Tcl command:

   ```
   run GenerateVerilog
   ```

After performing the above steps, GenSys generates a Verilog file in the *<curr-work-dir>/output/Verilog* directory. This file preserves the changes made to the `u` Multi-Instance.

The following Verilog file is generated for this example:

```
module top #(parameter    Z0   =    0,parameter    Z1   =    15,parameter
Z2   =    5)          (clk  ,           reset,           out  ,
in);input             clk;input             reset;output logic
[((Z1*7)-1):0] out [0:(2-1)]  ;input  wire [Z1:0]          in
;wire    [7:0]     path;wire             ex_output;//genvar l;genvar
n;genvar m;////  generate    for (l=0; l<7; l=l+1) begin : flops
flop u (.clk(clk), .in(path[l]), .out( ex_output));    end  endgenerate
//
//
  generate
    for (n=0; n<7; n=n+1) begin : dflops
      dflop #(.Z0(Z0),.Z1(Z1))u1 (clk, in,
      {out[0][Z1*n+7:Z1*n]});
    end
    for (m=0; m<7; m=m+1) begin : tflops
      tflop #(.Z0(Z0),.Z1(Z1))u2 (clk,in,
      out[1][Z1*m+7:Z1*m]);
    end
  endgenerate
//
endmodule
module dflop#(parameter Z0 = 0,
parameter Z1 = 12)(input clk,
input [Z1:Z0] in,
output reg [Z1:Z0] out
);
always@(posedge clk)
out<=in;
endmodule
module tflop#(parameter Z0 = 0,
parameter Z1 = 12)(input clk,
input [Z1:Z0] in,
output reg [Z1:Z0] out
);
always@(posedge clk)
out<=~in;
endmodule
module flop #(parameter Z1 = 12)(input clk,input in,output reg
out);always@(posedge clk)out<=in;endmodule
```

Exporting Complex Multi-Instance Pins

To connect complex multi-instance pins inside generate-for or array-of-instance, you must first export such pins on the generate-block boundary by using the `export_pin` Tcl command.

For such pins, you must specify an RTL expression by using the `-loop_expr` argument of the `export_pin` Tcl command. This is the new argument added in the 5.5 release. The RTL expression comprises of:

- The signal name.

- The `genvar` variables controlling the generate-for loop.

See Example of Specifying RTL Expression in -loop_expr.

If you do not use the `-loop_expr` argument, the corresponding pin of all the instances present inside Multi-Instance get connected to the same exported pin. This might be the desired behavior for clock pins but not for other pin types.

After exporting pins, you can create connections with the exported pins by using the `add_connection` Tcl command. For details on this command, refer to the *GenSys Tcl Commands Reference Guide*.

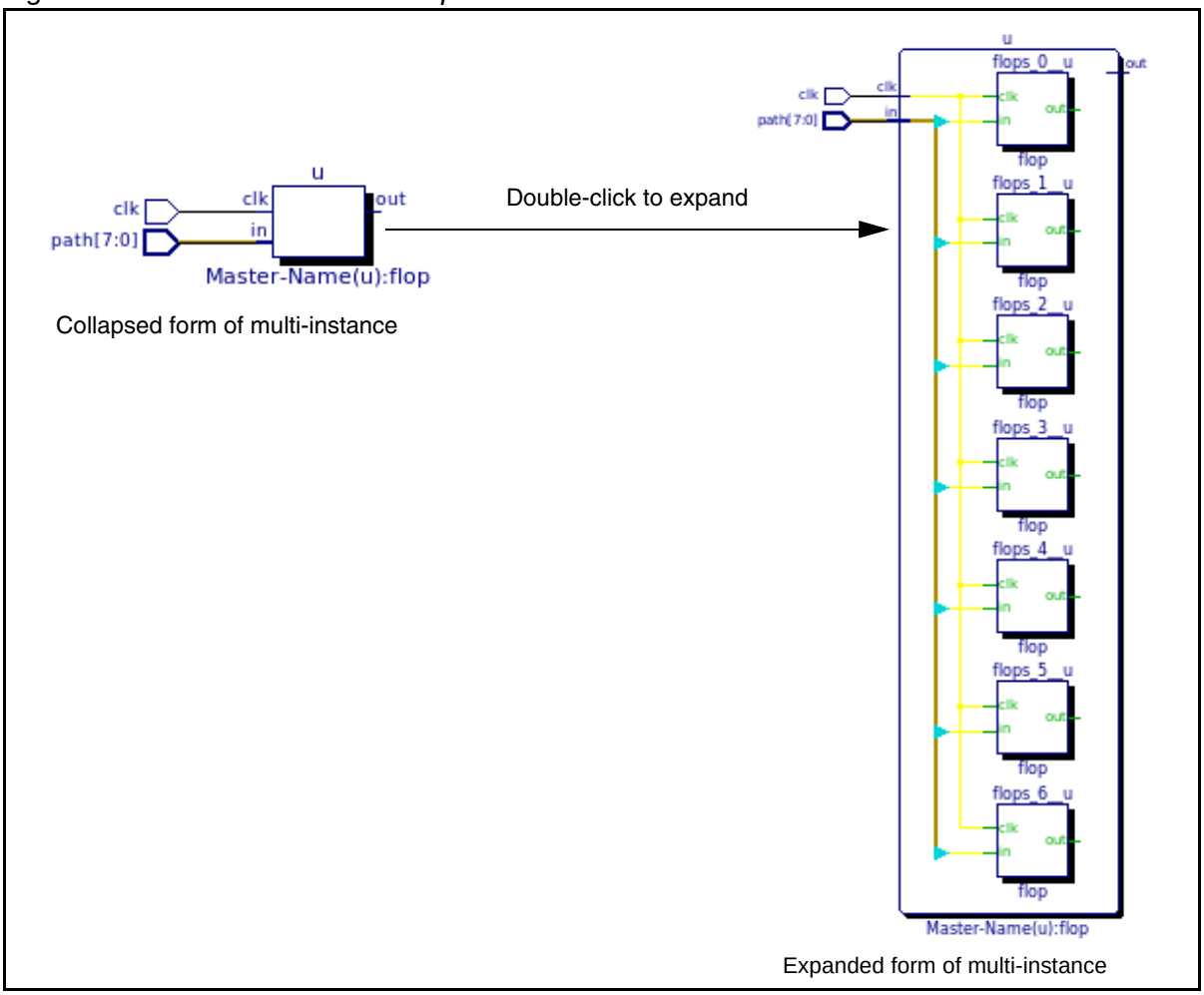
Example of Specifying RTL Expression in -loop_expr

Consider the following RTL snippet:

```
genvar l,n,m;  wire [7:0] path;  generate    for (l=0; l<7; l=l+1) begin
: flops      flop u (.clk(clk), .in(path[l]), .out());    end
endgenerate
```

The *for* loop in the above RTL generates eight instances that form the `u` Multi-Instance group, as shown below:

Figure 1-13 Multi-instance example



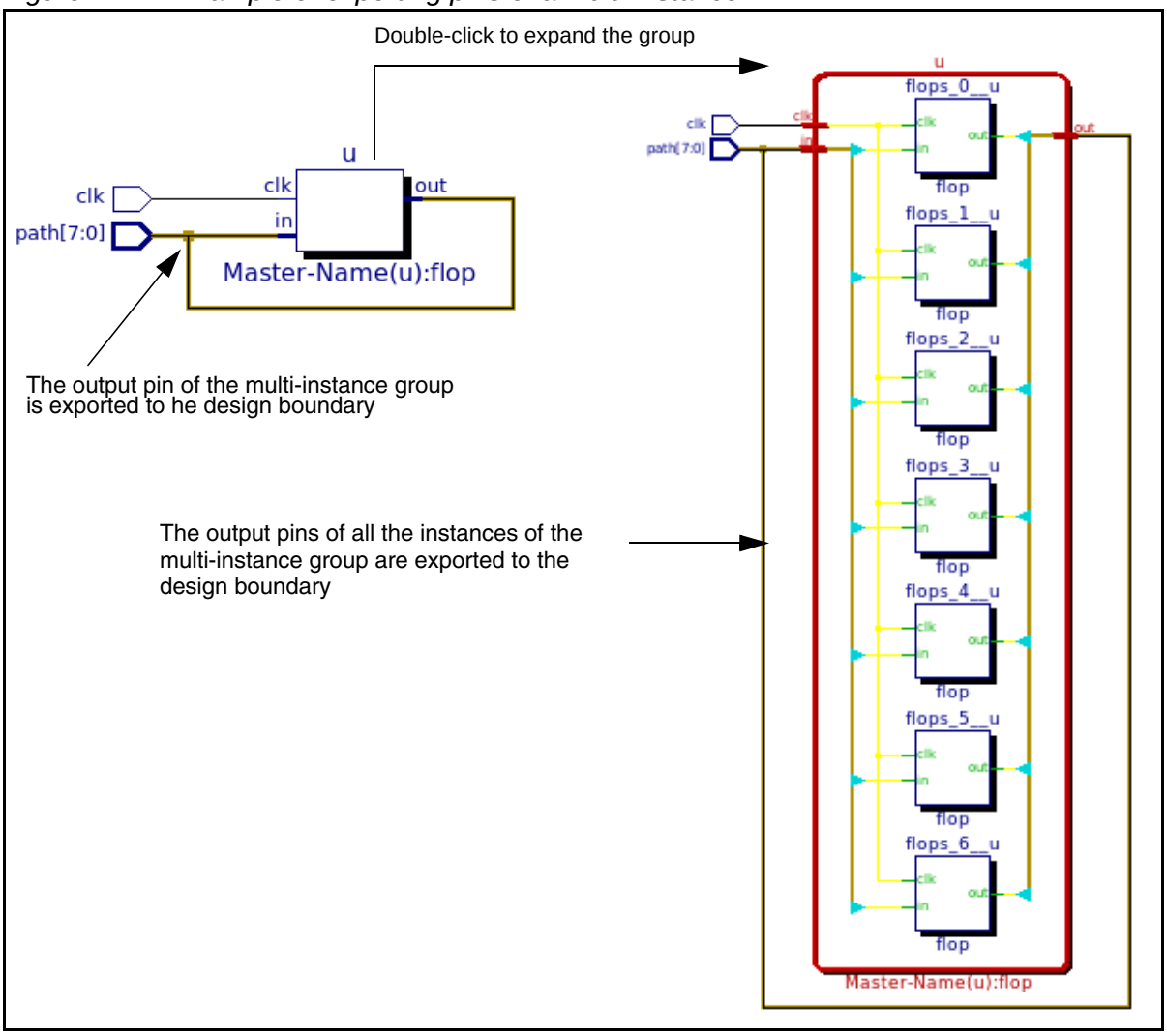
In the above example, to export the `out` pin of the `u` Multi-Instance, specify the expression `{path[l+1]}` to the `-loop_expr` argument, as shown in the following Tcl command:

```
export_pin -instance u -pin out -loop_expr {path[l+1]}
```

Note that expression in the above command contains the RTL variables, `path` and `l`.

The out pin gets exported to the design boundary, as shown in the following figure:

Figure 1-14 Example of exporting pins of a multi-instance



The RTL Profiling Feature

GenSys is an RTL modification and creation engine in which you read the RTL (`load_rtl`), perform RTL modifications, and then generate the modified RTL.

However, the modified RTL may not be correctly generated due to the presence of some unsupported constructs in the original RTL.

Therefore, the need is to report such constructs so that the user performs the required actions, such as remove such constructs or stop the modules containing such constructs.

This section describes the constructs unsupported in GenSys and the RTL profiler report that shows information about these constructs. For details, see Unsupported Constructs in GenSys and The RTL Profiler Report.

Unsupported Constructs in GenSys

This section describes the unsupported constructs in the VHDL, Verilog, and SystemVerilog flow. For details, see:

- Unsupported Constructs in the VHDL Flow
- Unsupported Constructs in the Verilog Flow
- Unsupported Constructs in the SystemVerilog Flow

Unsupported Constructs in the VHDL Flow

The following VHDL constructs used inside `process` blocks are supported, but unsupported if used outside the `process` blocks:

- Linkage ports in VHDL
- Usage of VHDL alias constructs
- Usage of VHDL extended identifier
- Usage of VHDL attributes
- Usage of unsynthesizable constructs based on keywords only
- Usage of VHDL configuration and if it has a specific binding, the usage becomes illegal
- Tieoff connections to non std-logic types
- Type casting across connections except for signed, unsigned, and std_logic_vector
- Generics with parameterized user defined types

Unsupported Constructs in the Verilog Flow

The following Verilog constructs are not supported in GenSys:

- Presence of `` `ifdef `` on `` `define `` constructs if the `-preserve_configurations` argument of `set_rtlimport_option` is set to `TRUE`.

  `` `ifdef `` on `` `define `` constructs are shown as unsupported in The RTL Profiler Report if it is present inside a module. Else, it will be shown as an audit information in this report. (See Example of ifdef on `define, `undef, `timescale)

- Presence of `ifdef on `undef constructs if the -preserve_configurations argument of set_rtlimport_option is set to TRUE. (See Example of ifdef on `define, `undef, `timescale)

  `ifdef on `undef constructs are shown as unsupported in The RTL Profiler Report if it is present inside a module. Else, it will be shown as an audit information in this report.

- Presence of `ifdef on `timescale, timeunit, and timeprecision constructs if the -preserve_configurations argument of set_rtlimport_option is set to TRUE. (See Example of ifdef on `define, `undef, `timescale)

  It is reported as unsupported in The RTL Profiler Report if it is present inside a module. Else, it will be shown as an audit information in this report.

- Presence of `ifdef on a complete module if the -preserve_configurations argument of set_rtlimport_option is set to TRUE (See Example of `ifdef on Complete Module)

- Presence of `ifdef on assertions, initial blocks, and unsynthesizable constructs if the -preserve_configurations argument of set_rtlimport_option is set to TRUE (See Example of `ifdef on Assertions, Initial Blocks, and Unsynthesizable Constructs)

- Presence of `ifdef on partial expressions/statements if the -preserve_configurations argument of set_rtlimport_option is set to TRUE.

  (See Examples of `ifdef on Partial Expressions/Statements)

- Presence of local `undef constructs (See Example of Local `undef Constructs)

- Presence of multiple definitions of the same macro (See Example of Multiple Definitions of the Same Macro)

- Presence of multiple `ifdef branches in the same line if the -preserve_configurations argument of set_rtlimport_option is set to TRUE.

  This is shown in Example 2. Also see Example 1.

- Presence of pragmas in the same line as RTL constructs

  This is shown in the following example:

  ```
  input a; /* synopsys translate_off */ initial begin a = 1; end /*
  synopsys translate_on */
  ```

  (Also see Example of Pragmas in the Same Line as RTL Constructs)

- Presence of RTL code for unsynthesizable constructs in same line as the macros.

  (See Example of RTL Code for Unsynthesizable Constructs in Same Line as Macros)

- Presence of the defparam RTL construct (See Example of defparam/Usage of real Keyword in AMS)

- Presence of gate primitives, such as `buf` and `bufif1` (See Example of Gate Primitives/Supply Net Types/Ports with Implicit Net Declarations)
- Presence of AMS constructs, such as the `real` keyword (See Example of defparam/Usage of real Keyword in AMS)
- Presence of escaped identifiers (See Example of Escaped Identifiers)
- Presence of `supply0` and `supply1` net types (See Example of Gate Primitives/Supply Net Types/Ports with Implicit Net Declarations)
- Presence of ports with implicit net declarations, as shown in the following example:

  ```
  module abc (in1 [1:0], out1);
  ```

  (Also see Example of Gate Primitives/Supply Net Types/Ports with Implicit Net Declarations)

Unsupported Constructs in the SystemVerilog Flow

The following SystemVerilog constructs are not supported in GenSys:

- Presence of `` `ifdef `` on SystemVerilog interfaces if the `-preserve_configurations` argument of `set_rtlimport_option` is set to `TRUE`.
- Presence of unsynthesizable SystemVerilog constructs
- Presence of the `extern` keyword in module definition
- Presence of nested modules and nested interfaces
- Presence of glue logic in interface definitions
- Presence of a parameter/macro of the same name inside the module definition as in the global scope

---

The RTL Profiler Report

The RTL profiler is a text report that shows the following information:

- The RTL constructs unsupported in GenSys
- Impact of GenSys switches on RTL constructs
- Information on various constructs present in the RTL

Generating the RTL Profiler Report

To generate this report, perform the following actions:

- Specify the `set_rtlprofiler_options` Tcl command as shown below:

```
set_rtlprofiler_options  -file <report-name>  -hierarchy <TRUE |
FALSE>  -report < all | unsupported | constructs>  -detail <TRUE |
FALSE>
```

  The details of the above arguments are as follows:

- `-file` <*report-name*>: Specifies the name of the RTL profiler report.
- `-hierarchy` <*TRUE* | *FALSE*>: Specifies whether the report is generated for the top-level design only or should be run hierarchically on all the modules.
- `-report` <*all* | *unsupported* | *constructs*>: Specify `all` to report all the design properties.

  Specify `unsupported` to report all the unsupported GenSys constructs. For information on these constructs, see Unsupported Constructs in GenSys.

  Specify `constructs` to report the constructs that can be useful in understanding the RTL and for which GenSys can suggest the recommended switches to be used. For information on these constructs, see Information on various constructs present in the RTL.

- `-detail` <*TRUE* | *FALSE*>: Set this argument to `FALSE` to report a concise view of information. To view the detailed information for better debugging, set this argument to `TRUE`.
- Set the `-profile_rtl` argument of the `load_rtl` Tcl command to `TRUE` to generate the report during RTL load.

The RTL constructs unsupported in GenSys

To know these constructs, see Unsupported Constructs in GenSys.

Impact of GenSys switches on RTL constructs

This section of the report provides the following information:

- Presence of `` `ifdef `` constructs in the RTL when the `-preserve_configurations` argument of `set_rtlimport_option` Tcl command is not set to `TRUE`.
- Presence of unsynthesizable constructs `-preserve_unsynth_constructs` argument of `set_gensys_options` Tcl command is not set to `TRUE`.
- Presence of pragmas and when the `-preserve_pragmas argument of set_rtlimport_option` Tcl command is not set to `TRUE`.

- Presence of glue logic and the usage of the `-stop_on_partial_glue_modules` or `-stop_on_complete_glue_modules` arguments of the `set_rtlimport_option` Tcl command.
- Presence of instances in a module when the number of modules is greater than what is specified in the `-instance_threshold` <*num*> argument of the `load_rtl` Tcl command or the default argument value.

Information on various constructs present in the RTL

This section of the report provides the information that helps the user to understand the RTL better:

- Presence of `` `ifdef ``: The report captures various `` `ifdef `` macros used, various `` `ifdef `` macro combinations, and the `` `ifdef `` macro combination per module. (See Example on the Usage of ‘ifdef and ‘define)
- Usage of `` `define ``: The report captures the location where the `` `define `` pre-processor directives are defined (file name), total count per file, and the total count per module. (See Example on the Usage of ‘ifdef and ‘define)
- Presence of parameters and generics: The report captures the total number of parameters/generics per module. (See Example on Generics and Local/Complex/String Parameters)
- Presence of string parameters (See Example on Generics and Local/Complex/String Parameters)
- Presence of parameters of complex types (anything that is not simple integer type) (See Example on Generics and Local/Complex/String Parameters)
- Number of local parameters (See Example on Generics and Local/Complex/String Parameters)
- Presence of parameterized multi-dimensional ports and wires (See Example of Parameterized Multi-Dimensional Ports/Wires)
- Presence of complex user-defined ports (See Example of Complex User-Defined Ports)
- Presence of glue logic in a module (See Example on the Presence of Generate Blocks)
- Count of generate blocks in a module (See Example on the Presence of Generate Blocks)
- Count of generate-if, for, and case constructs in a module (See Example on the Presence of Generate Blocks)
- Presence of array-of-instances (See Example on the Presence of Array-of-Instances)

- Partial assignments inside glue logic or generate logic (See Example of Partial Assignments in Glue/Generate Logic)
- Presence of VHDL modules with complex types instantiated inside Verilog or SystemVerilog modules (See Example Of VHDL Modules of Complex Types)
- Presence of `define macros in any place except port msb/lsb, parameter value, parameter map, port map, glue logic, and instance master name
- Presence of local `define constructs (See Example of Local ‘define Constructs)

Examples of Constructs to Help Better Understand the RTL

This section covers the following examples:

- Example on the Usage of ‘ifdef and ‘define
- Example on Generics and Local/Complex/String Parameters
- Example of Parameterized Multi-Dimensional Ports/Wires
- Example of Complex User-Defined Ports
- Example on the Presence of Generate Blocks
- Example on the Presence of Array-of-Instances
- Example of Partial Assignments in Glue/Generate Logic
- Example Of VHDL Modules of Complex Types
- Example of Local ‘define Constructs

Example on the Usage of ‘ifdef and ‘define

```
`ifdef x`define y 10`else`define Y 20`endif module test();parameter P1 =
`ifdef Z 10 `else 20 `endif ;`ifndef Wparameter P2 =10;`endif`ifdef
MM`ifdef NNparameter R = 10;`endif`endif`define HH 10`define GG
20endmodule
```

Example on Generics and Local/Complex/String Parameters

```
`define Y 2`include "inc.v"module test();`define W 20`include
"inc2.v"`ifndef Xparameter A1 = 2;`endifparameter A2 = 4'b1111;parameter
A3 = "abcsde";parameter A5 = 3'b000;localparam L1 = 10;endmodule
```

Example of Parameterized Multi-Dimensional Ports/Wires

```
`define N_PORTS 3`define N_CHANNELS 4 module pwr_ctrl(           input
logic               clk, rst_n,                input   logic
[`N_PORTS-1 : 0] [2 : 0]  channel_addr_i,        input   logic
[`N_CHANNELS-1  : 0]          transaction_complete_i,        output
logic [`N_CHANNELS-1  : 0]       sleep,        output  logic [3:0][2:0]
run    );pwr_ctrl2 test0();endmodulemodule pwr_ctrl2(           input
logic               clk, rst_n,                input   logic [`N_PORTS-1
: 0] [2 : 0]            channel_addr_i_2,           input   logic
[`N_CHANNELS-1  : 0][`N_PORTS-1 :           0]
transaction_complete_i_2,         output  logic [`N_CHANNELS-1  : 0]
sleep,           output  logic [3:0][2:0] run    );endmodule
```

Example of Complex User-Defined Ports

```
typedef struct {  logic [7:0] x;  logic [7:0] y;} s_point; // named
structuremodule m1 (output s_point outPoint);//endmodulemodule m2 (input
s_point inPoint);//endmodulemodule top ();    s_point point;    m1
m1_inst (point);    m2 m2_inst (point);endmodule
```

Example on the Presence of Generate Blocks

```
module top (clk, out);  input         clk;  reg     [3:0] count = 0;
wire    [8:0] path;  output        out;  always @ (posedge clk)    count
<= count + 1;  assign path[0] = count[3];  assign out = path[8];  genvar
n;  generate    for (n=0; n<=7; n=n+1) begin : flops      dflop u
(.clk(clk), .in(path[n]), .out(path[n+1]));    end
endgenerateendmodulemodule dflop (clk, in, out);  input         clk, in;
output reg    out;  always @ (posedge clk)    out <= in;endmodule
```

Example on the Presence of Array-of-Instances

```
module my_module (a, b, c);input a, b;output c;  assign c = a & b
;endmodulemodule top (a, b, c) ;input [3:0] a, b;output [3:0] c;
my_module inst [3:0] (a, b, c);endmodule
```

Example of Partial Assignments in Glue/Generate Logic

```
module test(output p);  reg [4:0] p;  always@(*)  begin    p[2:0] =
2'b100;  endendmodule
```

Example Of VHDL Modules of Complex Types

```
// top.v
module top();  InstExecMon InstExecMon();endmodule

//test.vhd
library ieee;use ieee.std_logic_1164.all;use ieee.numeric_std.all;use
ieee.std_logic_textio.all;library work;use work.pkg.all;entity
InstExecMon is    generic(        gc_PrintAddr : boolean := true
);    port(        nRst            : in std_logic;        Clk
: in std_logic;        DnovaCoreNr      : in std_ulogic_vector(7 downto
0);        Core2InstExecMon : in t_Core2InstExecMon        );end
InstExecMon;
```

Example of Local ‘define Constructs

```
`ifdef x`define y 10`else`define Y 20`endifmodule test();parameter P1 =
`ifdef Z 10 `else 20 `endif ;`ifndef Wparameter P2 =10;`endif`ifdef
MM`ifdef NNparameter R = 10;`endif`endif`define HH 10`define GG
20endmodule
```

Example of Gate Primitives/Supply Net Types/Ports with Implicit Net Declarations

```
module gates();  wire out0;  wire out1;  wire out2;  reg
in1,in2,in3,in4;  not U1(out0,in1);  and U2(out1,in1,in2,in3,in4);  xor
U3(out2,in1,in2,in3);endmodule
```

Example of Escaped Identifiers

```
// There must be white space after the// string which uses escape
charactermodule \1dff (q,      // Q output\q~ ,   // Q_out outputd,
// D inputcl$k,   // CLOCK input\reset* // Reset input);input d, cl$k,
\reset* ;output q, \q~ ;  endmodule
```

Example of Pragmas in the Same Line as RTL Constructs

```
/*synopsys synthesis_off*/ `define  x 10 /*synopsys synthesis_on*/
module test(
/*pragma translate_off*/ input p1 /*pragma translate_on*/
);
//synopsys translate_off
parameter P = 10; //synopsys translate_on
endmodule
```

Example of defparam/Usage of real Keyword in AMS

```
module my_module (Clk, D, Q) ;  parameter width = 2,  delay = 10 ;  input
[width - 1 : 0] D ;  input Clk ;  output [width : 0] Q ;  assign #delay Q
= D;endmodulemodule top;  reg Clk ;  reg [7:0] D ;  wire [7:0] Q
;  my_module inst_1(Clk, D, Q) ;endmodulemodule override ;  defparam
top.inst_1.width = 7 ;endmodule
```

Example of Multiple Definitions of the Same Macro

```
`define NN 1`define Y 20module test();`define RR 3`define NN 2`define X
10`define RR 4endmodule`define Z 30
```

Example of Local `undef Constructs

```
`undef Xmodule A();  `undef Yendmodule
```

Examples of `ifdef on Partial Expressions/Statements

Example 1

```
genvar i;`ifdef FPGA   for (i = 0; i < SIZE1; i = i +1)`else  for (i = 0;
i < SIZE2; i = i + 1)`endifbegin : sreggen   sreg # (.P(x)) master
(.clk(clk), .data (data))endendgenerate
```

Example 2

```
`ifdef BEHAVIOR   always @(posedge clk1)`else  always @(posedge clk2)
`endifbeginout <= 8'b0;end
```

Example 3

```
module test(I1,I2,I3,O1);input I1;input I2;input I3;output O1;assign O1 =
I1 & `ifdef X                  I2               `else
I3                `endif                   ;endmodule
```

Example of RTL Code for Unsynthesizable Constructs in Same Line as Macros

Example 1

```
`ifdef XX initial begina = 1;end `endif
```

Example 2

```
always @(posedge clock)begin   `ifdef ALTERA_XCVR_ASSERTIONS assert
property   ($rose(reset_all) |=> $rose(pll_powerdown));`endif    end
```

Presence of Multiple `ifdef Branches in the Same Line

Example 1

```
`ifdef Y`define Z `ifdef R 10 `else 20 `endif`endifmodule
test();parameter P = `ifdef X 10 `else 20 `endif ;endmodule
```

Example 2

The following is the example of multiple `‘ifdef` branches in the same line when the `-preserve_configurations` argument of `set_rtlimport_option` is set to `TRUE`.

```
assign a = `ifdef XX b&c `else d&e `endif;wire w1, `ifdef XX w2 `endif,
`ifdef YY w3 `endif;
```

Example of `ifdef on Complete Module

```
`ifdef Xmodule A ();`elsemodule A(input a1);`endif endmodule`ifdef
Ymodule B();endmodule`endifmodule top();`ifdef XA A0();`elseA
A0(1'b1);`endif`ifdef YB B0();`endifendmodule
```

Example of ifdef on `define, `undef, `timescale

```
module test();`ifdef N`timescale 10ns/1ns`else`timescale 100s/
1ns`endif`ifdef X`ifdef Y`define Z 10`endif`endif`ifdef XX`undef
Z`endifendmodule
```

Example of `ifdef on Assertions, Initial Blocks, and Unsynthesizable Constructs

```
`ifdef simtypeBehavEmureg [1023:0] mem_file_0;   initial begin
mem_file_0 = "lib/new_ram/ram_code/B13RF16X74RR1_0.txt";    #50;
$display("%0t: ** Info: Pre-Loading the RAM    B13RF16X74RR1_0.txt %0s",
$time, mem_file_0);     //$readmemb(mem_file_0, memory_0.mem);
End`endif
```