

Verification Continuum™

Messages

Message Reference

November 2020

SYNOPSYS®

Copyright Notice and Proprietary Information

© 2020 Synopsys, Inc. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

Free and Open-Source Licensing Notices

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

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.

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 East Middlefield Road
Mountain View, CA 94043
www.synopsys.com

November 2020

Introduction to Messages

Messages for the Synopsys synthesis tools range from error messages that stop the intended operation to informative notes that echo system status. The messages are categorized according to type:

Message Type	Prefix	Description
Error	@E	A message that indicates a serious problem which must be corrected before proceeding with the design flow.
Warning	@W	The standard warning message. Check the message and make sure you understand the consequences of ignoring the message.
Note	@N	A standard notification message reporting expected system status. Notes are considered informative and require no user response.
Advisory	@A	An explanatory message that suggests possible design changes that can potentially improve the outcome of the intended operation. Always read and evaluate the content of an advisory message.

For detailed information about working with tool messages, consult the *User Guide* for your product.

CHAPTER 3

BM Messages 100 – 107

BM100

@W: Clock <clk> too slow at <00.020> MHz, resetting it to 1MHz(1000ns)

A clock frequency of less than 1 MHz was applied to a clock signal. The synthesis tool automatically resets any frequency less than 1 MHz to 1 MHz.

The following test case along with the constraint file causes the above warning because clock signal clk is assigned a clock frequency of 0.020 MHz.

Constraint File

```
# Synopsys, Inc. constraint file
# Clocks
define_clock -name {clk} -freq 0.020 -clockgroup default_clkgroup
```

Example.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity dff1 is
    port  (d, clk, rst: in std_logic;
           q: out std_logic );
end dff1;
```

```

architecture rtl of dff1 is
begin
    process (clk, rst)
    begin
        if rst = '1' then
            q <= '0';
        elsif (clk'event and clk ='1') then
            q <= d;
        end if;
    end process;
end;

```

Action

Assign clock signal frequencies of 1 MHz or greater. Change the clock frequency in the constraint file to at least 1 MHz.

Constraint File

```

# Synopsys, Inc. constraint file
# Clocks
define_clock -name {clk} -freq 1.0 -clockgroup default_clkgroup

```

Example.vhd

```

library ieee;
use ieee.std_logic_1164.all;

entity dff1 is
    port  (d, clk, rst: in std_logic;
           q: out std_logic );
end dff1;

architecture rtl of dff1 is
begin
    process (clk, rst)
    begin
        if rst = '1' then
            q <= '0';
        elsif (clk'event and clk ='1') then
            q <= d;
        end if;
    end process;
end;

```

BM101

@W: Clock <clkmy> too slow at <50000> ns, resetting it to 1000ns(1MHz)

A clock signal was assigned a clock period of more than 1000 ns. The synthesis tool resets the clock signal period to 1000 ns and generates the warning.

In the following test case, clock signal clk is assigned a clock period of 5000 ns. The synthesis tool resets the period to 1000 ns and generates the above warning. At a period of 5000 ns, the clock frequency is 0.2 MHz, which is very slow. Therefore, the synthesis tool resets the clock period to 1000 ns or 1 MHz as the minimum frequency for the design.

Constraint File

```
# Synopsys, Inc. constraint file
# Clocks

define_clock -name {clk} -period 5000.000
    -clockgroup default_clkgroup
```

Example.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity dff1 is
    port  (d, clk, rst: in std_logic;
           q: out std_logic );
end dff1;

architecture rtl of dff1 is
begin
    process (clk, rst)
    begin
        if rst = '1' then
            q <= '0';
        elsif (clk'event and clk ='1') then
            q <= d;
        end if;
    end process;
end;
```

Action

Use a clock period of 1000 ns or less (or a clock frequency of 1 MHz or greater). Change the clock period in the constraint file to 1000 ns or less.

Constraint File

```
# Synopsys, Inc. constraint file
# Clocks

define_clock -name {clk} -period 1000.000
    -clockgroup default_clkgroup
```

Example.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity dff1 is
    port  (d, clk, rst: in std_logic;
           q: out std_logic );
end dff1;

architecture rtl of dff1 is
begin
    process (clk, rst)
    begin
        if rst = '1' then
            q <= '0';
        elsif (clk'event and clk ='1') then
            q <= d;
        end if;
    end process;
end;
```

BM102

@W: Invalid frequency of <-20.000> specified for clock <clkmy> resetting it to 1MHz(1000ns)

If you applied a negative clock frequency to a clock signal, the synthesis tool resets the frequency to its default of 1 MHz. In the test case below, a negative clock period of -50 ns is applied to clock signal clk through the following constraint file.

Constraint File

```
# Synopsys, Inc. constraint file
# Clocks
create_clock -name {clk} -period {-50}
```

Example.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity dff1 is
    port (d, clk, rst: in std_logic;
          q: out std_logic
        );
end dff1;

architecture rtl of dff1 is
begin
    process (clk, rst)
    begin
        if rst = '1' then
            q <= '0';
        elsif (clk'event and clk ='1') then
            q <= d;
        end if;
    end process;
end;
```

Action

Apply a positive clock frequency to the clock signals in your design. Change the clock frequency in the constraint file to a positive value greater than 1 MHz.

Constraint File

```
# Synopsys, Inc. constraint file
# Clocks
define_clock -name {clk} -freq 1.000
    -clockgroup default_clkgroup
```

Example.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity dff1 is
    port (d, clk, rst: in std_logic;
          q: out std_logic
        );
end dff1;

architecture rtl of dff1 is
begin
    process (clk, rst)
    begin
        if rst = '1' then
            q <= '0';
        elsif (clk'event and clk ='1') then
            q <= d;
        end if;
    end process;
end;
```

BM103

@W: Compile point <v:work.Module_1> has Invalid type <true>; changed to locked

When you use a Tcl script as a constraint file for your design and specify the compile point value as Boolean (true/false), or logical (1/0) instead of locked, the synthesis tool generates the above warning.

Action

Use locked as the key word in the value field of your compile points.

BM104

@W: Wildcards not supported for Compile Points; ignoring define_compile_point *.*. Use a complete module name.

A wildcard name was used instead of the explicit module name for a compile point name.

You can specify a module as a compile point through the SCOPE editor Compile Points tab. Normally, to select and lock a module, you select a compile point using the pull-down menu on the Compile Points tab in the SCOPE constraint editor. However, if your design has several modules with similar names, Module_1, Module_2, Module_3..., and you decide to use the wildcard Module_* or *.* to specify that all the modules starting with the name Module are compile points, the mapper rejects this compile point assignment and gives the above warning.

Action

Use the full module name and apply the compile point constraint on each module individually by selecting each of them from the pull-down menu of the Compile Points tab in the SCOPE constraint editor. You can also drag and drop the view from the HDL analyst.

BM105

@W: Compile points not supported for this technology, ignoring define_compile_point command

A compile point was defined and the target technology does not support multipoint synthesis. In the following test case, module reg4 is defined as a compile point in the constraints file. Running the test case with an unsupported technology causes the compile-point command to be ignored and results in the above warning.

```

module inc(a_in, a_out);
  input [3:0] a_in;
  output [3:0] a_out;
  assign a_out = a_in + 1;
endmodule

module reg4(clk, rst, d, q); //Specify this module as compile point
  input [3:0] d;input clk, rst;
  output [3:0] q;reg [3:0] q;
  always @(posedge clk or posedge rst)
    if(rst)
      q <= 0;
    else
      q <= d;
endmodule

module test (clk, rst, q);
  input clk, rst;output [3:0] q;
  wire [3:0] a_in;inc i1(q, a_in);
  reg4 r1(clk, rst, a_in, q);
endmodule

```

For the above test case, module `reg4` is specified as a top-level compile point by including the following entry in the constraint file.

```

define_compile_point -comment {Automatically Inserted}
{v:work.reg4}
-type {locked} -cpfile {}

```

Action

Compile points can be specified only for devices that support multi-point synthesis. Either use a technology that supports the multipoint synthesis flow or remove the compile-point definition.

BM106

@W: Invalid period of <-10> specified for clock <clock> resetting it to 1000ns(1Mhz).

An invalid (negative) period was specified for a clock. The mapper automatically resets the period to 1000ns (1MHz).

Example:

```
define_clock -name {clock} -period -10 -clockgroup  
default_clkgroup_0
```

Action

Negative values for a clock period are not supported. Change the period to a positive value.

BM107

@W: define_multicycle_path: Number of cycles given must be greater than 0, constraint will be ignored

An illegal value has been manually entered in the constraint file for a `define_multicycle_path` constraint (only integer values greater than 0 are allowed). The SCOPE GUI does not allow integer values less than 0 to be specified for the `define_multicycle_path` constraint.

Example:

```
define_multicycle_path -comment {1} -from {i:DECODE.ALUOP[3:0]} -1
```

Action

Change the value entered for the `define_multicycle_path` constraint to a positive integer.

CHAPTER 4

BN Messages 100 – 667

BN100

@W: Cannot find object <test> to mark as compile point

The object specified as a compile point in the constraints file was non-existent. For the following test case, the following entry to a non-existent module (test) was included in the constraints file.

```
define_compile_point -comment {Automatically Inserted} {test}
                     -type {locked} -cpfile {}
```

Running the test case below results in the above warning.

```
module inc(a_in, a_out);
  input [3:0] a_in;
  output [3:0] a_out;
  assign a_out = a_in + 1;
endmodule

module reg4(clk, rst, d, q);
  input [3:0] d;
  input clk, rst;
  output [3:0] q;
  reg [3:0] q;
```

```
always @(posedge clk or posedge rst)
if(rst)
    q <= 0;
else
    q <= d;
endmodule

module test_syn_hier(clk, rst, q);
input clk, rst;
output [3:0] q;
wire [3:0] a_in;
inc il(q, a_in);
reg4 r1(clk, rst, a_in, q);
endmodule
```

Action

Compile points must reference valid modules. For the following test case, changing the module name to inc as shown below in the compile-point definition to correspond to the test-case module eliminates the warning.

```
define_compile_point -comment {Automatically Inserted}
{v:work.inc}
-type {locked} -cpfile {}
```

BN101

@W: Cannot find object <temp> to apply define_clock

A define_clock attribute in a compile-point, module-level constraint file referenced a non-existent clock signal in the design.

Action

Check the spelling of the clock name to which the attribute is applied.

BN102

@W: Cannot find object <temp> to apply define_input_delay

A define_input_delay attribute in a compile-point, module-level constraint file referenced a non-existent input signal in the design.

Action

Check the spelling of the input signal name to which the attribute is applied.

BN103

@W: Cannot find object <temp> to apply define_output_delay

A define_output_delay attribute in a compile-point, module-level constraint file referenced a non-existent output signal in the design.

Action

Check the spelling of the output signal name to which the attribute is applied.

BN106

@W: Cannot apply constraint <syn_hier> to <v:work.top>

The mapper was unable to apply a particular constraint. Failure to apply a constraint can occur for several reasons including applying an attribute to a non-existent object or applying an invalid attribute for the indicated technology. In the test case below, the mapper cannot find a module named top in the design as indicated by the warning.

Constraint File Content

```
# Attributes
define_attribute {v:work.top} syn_hier {firm}
```

Test Case

```
module inc(a_in, a_out);
  input [3:0] a_in;
  output [3:0] a_out;
  assign a_out = a_in + 1;
endmodule

module reg4(clk, rst, d, q);
  input [3:0] d;
  input clk, rst;
  output [3:0] q;
  reg [3:0] q;

  always @(posedge clk or posedge rst)
    if(rst)
      q <= 0;
    else
      q <= d;
endmodule

module test(clk, rst, q);
  input clk, rst;
  output [3:0] q;
  wire [3:0] a_in;
  inc i1(q, a_in);
  reg4 r1(clk, rst, a_in, q);
endmodule
```

Action

Be sure to apply proper constraints. For the above test case, changing the constraint in the constraint file as shown below eliminates the warning.

```
# Attributes
define_attribute {v:work.inc} syn_hier {firm}
```

BN108

**@W: syn_hier attribute not currently supported on instances:
"<Dmux>". Apply syn_hier to module using name
"<v:Data_Mux>".**

A syn_hier attribute was applied to an instance through a constraints file instead of on the parent module. The syn_hier attribute is applied only to modules and not to instances of a module. Once specified on a module, the attribute is applied to all instances of that module.

Action

Change the attribute so that it applies only to the module. The warning message identifies the name of the parent module.

BN109

@W: Wildcards not allowed in object names -- Find command ignored.

A wildcard was used for an object name in the constraint file. In the constraint file attribute entry below, the asterisk (*) wildcard in the find command results in the above warning.

Constraint File Content

```
# Attributes
define_attribute {find -reg -enable *} syn_useioff {1}
```

Test Case

```
module test_tcl (d1,d2,clk,en,q1,q2);
  input d1,d2,en,clk;
  output reg q1 ;
  output reg q2;
```

```
always@(posedge clk)
if(en)
    q1<=d1;

always@(posedge clk)
    q2<=d2;
endmodule
```

Action

Avoid using wildcards in object names in the constraint file.

```
# Attributes
define_attribute {find -reg -enable en} syn_useioff {1}
```

BN110

@W: Incorrect syntax for find - <find -reg -clock clk>

An invalid find command was used in an attribute entry in the constraint file. For the following test case, the find command syntax for the syn_useioff attribute is invalid which results in the warning.

Constraint File Content

```
# Clocks
create_clock -name {clk} -period {10}
create_clock -name {clk1} -period {10}

# Attributes
define_attribute {find -reg -clock clk} syn_useioff {0}
```

Test Case

```
module test_tcl (d1,d2,clk,clk1,en,q1,q2);
input d1,d2,en,clk,clk1;
output reg q1 ;
output reg q2;
```

```
always@(posedge clk)
if(en)
    q1<=d1;

always@(posedge clk1)
    q2<=d2;
endmodule
```

Action

Be sure to use the correct find command syntax in all attributes. In the valid syntax below, find accepts the switch -enable and finds all registers with enable en.

```
# Clocks
create_clock -name {clk} -period {10}
create_clock -name {clk1} -period {10}

# Attributes
define_attribute {find -reg -enable en} syn_useioff {0}
```

The find command has increased scope when used in collections. For example, to apply a syn_useioff =0 property on all flip-flops clocked by clk, the constraint file would be modified as follows:

```
# Collections
define_scope_collection source {find -hier -seq *
    -filter ( @view==dff* && @clock==clk) }

# Clocks
create_clock {clk} -name {clk} -period {10}
create_clock {clk1} -name {clk1} -period {6.667}

# Attributes
define_attribute {$source} syn_useioff {0}
```

BN114

**@W: Removing instance "<test>" of black box
"<view:work.test>(verilog)" because it does not drive other instances.**

The named instance of a black box is being removed because the instance is not driving any of the primary design outputs.

Action

To preserve the black box, apply a syn_noprune attribute to the instance and rerun synthesis.

BN116

**@W: Removing sequential instance <q[1]> of <view:
UNILIB.FDCPE(PRIM)> because there are no references to its outputs**

During sequential optimization, the mapper removed a sequential instance because it had an unused output. In the test case below, output q[1] is always 0 which causes the mapper to remove sequential instance q[1] as indicated by the above warning.

```
module square(a,clk,q);
  input [1:0] a;
  input clk;
  output reg [3:0] q;
  always@( posedge clk)
    q = a * a; //q is square of a
endmodule
```

Action

Because this warning is generated as a part of the sequential optimization process, you can ignore the warning if the corresponding instance is not necessary or you can prevent the optimization by enabling the Disable Sequential Optimizations option on the Device tab.

BN119

@W: Global frequency too slow at <0.120>MHz, resetting it to 1MHz(1000ns)

The global frequency specified was less than 1MHz. Values that are less than 1MHz are considered invalid for timing analysis purposes and are automatically reset to 1MHz (1000ns). The above warning is generated if the project file includes a -frequency value of less than 1MHz (1.000).

Action

Fractional values for a clock frequency are not supported. Change the frequency to a value greater than 1.0 (1MHz).

BN120

@W: Unrecognized resource in syn_allowed_resources: </lut>

An illegal value was specified for a syn_allowed_resources attribute that was being applied to a compile point. The legal values are blockrams, M4Ks, M512s, M-RAM, dsp_blocks, and dyps depending on the vendor technology selected. For the test case, include the following constraint file entries:

Under attributes:

```
define_attribute {v:work.reg4} syn_allowed_resources {lut=15}
```

Under compile points:

```
define_compile_point -comment {Automatically Inserted}
{v:work.reg4} -type {locked} -cpfile {}
```

The test case below generates the above warning:

```
module inc(a_in, a_out);
  input [3:0] a_in;
  output [3:0] a_out;
  assign a_out = a_in + 1;
endmodule

module reg4(clk, rst, d, q); // reg4 is specified as compile point
  input [3:0] d;
  input clk, rst;
  output [3:0] q;
  reg [3:0] q;

  always @(posedge clk or posedge rst)
    if(rst)
      q <= 0;
    else
      q <= d;
endmodule

module top_counter(clk, rst, q);
  input clk, rst;
  output [3:0] q;
  wire [3:0] a_in;
  inc i1(q, a_in);
  reg4 r1(clk, rst, a_in, q);
endmodule
```

The mapper generates this warning for the first run. In the second run, the mapper does not find any change in the compile point data so it simply uses the previous data and does not issue the warning.

Action

Make sure that the values for the attribute are legal. Only blockrams, M4Ks, M512s, M-RAM, dsp_blocks, and dyps can be specified as values for the syn_allowed_resources attribute (see the online help for detailed use of this attribute).

BN124

@W: Disconnecting duplicate driver <driverName> on net <netName>.
Duplication of drivers is frequently used in ASIC designs to increase drive strength. This form of duplication is not permitted in FPGA designs.

A duplicate driver is being removed.

Action

Duplication of drivers is frequently used in ASIC designs where inverters are added in parallel to increase drive strength. This form of duplication is not permitted in FPGA-based designs and the additional driver is removed as reported in the message.

BN126

@E: Net %N is missing a driver\n

Optimizations on the path can cause this error. For example, the tool optimizes redundant logic and nets without drivers.

Action

Follow this procedure:

1. Check the net connections.
 - Find the net or instance in the RTL view and expand the logic connected to it.
 - Crossprobe the logic in the Technology view. If any of the inputs are tied to constants or if the logic is redundant, tool optimizations could result in driver-less nets.
 - Check the connections in the RTL and ensure that the constants are applied correctly.

2. To prevent optimizations and preserve the logic through mapping, use these attributes:
 - `/* synthesis syn_preserve =1 */` on sequential instances.
 - `/* synthesis syn_hier = "fixed" */` on the module of the instance in question.
3. Re-run synthesis.

BN129

@E: Net "<q[0]>" has mixed driver types

The mapper found a net with more than one driver and at least one of these drivers was not a tristate. In the following test case, top-level output port q has two drivers, one from the rotate_1 instance and one from the dff_1 instance. The driver for q from module rotate is tristated. However, the driver from module dff is not tristated which results in incorrect circuitry that causes the mapper to error out.

```
module mux (out, a, b, sel); // mux
  output [7:0] out;
  input [7:0] a, b;
  input sel;
  assign out = sel ? a : b;
endmodule

module reg8(q, data, clk, rst); // 8-bit register
  output [7:0] q;
  input [7:0] data;
  input clk, rst;
  reg [7:0] q;
  always @(posedge clk or posedge rst)
  begin
    if (rst)
      q <= 0;
    else
      q <= data;
  end
endmodule
```

```

module rotate (q1,data,clk,r_l,rst,sell); // rotates bits or loads
output [7:0] q1;
input [7:0] data;
input clk, r_l, rst,sell;
reg [7:0] q;
wire [7:0] q1;
// when r_l is high, it rotates; if low, it loads data

always @(posedge clk or posedge rst)
begin
if (rst)
  q <= 8'b0;
else if (r_l)
  q <= {q[6:0], q[7]};
else
  q <= data;
end

assign q1 = sell ? q : 8'bzz;
endmodule

module dff (q, data, clk);
output [7:0]q;
input [7:0]data;
input clk;
reg [7:0]q;

always @(posedge clk)
begin
  q <= data;
end

endmodule

//-----
// Top Design
// -----

module top2 (q, a, b, sel, r_l, clk, rst, sell);
output [7:0] q;
input [7:0] a, b;
input sel, r_l, clk, rst,sell;
wire [7:0] mux_out, reg_out;
mux mux_1 (.out(mux_out), .a(a), .b(b), .sel(sel));
reg8 reg8_1 (.clk(clk), .data(mux_out), .q(reg_out), .rst(rst));
rotate rotate_1 (q, reg_out, clk, r_l, rst,sell);
dff dff_1 (q,mux_out,clk);
endmodule

```

Action

This error is a user error that can be corrected by editing the code.

q Driven by Only One Driver

If q is driven by one driver, edit the HDL code to reflect this. To eliminate the error in the above test case, comment out the instantiation `dffdff_1(q,mux_out,clk);` which is a non tristated driver for q.

```
module mux (out, a, b, sel); // mux
  output [7:0] out;
  input [7:0] a, b;
  input sel;
  assign out = sel ? a : b;
endmodule

module reg8(q, data, clk, rst); // 8-bit register
  output [7:0] q;
  input [7:0] data;
  always @(posedge clk or posedge rst)
  begin
    if (rst)
      q <= 0;
    else
      q <= data;
  end
endmodule

module rotate (q1,data,clk,r_l,rst,sell); // rotates bits or loads
  output [7:0] q1;
  input [7:0] data;
  input clk, r_l, rst,sell;
  reg [7:0] q;
  wire [7:0] q1;
  // when r_l is high, it rotates; if low, it loads data

  always @(posedge clk or posedge rst)
  begin
    if (rst)
      q <= 8'b0;
    else if (r_l)
      q <= {q[6:0], q[7]};
    else
      q <= data;
  end
```

```

assign q1 = sel1 ? q : 8'bzz;
endmodule

module dff (q, data, clk);
output [7:0]q;
input [7:0]data;
input clk;
reg [7:0]q;

always @(posedge clk)
begin
    q <= data;
end

endmodule

//-----
// Top Design
// -----
module top2 (q, a, b, sel, r_l, clk, rst, sel1);
output [7:0] q;
input [7:0] a, b;
input sel, r_l, clk, rst,sel1;
wire [7:0] mux_out, reg_out;
mux mux_1 (.out(mux_out), .a(a), .b(b), .sel(sel));
reg8 reg8_1 (.clk(clk), .data(mux_out), .q(reg_out), .rst(rst));
rotate rotate_1 (q, reg_out, clk, r_l, rst,sel1);
//dff dff_1 (q,mux_out,clk);
endmodule

```

q Has Multiple Drivers

If q has multiple drivers, all drivers must be tristated in the RTL code. To eliminate the error in the above test case, the module dff must have a tristate implementation for q as shown in the corrected test case below.

```

module mux (out, a, b, sel); // mux
output [7:0] out;
input [7:0] a, b;
input sel;
assign out = sel ? a : b;
endmodule

```

```
module reg8(q, data, clk, rst); // 8-bit register
output [7:0] q;
input [7:0] data;
always @(posedge clk or posedge rst)
begin
if (rst)
    q <= 0;
else
    q <= data;
end
endmodule

module rotate (q1,data,clk,r_l,rst,sel1); // rotates bits or loads
output [7:0] q1;
input [7:0] data;
input clk, r_l, rst,sel1;
reg [7:0] q;
wire [7:0] q1;
// when r_l is high, it rotates; if low, it loads data

always @(posedge clk or posedge rst)
begin
if (rst)
    q <= 8'b0;
else if (r_l)
    q <= {q[6:0], q[7]};
else
    q <= data;
end

assign q1 = sel1 ? q : 8'bz;
endmodule

module dff (q1, data, clk, sel);
output [7:0]q1;
input [7:0]data;
input clk,sel;
reg [7:0]q;

always @(posedge clk)
begin
    q <= data;
end
q1 = sel ? q : 8'bz;
endmodule
```

```

//-----
// Top Design
// -----
module top2 (q, a, b, sel, r_l, clk, rst, sell);
output [7:0] q;
input [7:0] a, b;
input sel, r_l, clk, rst, sell;
wire [7:0] mux_out, reg_out;
mux mux_1 (.out(mux_out), .a(a), .b(b), .sel(sel));
reg8 reg8_1 (.clk(clk), .data(mux_out), .q(reg_out), .rst(rst));
rotate rotate_1 (q, reg_out, clk, r_l, rst, sell);
dff dff_1 (q,mux_out,clk);
endmodule

```

q Drivers Driven from a Black Box

If q has drivers that are driven from a black box and the black box contains a tristate implementation of q, attach a `black_box_tri_pins` directive to port q to eliminate this error as shown in the code example below.

```

module rotate(q, data, clk, r_l, rst) /* synthesis syn_black_box
black_box_tri_pins ="q[7:0]" */;

```

BN130

@E: Net "<q[0]>" has multiple drivers

The mapper found a net with more than one driver and none of the drivers was a tristate. In the following test case, top-level output port q has two drivers, one from the `rotate_1` instance and one from the `dff_1` instance. The driver for q from the module `rotate` is tristated. However, the driver from the module `dff` is not tristated which results in incorrect circuitry that causes the mapper to error out.

```

module mux (out, a, b, sel); // mux
output [7:0] out;
input [7:0] a, b;
input sel;
assign out = sel ? a : b;
endmodule

```

```
module reg8 (q, data, clk, rst); // eight bit register
output [7:0] q;
input [7:0] data;
input clk, rst;
reg [7:0] q;

always @(posedge clk or posedge rst)
begin
if (rst)
    q <= 0;
else
    q <= data;
end

endmodule

module rotate (q1, data, clk, r_l, rst, sell) /* synthesis
    syn_black_box */; // rotates bits or loads
output [7:0] q1;
input [7:0] data;
input clk, r_l, rst,sell;
reg [7:0] q;
wire [7:0]q1;
// when r_l is high, it rotates; if low, it loads data

always @(posedge clk or posedge rst)
begin
if (rst)
    q <= 8'b0;
else if (r_l)
    q <= {q[6:0], q[7]};
else
    q <= data;
end

assign q1 = sell ? q : 8'bz;
endmodule

//-----
// Top Design
//-----


module top2 (q, a, b, sel, r_l, clk, rst);
output [7:0] q;
input [7:0] a, b;
input sel, r_l, clk, rst;
wire [7:0] mux_out, reg_out;
mux mux_1 (.out(mux_out), .a(a), .b(b), .sel(sel));
```

```
// notice that port connections listed by name can be in any order
reg8 reg8_1 (.clk(clk), .data(mux_out), .q(reg_out), .rst(rst));
// can mix port connections "in order" (below) with port
// connections "by name" (above)
rotate rotate_1 (q, reg_out, clk, r_l, rst,sel1);
rotate rotate_2 (q, reg_out, clk, r_l, rst,sel2);
endmodule
```

Action

This error is a user error that can be corrected by editing the code. How you correct the code varies based on your design considerations. Refer to the following situations.

q is Driven by One Driver

If q is driven by only one driver, edit the HDL code to reflect this. To eliminate the error in the above test case, comment out q's second non-tristated driver as shown in the corrected test case below.

```
module mux (out, a, b, sel); // mux
output [7:0] out;
input [7:0] a, b;
input sel;
assign out = sel ? a : b;
endmodule

module reg8 (q, data, clk, rst); // eight bit register
output [7:0] q;
input [7:0] data;
input clk, rst;
reg [7:0] q;

always @(posedge clk or posedge rst)
begin
if (rst)
    q <= 0;
else
    q = data;
end
endmodule
```

```

module rotate (q1, data, clk, r_l, rst, sel1) /* synthesis
    syn_black_box */; // rotates bits or loads
output [7:0] q1;
input [7:0] data;
input clk, r_l, rst,sel1;
reg [7:0] q;
wire [7:0]q1;
// when r_l is high, it rotates; if low, it loads data

always @(posedge clk or posedge rst)
begin
if (rst)
    q <= 8'b0;
else if (r_l)
    q <= {q[6:0], q[7]};
else
    q <= data;
end

assign q1 = sel1 ? q : 8'bz;
endmodule

//-----
// Top Design
//-----

module top2 (q, a, b, sel, r_l, clk, rst);
output [7:0] q;
input [7:0] a, b;
input sel, r_l, clk, rst;
wire [7:0] mux_out, reg_out;
mux mux_1 (.out(mux_out), .a(a), .b(b), .sel(sel));
// notice that port connections listed by name can be in any order
reg8 reg8_1 (.clk(clk), .data(mux_out), .q(reg_out), .rst(rst));
// can mix port connections "in order" (below) with port
// connections "by name" (above)
rotate rotate_1 (q, reg_out, clk, r_l, rst,sel1);
//rotate rotate_2 (q, reg_out, clk, r_l, rst,sel2);
endmodule

```

q Has Multiple Drivers

When q has multiple drivers, the drivers must be tristated in the HDL code. For example, in module rotate, the q1 port is tristated.

```

module rotate (q1, data, clk, r_l, rst, sell);
output [7:0] q1;
input [7:0] data;
input clk, r_l, rst, sell;
reg [7:0] q;
wire [7:0] q1;
// when r_l is high, it rotates; if low, it loads data

always @(posedge clk or posedge rst)
begin
if (rst)
    q <= 8'b0;
else if (r_l)
    q <= {q[6:0], q[7]};
else
    q <= data;
end

assign q1 = sell ? q : 8'bzz;
endmodule

```

q Has Multiple Drivers from a Black box

If q has multiple drivers that are driven from a black box and the black box contains a tristate implementation of q, attach a `black_box_tri_pins` directive to port q to eliminate this error as shown in the code example below.

```

module rotate (q, data, clk, r_l, rst) /* synthesis syn_black_box
black_box_tri_pins ="q[7:0]" */;

```

BN132

@W: Removing instance <`add_1.Q[7:0]`> because it is equivalent to instance <`inst.Q[7:0]`>. To keep the instance, apply constraint `syn_preserve=1` on the instance.

The mapper detected a sequential instance (flip-flop or latch) that had the same functionality as a sequential instance already present in the netlist. These instances can be merged for better resource utilization. When such optimizations are performed by the mapper, the above message appears in the log file to indicate which instance was removed and its equivalent instance in the mapped netlist. In the test case below, register Q in the adder

module and register Q in the prep5 module are equivalent. The mapper removes one instance (in adder) and maps the other instance (in prep5) in the top-level design; the outputs of the top module include a single set of registers driving both the Q1 and Q2 outputs.

```
module prep5(Q, CLK, MAC, RST, A, B,adder_output );
output [7:0] Q;
output [7:0] adder_output;
input CLK, MAC, RST;
input [3:0] A, B;
reg [7:0] Q;

// multiplier
wire [7:0] multiply_output = A * B;

// adder:
wire [7:0] adder_output = MAC ? multiply_output + Q :
multiply_output;

// register with asynchronous reset
always @(posedge CLK or posedge RST)
begin
    if (RST)
        Q = 0;
    else
        Q = adder_output;
end
endmodule

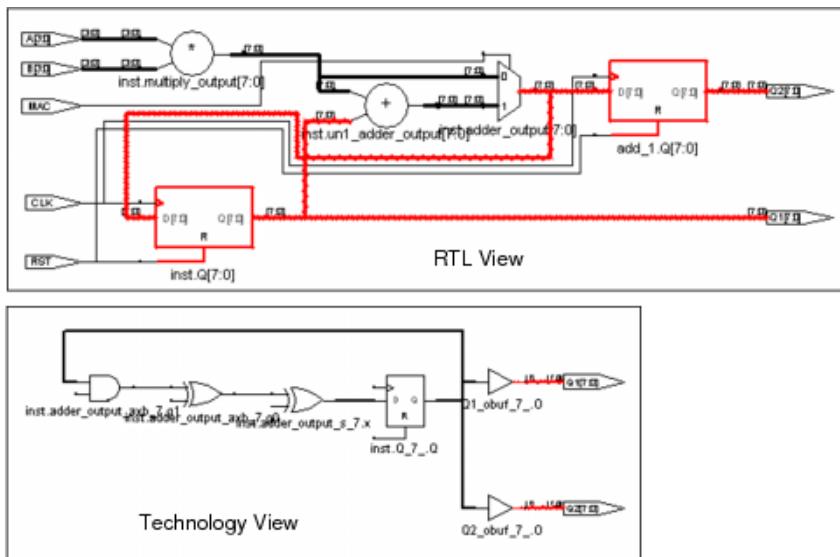
module adder ( CLK,RST, Q,adder_output );
input CLK,RST;
input [7:0] adder_output;
output [7:0] Q;
reg [7:0] Q;

always @(posedge CLK or posedge RST)
begin
    if (RST)
        Q = 0;
    else
        Q = adder_output;
end
endmodule
```

```
module top (CLK, MAC, RST, A, B, Q1, Q2 ) ;
  input CLK,MAC, RST;
  input [3:0] A, B;
  output [7:0] Q1,Q2;
  wire [7:0] adder_output;
  adder add_1 ( CLK,RST, Q2,adder_output ) ;
  prep5 inst (Q1, CLK, MAC, RST, A, B,adder_output );
endmodule
```

Schematic Views

Register Q in the adder module and register Q in the prep5 module are equivalent as shown in the RTL view below. The mapper removes one instance (in adder) and maps the other instance (in prep5) in the top-level design; the outputs of the top module include a single set of registers driving both the Q1 and Q2 outputs. The technology view below shows bit 7 of the Q1 and Q2 outputs driven by the merged register.



Action

Such optimizations are performed by both the compiler and the mapper. In the mapping stage, the hierarchy is dissolved to do such optimizations and merges. If the above message occurs, it is likely that the equivalent instances mentioned are in different module boundaries. If you need to disable this optimization, apply the `syn_preserve` directive on the module or globally.

BN133

@N: Ignoring syn_hier=hard property on top-level design.

A `syn_hier = "hard"` attribute was explicitly applied to the top module of the design (a `syn_hier = "hard"` attribute is implied at every top level; applying the attribute is redundant).

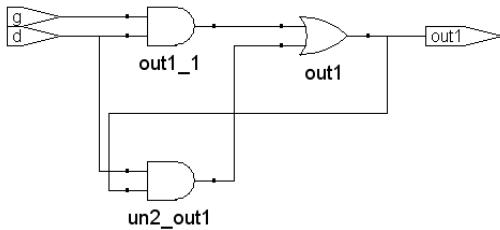
Action

Remove the explicit `syn_hier = "hard"` attribute from the top-level module.

BN134

@W: Found combinational loop during mapping.

A combinational loop was encountered during mapping. In the test case below, `out` is an output that uses itself as one of the inputs as shown in the schematic below and hence the mapper reports the above warning.



```

library ieee;
use ieee.std_logic_1164.all;

entity test is
port (g,d: in std_logic;
      out1: inout std_logic );
end entity;

architecture combo of test is
begin
    out1 <= (g and d) or
              (d and out1);
end combo;

```

Action

Make sure that there are no combinational loops in the design which can cause unexpected behavior and mismatches in RTL to post-synthesis simulation. To eliminate the warning in the above test case, register the signal out as shown:

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
port (g, d, clk: in std_logic;
      out1: inout std_logic );
end entity;

```

```
architecture seq of test is
begin
  process ( clk )
  begin
    if (rising_edge (clk )) then
      out1 <= (g and d) or ( d and out1);
    end if;
  end process;
end seq;
```

BN137

@W: Found combinational loop during mapping at net <*out_0*>

A combinational loop was encountered during mapping. In the following test case, out is an output that uses itself as one of the inputs.

```
module test (in,out);
  input  in;
  output out;
  assign out = in & !out;
endmodule
```

Action

Make sure that there are no combinational loops in the design; combinational loops cause unexpected behavior and mismatches in RTL to post-synthesis simulation. To eliminate the warning in the above test case, register the signal out to remove the combinational loop in the design.

BN139

@W: syn_hier=macro property not allowed on top-level design -- property removed.

A `syn_hier="macro"` attribute was applied to the top-level module of a design (some device targets allow a `syn_hier` attribute to have a “macro” value to indicate to the mapper that the module was a macro and that its interface was to be maintained). The mapper expected macro modules not to have any unmapped primitives. Macros should be technology-specific, post-mapped netlists. The attribute was automatically removed.

Action

Informative message; no action required.

BN169

@W: Cyclic dependency detected among libraries: *libraryName* *libraryName* ...

Your design has multiple modules that reference the same library at different levels of the hierarchy in a discontinuous way, causing a dependency loop between the libraries.

Consider a design that references the top level from the default work library, while an instantiated module test is referenced from the test library. The test module instantiates another lower-level module called add, which in turn is referenced from the default work library. This results in a cyclic dependency between the libraries: work -> test -> work.

The message identifies the libraries at issue and warns you of the cyclic dependency which you may not have intended.

Action

Check that you are calling the correct module from the correct library. To avoid this warning, rename the libraries for the different modules by right-clicking the module, selecting File Options, and changing the library name.

BN170

@W: Cyclic dependencies among libraries found.

Your design has multiple modules that reference the same library at different levels of the hierarchy in a discontinuous way, causing a dependency loop between the libraries.

Consider a design that references the top level from the default work library, while an instantiated module test is referenced from the test library. The test module instantiates another lower-level module called add, which in turn is referenced from the default work library. This results in a cyclic dependency between the libraries: work -> test -> work.

This message warns you of this cyclic dependency, which you might not have intended.

Action

Check that your design calls the intended module from the correct library. To avoid this warning, rename libraries for the different modules by right-clicking the module, selecting File Options, and changing the library name.

BN191

@N: Writing property annotation file <filename>.

The timing information from the back annotation file from the place and route tool was written to a text file in the implementation directory.

Action

Informative message; no action required.

BN196

@E: Duplicate driver <*driverName*> on net <*netName*>. This driver is a primary input. Duplication of drivers is frequently used in ASIC designs to increase drive strength. This form of duplication is not permitted in FPGA designs.

A wire with multiple drivers with unresolved priority cannot be resolved. Driver duplication is frequently used in ASIC design to increase drive strength, but is not permitted in FPGA designs.

In the test case below, input ports in1 and in3 drive a single wire which causes the compiler to error out.

```
library ieee;
use ieee.std_logic_1164.all;

entity tristate is
    port (in1,in2,in3,clk: in std_logic;
          regout:out std_logic);
end tristate;

architecture behave of tristate is
signal tmp1,regin: std_logic;
begin
tmp1 <= in1;
tmp1 <= in3;
regin <= tmp1 and in2;
process (clk)
begin
    if (clk = '1' and clk'event ) then
        regout <= regin;
    end if;
end process;
end behave;
```

Action

Make sure that a wire does not have multiple drivers with unresolved priority. In the corrected test case below, the priority is resolved by using mux and buffer logic.

```
library ieee;
use ieee.std_logic_1164.all;

entity tristate is
    port (en1,en2,in1,in2,in3,clk: in std_logic;
          regout:out std_logic);
end tristate;

architecture behave of tristate is
signal tmp1,regin: std_logic;
begin
    process (en1, in1)
    begin
        if (en1 = '0') then
            tmp1 <= 'Z';
        else
            tmp1 <= in1;
        end if;
    end process;

    process (en2, in3)
    begin
        if (en2 = '0') then
            tmp1 <= 'Z';
        else
            tmp1 <= in3;
        end if;
    end process;

    regin <= tmp1 and in2;
    process (clk)
    begin
        if (clk = '1' and clk'event ) then
            regout <= regin;
        end if;
    end process;
end behave;
```

BN234

@W: syn_probe attribute can only be applied to nets and requires object qualifier "n:" for object <alu_tmp[7:0]>

A syn_probe attribute was applied on a net without the n: qualifier. For the test case below, if the constraint file contains the entry

```
define_attribute {alu_tmp[7:0]} {syn_probe} { alu1_probe[] }
```

the mapper issues the above warning to indicate that the attribute is being applied on the net without an n: qualifier.

```
module alu(out1, opcode, clk, a, b, sel);
    output [7:0] out1;
    input [2:0] opcode;
    input [7:0] a, b;
    input clk, sel;
    reg [7:0] alu_tmp ;
    reg [7:0] out1;

    // Other code

    always @ (opcode or a or b or sel)
    begin
        case (opcode)
            3'b000:alu_tmp <= a+b;
            3'b001:alu_tmp <= a-b;
            3'b010:alu_tmp <= a^b;
            3'b011:alu_tmp <= sel ? a:b;
            default:alu_tmp <= a|b;
        endcase
    end

    always @(posedge clk)
        out1 <= alu_tmp;
endmodule
```

Action

Make sure that the syn_probe attribute is applied on a net with a proper n: qualifier as shown in the corrected constraint file entry below.

```
define_attribute {n:alu_tmp[7:0]} {syn_probe} { alu1_probe[] }
```

BN238

@W: Constraint object has wildcard(s) but is missing a qualifier (e.g., "p:", "i:", "t:", or "n:"): define_attribute <out1*> syn_useioff 0

The target object of an attribute included a wildcard, but no qualifier. For the test case below, if the constraint file includes the following entry, the mapper issues the above warning to indicate that without a qualifier, it cannot identify the proper object on which to apply the attribute.

```
define_attribute {out1*} {syn_useioff} {0}
```

Test Case

```
module test(out1, clk, a, b, sel);
output [1:0]out1;
input [1:0]a, b;
input clk;
input [1:0]sel;
reg [1:0]alu_tmp;
reg [1:0]out1;

always @(sel[0],a[0],b[0])
begin
    if(sel[0])
        alu_tmp[0]=a[0];
    else
        alu_tmp[0]=b[0];
end

always @(sel[1],a[1],b[1])
begin
    if(sel[1])
        alu_tmp[1]=a[1];
    else
        alu_tmp[1]=b[1];
end

always @(posedge clk)
    out1 <= alu_tmp;
endmodule
```

Action

Make sure that the constraint object has the proper qualifier. For the above case, adding the proper p: qualifier in the constraint file entry eliminates the warning message.

```
define_attribute {p:out1*} {syn_useioff} {0}
```

BN243

@E: Primitive logic found inside macro <inc>; syn_hier=macro not supported for macros with primitives

During mapping, a syn_hier=macro attribute was encountered on a macro with an internally-defined primitive. In the test case below, module inc has its functional definition explicitly defined which makes the syn_hier=macro attribute an invalid statement on the module and causes the compiler to error out.

```
module inc(a_in, a_out) /*synthesis syn_hier="macro"*/ ;
  input [3:0] a_in;
  output [3:0] a_out;
  assign a_out = a_in + 1;
endmodule

module reg4(clk, rst, d, q);
  input [3:0] d;
  input clk, rst;
  output [3:0] q;
  reg [3:0] q;

  always @(posedge clk or posedge rst)
    if(rst)
      q <= 0;
    else
      q <= d;
endmodule
```

```
module top(clk, rst, q);
  input clk, rst;
  output [3:0] q;
  wire [3:0] a_in;
  inc i1(q, a_in);
  reg4 r1(clk, rst, a_in, q);
endmodule
```

Action

Specify the `syn_hier=macro` attribute only on proper macros such as EDIF files. In the test case below, module `inc` is a wrapper over an EDIF with the same name which makes the `syn_hier=macro` attribute valid.

```
module inc(a_in, a_out) /* synthesis syn_hier = "macro" */;
  input [3:0] a_in;
  output [3:0] a_out;
endmodule

module reg4(clk, rst, d, q);
  input [3:0] d;
  input clk, rst;
  output [3:0] q;
  reg [3:0] q;

  always @(posedge clk or posedge rst)
    if(rst)
      q <= 0;
    else
      q <= d;
endmodule

module top(clk, rst, q);
  input clk, rst;
  output [3:0] q;
  wire [3:0] a_in;
  inc i1(q, a_in);
  reg4 r1(clk, rst, a_in, q);
endmodule
```

BN244

@E: Primitive logic found inside macros. syn_hier=macro is not supported for macros with primitives.

The mapper encountered a `syn_hier=macro` attribute on a macro with internally defined primitives. In the test case below, module `inc` has its functional definition explicitly defined which makes the `syn_hier=macro` attribute an invalid statement on the module and causes the compiler to error out.

```
module inc(a_in, a_out) /*synthesis syn_hier="macro" */ ;
  input [3:0] a_in;
  output [3:0] a_out;
  assign a_out = a_in + 1;
endmodule

module reg4(clk, rst, d, q);
  input [3:0] d;
  input clk, rst;
  output [3:0] q;
  reg [3:0] q;

  always @ (posedge clk or posedge rst)
    if(rst)
      q <= 0;
    else
      q <= d;
endmodule

module top(clk, rst, q);
  input clk, rst;
  output [3:0] q;
  wire [3:0] a_in;
  inc i1(q, a_in);
  reg4 r1(clk, rst, a_in, q);
endmodule
```

Action

Specify the `syn_hier=macro` attribute only on proper macros such as EDIF files. In the test case below, module `inc` is a wrapper over an EDIF with the same name which makes the `syn_hier=macro` attribute valid.

```
module inc(a_in, a_out) /* synthesis syn_hier = "macro" */;
  input [3:0] a_in;
  output [3:0] a_out;
endmodule

module reg4(clk, rst, d, q);
  input [3:0] d;
  input clk, rst;
  output [3:0] q;
  reg [3:0] q;
  always @(posedge clk or posedge rst)
    if(rst)
      q <= 0;
    else
      q <= d;
endmodule

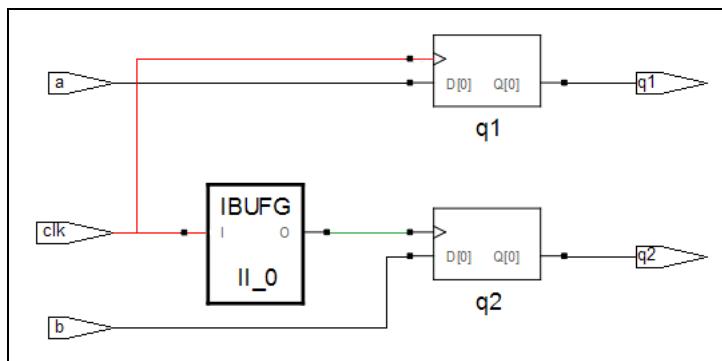
module top(clk, rst, q);
  input clk, rst;
  output [3:0] q;
  wire [3:0] a_in;
  inc i1(q, a_in);
  reg4 r1(clk, rst, a_in, q);
endmodule
```

BN245

@E: Port '<clk>' on chip '<test>' drives <1> pad loads and <1> non-pad loads

The mapper encountered an input driving both a pad load and a non-pad load (inputs cannot drive both a pad load and a non-pad load). This error is also reported if an input or output buffer signal is marked for instrumentation in the instrumentor.

```
module test (
    input clk,
    input a,b,
    output reg q1,q2 );
wire clk_b;
IBUFG (.O(clk_b),.I(clk));
always @ (posedge clk)
begin
    q1 <= a;
end
always @ (posedge clk_b)
begin
    q2 <= b;
end
endmodule
```



Action

Make sure that input loads are used consistently. To correct the above test case, change the unbuffered clock driving q1 (clk) to the buffered clock driving q2 (clk_b).

```
module test (
    input clk,
    input a,b,
    output reg q1,q2 );
wire clk_b;
IBUFG (.O(clk_b),.I(clk));

always @ (posedge clk_b)
begin
    q1 <= a;
end

always @ (posedge clk_b)
begin
    q2 <= b;
end

endmodule
```

BN257

@E: Compile point declared on <view> must be of type "locked"

A compile point with a value other than locked is set on an EDIF module (EDIF compile-point modules can only be of type locked).

Action

When using EDIF-based compile points, make sure that the compile-point type is set to locked.

BN291

**@A: Boundary register <registerName> is packed into a complex cell.
To disable register packing, set syn_keep=1 on the net between
the register and the complex cell.**

A boundary register, in this case, was defined as a register that separates clock groups. This definition includes:

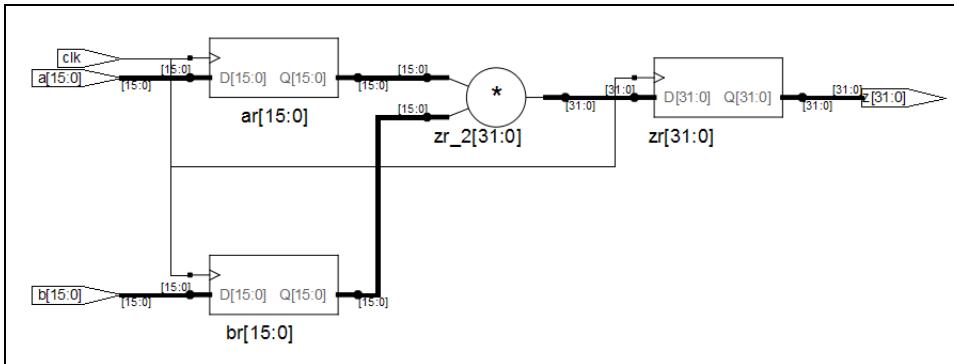
- Registers that have one clock group driving the input and a different clock group as the load. These registers are often used as synchronizers or in hand-shaking protocols.
- Registers that are separated from unconstrained I/O ports by only combinational logic. These registers effectively separate a non-clock group from an actual clock group. When these I/Os are left unconstrained, the on/off chip paths have usually been minimized by hand (i.e., the I/O port is directly connected to a register). In these cases, there is nothing to be optimized and the user may not invest the time defining the I/O delays.

From a synthesis perspective, paths between unrelated clocks or from/to unconstrained I/Os have no defined timing goals and are effectively false paths. However, because the situations discussed above are so common, optimizations that move registers, such as retiming, are restricted on boundary registers to avoid adding significant logic and delay to these special (synchronizer, hand-shake, or pre-optimized I/O) paths. There is one exception to this rule: a boundary register is allowed to be packed into a complex cell such as a RAM or DSP block. This advisory message is reported in these situations.

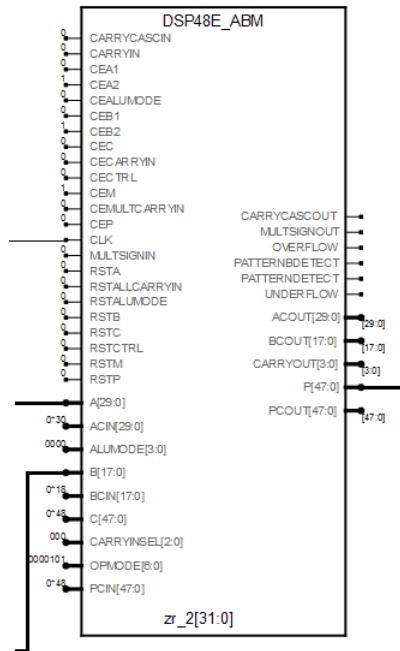
The following source code describes a simple 16-by-16 multiplier with its inputs and outputs registered.

```
module dont_pack (clk, a, b, z);
    input clk;
    input [15:0] a, b;
    output [31:0] z;
    reg [15:0] ar, br;
    reg [31:0] zr;
```

```
always @(posedge clk) begin
    ar <= a;
    br <= b;
    zr <= ar * br;
end
assign z = zr;
endmodule
```



If the I/Os are unconstrained, the Synopsys FPGA mapper packs the registers into a multiplier block if one is available as shown below.



Action

If the packing of a boundary register into a complex cell is adding too much delay to one of these special paths, packing can be disabled by placing a `syn_keep` attribute on the net between the register and the complex cell using constraints as illustrated below.

```

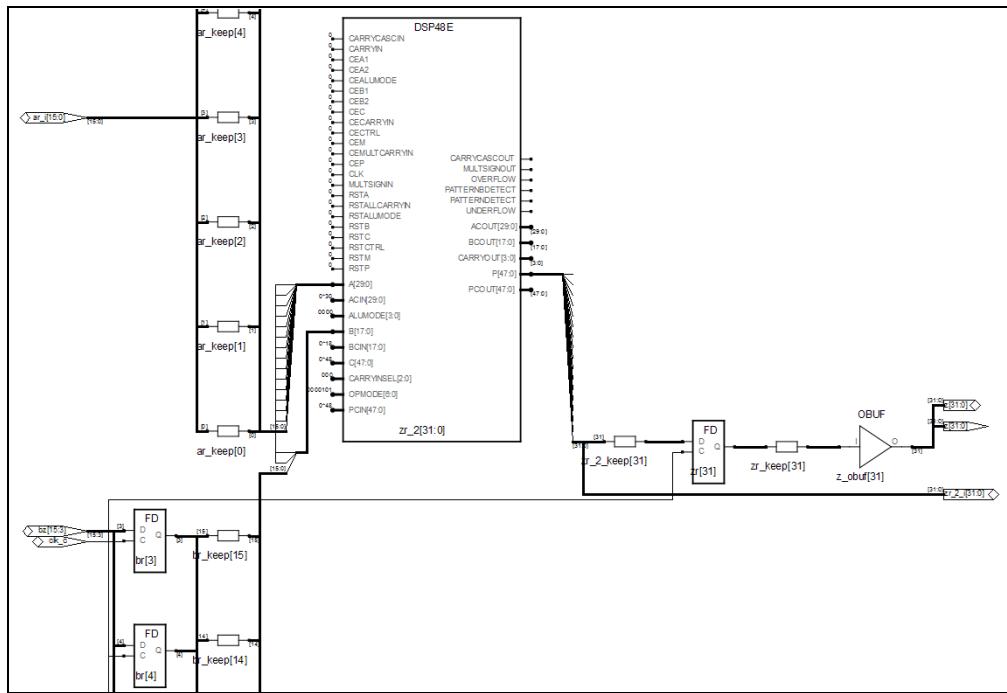
# Disable packing register "ar" into a multiplier cell
define_scope_collection dont_pack_rega
  {define_collection {i:ar[15:0]}}
define_scope_collection output_neta {expand -level 1 -net
  -from $dont_pack_rega}
define_scope_collection input_pina
  {find -pin *.*D\[* -in [expand -level 1 -pin -to
$dont_pack_rega]}
define_scope_collection input_neta [expand -level 1 -net
  -to $input_pina]
define_attribute $output_neta {syn_keep} {1}
define_attribute $input_neta {syn_keep} {1}

```

```
# Disable packing register "br" into a multiplier cell
define_scope_collection dont_pack_regb {define_collection
    {i:br[15:0]}}
define_scope_collection output_netb {expand -level 1 -net
    -from $dont_pack_regb}
define_scope_collection input_pinb
    {find -pin *.D\[* -in [expand -level 1 -pin -to
    $dont_pack_regb]}
define_scope_collection input_netb [expand -level 1 -net
    -to $input_pinb]
define_attribute $output_netb {syn_keep} {1}
define_attribute $input_netb {syn_keep} {1}

# Disable packing register "z" into a multiplier cell
define_scope_collection dont_pack_regz
    {define_collection {i:zr[31:0]}}
define_scope_collection output_netz {expand -level 1 -net
    -from $dont_pack_regz}
define_scope_collection input_pinz
    {find -pin *.D\[* -in [expand -level 1 -pin -to
    $dont_pack_regz]}
define_scope_collection input_netz [expand -level 1 -net
    -to $input_pinz]
define_attribute $output_netz {syn_keep} {1}
define_attribute $input_netz {syn_keep} {1}
```

After applying these constraints, the registers are no longer packed into the multiplier cell as shown in the figure below.



BN295

@E: Design uses DesignWare components, but no DesignWare license available

The Synplify® Premier synthesis tool is waiting for a DesignWare® license to process the DesignWare IP and times out when no license is available.

Action

An IP block without a requested license is processed as either an error which stops the process or as a black box which allows the process to continue (by default, the tool waits for a DesignWare license). To stop the process and error out immediately when no license is available, include the following set_option:

```
set_option -dw_stop_on_nolic 1
```

For information on Queuing Synopsys DesignWare IP licenses, see *Queuing Synopsys DesignWare IP Licenses* in the *User Guide*.

BN301

@E: Combinational logic depth greater than <integer>

The number of gates in a combinational loop exceeds a predefined value (*integer*). The depth of the loop is the number of gates within the loop. The current maximum depth is 10,000 gates.

Action

Large loops are generally an indication of design problem. If a loop in excess of 10,000 gates is acceptable, contact a Synopsys applications consultant to arrange raising the limit.

BN314

@E: Net <netName> has multiple drivers

An attempt was made to assign multiple drivers to the indicated net. The target net (*netName*) is reported in the message.

Action

A net cannot be assigned to more than one driver. If one of the drivers has a constant value (either true or false), enable the Resolve Mixed Drivers option to assign the net to the constant and then recompile the design.

BN321

@A: Found multiple drivers on net <netName>; if one driver is a constant (true or false), use Resolve Mixed Drivers option to connect the net to VCC or GND.

This advisory accompanies error message [BN314](#). It offers a solution to resolve multiple drivers assigned to the indicated net when one of the drivers is a constant. The target net (*netName*) is reported in the message.

BN354

@E: True (VCC) and false (GND) driving same net in netlist <netListName>, port <portName>. Multiple-driver error

An attempt was made to assign conflicting constants to the indicated net. The netlist (*netListName*) and the target port (*portName*) are reported in the message.

Action

A design error is indicated. The net cannot be assigned to two constants. Make sure that the net is assigned to only one constant.

BN363

@E: True (VCC) and false (GND) driving same net in netlist <netListName>. Multiple-driver error

An attempt was made to assign conflicting constants to the indicated net. The netlist (*netListName*) is reported in the message.

Action

A design error is indicated. The net cannot be assigned to two constants. Make sure that the net is assigned to only one constant.

BN364

**@E: True (VCC) and false (GND) driving same net in netlist
<netListName>; one load of the net is <instanceName>. Multiple-driver error**

An attempt was made to assign conflicting constants to the indicated net. The netList (*netListName*) and one of the target load instances (*instanceName*) are reported in the message.

Action

A design error is indicated. The net cannot be assigned to two constants. Make sure that the net is assigned to only one constant.

BN384

@W: Found more than 100 combinational loops; only 100 will be annotated

Instances in combinational loops are normally annotated with an identifying property. To prevent excessive run times in the HDL analyst, a limit is placed on both:

- the number of instances that can be assigned this property (see [BN385](#))
- the number of loops that can include annotated instances (if the above instance limit is not reached first)

The loop limit is set to 100.

Action

No direct user action. When the 100-loop or 1000-instance limit is reached, property annotation is automatically discontinued. All combinational loops are still detected even when either limit is reached.

BN385

@W: Found more than 1000 instances in combinational loops; only 1000 instances will be annotated

Instances in combinational loops are normally annotated with an identifying property. To prevent excessive run times in the HDL analyst, a limit is placed on both:

- the number of instances that can be assigned this property
- the number of loops that can include annotated instances (if the above instance limit is not reached first – see warning message [BN384](#)).

The instance limit is set to the first 1000 instances found in no more than 100 combinational loops.

Action

No direct user action. When the 1000-instance or 100-loop limit is reached, property annotation is automatically discontinued. All combinational loops are still detected even when either limit is reached.

BN393

@W: Resolving multiple drivers on net <netName>, connecting output <outputName> to <netName>.

This warning indicates that multiple drivers are being resolved for the specified net. The target net (*netName*) and target output (*outputName*) are reported in the message.

This warning occurs when all of the following are true:

- There are multiple drivers assigned to a net
- Only one of the drivers is a constant (either true or false)
- Resolve Mixed Drivers option is enabled in Device Mapping Options

BN396

@E: Cannot connect across fixed hierarchies

A hyper source or hyper connect cannot be connected across a fixed hierarchy. As new ports are not allowed to be created for a hyper source or hyper connect that is within a fixed hierarchy, connection to a hyper source or hyper connect external to the hierarchy is prohibited.

In the following example, view t1_fixed is marked syn_hier=fixed with a hyper connect inside the hierarchy and its hyper source outside of the hierarchy. Net i1 cannot be threaded to cell t1_xmr0 because of locked view view:work.t1_xmr0 which results in the error.



Action

To resolve this issue, remove the `syn_hier=fixed` property from the view.

BN402

@E: Errors encountered while making syn_hyper (or XMR/Identify) connections

Cross-module references (XMRs) are included within the generate statements, and the width mismatch between the source and destination of XMR connections have caused the compiler to fail.

Action

Write the XMR outside the generate statements.

If XMR is inside a for-generate statement, then unroll the XMR access.

BN502

@E: Unable to chain UMR instance <*instance*> because of syn_hier-fixed or compile points

The specified UMRBus® instance could not be chained (or unchained) because of the presence of a syn_hier=fixed attribute or a compile-point boundary.

Action

Remove the syn_hier=fixed attribute from the instance.

BN511

@E: Sequential logic depth greater than <*integer*>

The number of elements in a sequential loop exceeds a predefined value (*integer*). A sequential loop is made up of both sequential and combinational elements; the depth of the loop is the number of elements within the loop. The current maximum depth is 10,000 elements.

Action

Large loops are generally an indication of design problem. If a loop in excess of 10,000 elements is acceptable, contact a Synopsys applications consultant to arrange raising the limit.

BN519

@W: Possible stack overflow -- make sure that the run-time stack size is unlimited

The above warning occurs when a potential process memory overflow is possible (overflows can occur from recursive function calls or extremely large data sets).

Action

To circumvent a stack overflow, set the stack size limit to unlimited using one of the following commands:

- Linux C-shell: limit stacksize unlimited
- Linux Bash shell: ulimit -s unlimited
- Windows: \$ editbin /STACK:size

BN521

@E: EDIF input flow is not supported by the selected technology

This error occurs when an EDIF file is applied as input to the synthesis flow in the selected technology.

Action

Make sure to use only Verilog or VHDL input files in the synthesis flow.

BN528

@E: Clock period <*period*> is too short

The period of a `create_clock` command is near zero.

Action

Increase the period (lower the frequency).

BN529

@E: Clock frequency <*frequency*> is too low

The frequency of a `create_clock` command is near zero.

Action

Increase the frequency.

BN530

@E: Clock frequency <*frequency*> is too high

The frequency of a `create_clock` command is too high (the period is too short).

Action

Decrease the frequency.

BN533

@E: Redefining generated clock <*clockName*> as not generated

Clock names must be unique. If two clocks have the same name, the name of the clock created first will be dropped.

Action

Give one of the clocks a new, unique name.

BN534

@E: Redefining clock <*clockName*> as generated

Clock names must be unique. If two clocks have the same name, the name of the clock created first will be dropped.

Action

Give one of the clocks a new, unique name.

BN536

@E: Format mismatch of attribute <*attributeName*> on instance <*instanceName*>

An attribute definition for a user-instantiated component was not applied correctly. For example, this error can occur when an attribute parameter is defined as an integer, but was specified as a string.

Action

You can use the attribute format defined for unisim models.

BN539

@W: syn_hier=<value> on compile point <cpName> not supported. Removing syn_hier=<value>.

This warning occurs when you apply a syn_hier attribute/directive with a value other than flatten to a compile point (flatten is the only accepted attribute value for compile points).

BN542

@E: Unable to find internals for black-box view <blackBoxView>

The software is unable to find the internals of a blackboxed view.

Action

Open a support ticket by visiting the Synopsys website and clicking the Support link at the top of the page.

BN543

@E: Internals for black-box view <blackBoxView> not found in file <databaseFile>

The database file is incomplete. This can occur if the files were not copied to the database correctly.

Action

Verify that all files were copied to the database. If all files exist, then open a support ticket on the Synopsys website.

BN547

**@E: The SRS file loaded is not compatible with this software release.
Please regenerate the SRS file.**

The SRS file being used was generated by an older version of the software.

Action

Regenerate the SRS file with the current version of the software.

BN548

@E: Multi-file database version <fileVersion> is not supported by this tool

The multi-file database is newer than the software version used to load it. (A newer software version may generate a multi-file database that cannot be read by older versions of the software.)

Action

You must load the multi-file database with the same or later version of the software that was used to create it.

BN551

@E: Could not read the skeleton netlist

The skeleton netlist file mapped to the multi-file database cannot be read. This can happen if the netlist file:

- Cannot be found. The missing skeleton file may have been copied incorrectly or was deleted accidentally. (see [BN552](#))
- Has been corrupted. The skeleton file can become corrupted if it was partially copied or data corruption occurred while writing to the disk.

Action

If you copy the multi-file database, make sure that the skeleton file is not corrupt when it gets copied. Also, when the synthesis tool generates the multi-file database, the skeleton file might be missing because the software terminated while writing to this file. In this case, you must regenerate the multi-file database.

BN552

@E: Skeleton file <fileName> does not exist

The skeleton netlist file, which is mapped to the multi-file database, cannot be found. The missing skeleton file may have been copied incorrectly or deleted accidentally.

Action

The skeleton netlist file is an optional file that is a stripped-down version of the netlist. This file allows a representation of the entire netlist to be stored in a single file. To use an srs file with the add_file command, the complete, multi-file database must be present in the *impDir/synwork/projectName_mult_srs* directory with the top-level srs file.

If you copy the multi-file database, make sure that the skeleton file is not corrupt when it is copied. Also, when the synthesis tool generates the multi-file database, the skeleton file could be missing if the software terminated while writing to this file. In these cases, you must regenerate the multi-file database.

BN570

@E: Instance <instance> in <instance>. <instance>. <instance> has a corrupt instof view (<instance>. <instance>. <instance>).

This message indicates that there is a corrupt netlist in memory.

Action

Contact Synopsys support and provide a testcase.

BN577

@E: The current version of distributed synthesis does not support entities with multiple architectures. Please create unique entities for the following entity architecture pairs.

The error is a notification that the design contains multiple architectures for the same entity. Distributed synthesis does not support this setup in certain cases.

Action

Create unique entries by duplicating the definition for the reported entity or architecture. See [BN578, on page 108](#) for a list of the multiple architecture pairs.

BN578

@E: Multiple architectures present for <name>:<name>

The error occurs because the design contains multiple architectures for the same entity, and the multiple architectures are being used in the design. Distributed synthesis does not support this setup in certain cases.

Action

Create unique entries by duplicating the definition for the reported entity or architecture.

BN581

@E: Global attribute <attributeName> is not supported in product <productName>.

You get this error because you defined the attribute as a global attribute, and the tool you are using does not support this attribute.

Action

Set the attribute on an instance.

BN583

@E: Encrypted block <blockName> not allowed for this vendor

The encrypted IP block for the design is not supported with the specified device.

Action

You can either remove the encrypted IP block from the design, or specify a device that supports encryption.

BN584

@E: Module <*moduleName*> is only valid for <*haps*> systems and cannot be used with <*haps*> systems

This can occur when a GPIO IP or system IP is instantiated in the RTL, but is specified with an incorrect HAPS target system. For example, the following IP specifications are designed for the HAPS target systems below:

- HAPS-80 – HAPS80_SYSTEMIP
- HAPS-70 – HAPS70_2000T_GPIO
- HAPS-DX7 – HAPSDX7_690T_GPIO

For example, if HAPS80_SYSTEMIP is instantiated in the RTL but HAPS-70 is specified as the target system, this generates an error.

Action

Make sure to specify the correct combination of GPIO or system IP to be used with the specified HAPS target system.

BN585

@E: Could not read the constraint database

The constraint database that resides in the cdb.srs file is missing from the multi-file (srs) database. This can occur if the multi-file database was not copied correctly or if the server does not have sufficient disk space for the multi-file database.

Action

Make sure that the multi-file database is copied correctly and has sufficient disk space on the server.

BN586

@E: Could not open file <fileName>

The specified file is missing from the multi-file (srs) database. This can occur if the multi-file database was not copied correctly or if the server does not have sufficient disk space for the multi-file database.

Action

Make sure that the multi-file database is copied correctly and has sufficient disk space on the server.

BN587

@E: File <fileName> is corrupt or does not exist

The specified file is missing from the multi-file (srs) database (generates [BN586](#)) or the file has been corrupted. This can occur if the multi-file database was not copied correctly, or if the server did not have enough disk space for the multi-file database.

Action

Make sure that the multi-file database is copied correctly and that the server has enough disk space.

BN589

@E: Netlist-locking mechanism cannot be enabled without System IP. Please enable System IP automation flow.

For HAPS-80 systems, the tool automatically inserts HAPS netlist locking in the design based on the system IP flow. If you attempt to disable system IP insertion, this error occurs causing the netlist-locking mechanism not to work.

Action

Do not disable the system IP automation flow with environment variables or a configuration switch.

BN591

@E: More debug information: The last object processed before the error was <objectName> in module <name>. <name>

This message appears when the software crashes, and provides more information to debug the cause of the crash.

Action

Check the object named in the message, because that was the last instance, module, or view the software was working on when the crash occurred. You can temporarily work around the issue by black-boxing the identified object or the module/view containing it.

If you are unable to debug the problem, extract the RTL for the indicated module or view. Use this to create a small testcase without proprietary information, which can then be used in communications with Synopsys support.

BN592

@E: More debug information: The last module processed before the error was <name>. <name>

This message appears when the software crashes, and provides additional information to debug the cause of the crash.

Action

Check the module named in the message, because that was the last module or view that the software was working on when the crash occurred. You can temporarily work around the issue by black-boxing the identified module/view.

If you are unable to debug the problem, extract the RTL for the indicated module or view. Use this to create a small testcase without proprietary information, which can then be used in communications with Synopsys support.

BN593

@E: More debug information: Cross-probe this line to find the last object parsed.

This message appears when the software crashes, and provides additional information to debug the cause of the crash.

Action

Follow the link to the HDL file that the software was working on when the crash occurred. You can temporarily work around the issue by black-boxing the module or view that contains the identified instance or module.

If you are unable to debug the problem, extract the RTL for the indicated module or view. Use this to create a small testcase without proprietary information, which can then be used in communications with Synopsys support.

BN594

@E: No additional debug information available

This message appears when the software crashes, and there is no additional information available.

Action

Retrace your steps and diagnose the problem as best you can. Extract the relevant RTL and create a small testcase without proprietary information, which can then be used in communications with Synopsys support.

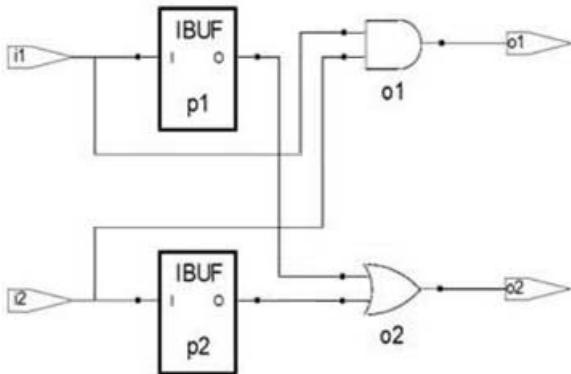
BN612

@E: Found <integer> ports with pad loads and non-pad loads

This message appears when the pad pin of an I/O pad has more than one connection and reports the number of ports with both pad loads and non-pad loads.

For example:

```
module top(i1,i2,o1,o2);
  input i1,i2;
  output o1,o2;
  wire w1,w2;
  IBUF p1(w1,i1);
  IBUF p2(w2,i2);
  assign o1=i1&i2;
  assign o2=w1|w2;
endmodule
```



The example generates the following error messages:

```
@E:BN245 : one.v(2) | Port 'i1' on chip 'top' drives 1 pad loads  
          (p1) and 1 non-pad loads  
@E:BN245 : one.v(2) | Port 'i2' on chip 'top' drives 1 pad loads  
          (p2) and 1 non-pad loads  
@E:BN612 : one.v(10) | Found 2 ports with pad loads and  
           non-pad loads
```

Action

Use this message in conjunction with the message or messages that immediately precede it. This message only reports the number of ports with the loading error, and to eliminate this message, you must fix the preceding errors. Make sure the pad pin listed in the preceding message(s) is only connected to the top-level port and is not driving anything else.

BN616

**@E: Overlapping bus index range for bus instances
 <instanceName>[<bit>:<bit>] and <instanceName>[<bit>:<bit>
 in module <moduleName>.**

This message is generated when there are two buses with overlapping ranges in the same netlist. For example, if buses a[5:3] and a[4:1] are specified, bus instances a[4] and a[3] overlap. Overlaps in bus indexes can cause unpredictable results when constraints and optimizations are applied.

Action

Avoid creating designs with overlapping bus ranges.

If the design was created with an older version, recompile the design with a current version of the tool.

BN623

@E: Fileinfo file <fileName> does not exist

The specified fileinfo.srs netlist file is missing from the multi-file (.srs) database. This can occur if the file was not copied to the multi-file database.

Action

Make sure that the fileinfo.srs netlist file was copied properly to the multi-file database. Otherwise, contact Synopsys support and provide a test case if the problem persists.

BN624

@E: Target technology can not be changed from <device> to <newDevice> after partitioning

This message occurs if you attempt to change the target technology of individual FPGAs after partitioning. This can happen when you use the following Tcl command to specify a new device while synthesizing a single FPGA database:

```
database apply_state -technology newDevice
```

Action

Do not change the target technology of any FPGA after partitioning in the multi-FPGA flow. Otherwise, assumptions about resources available on the FPGA or what IP to insert may no longer be valid.

BN625

@E: Speedgrade for target can not be changed from <speedgrade> to <newSpeedgrade> after partitioning

This message occurs if you attempt to change the speedgrade for the target technology of individual FPGAs after partitioning. This can happen when you use the following Tcl command to specify a new speedgrade for the target technology while synthesizing a single FPGA database:

```
option set speed_grade newSpeedgrade
```

Action

Do not change the speedgrade for the target technology of any FPGA after partitioning in the multi-FPGA flow. Otherwise, time-budgeting information generated earlier may be incorrect and can result in a failure to meet timing at the system level.

BN626

@E: UMR port direction mismatch on port <portName>

FPGA ports UMR_IN[11:0] and UMR_OUT[11:0] are created for HAPS systems to handle UMR logic. These ports have specific locations on the FPGA. You can also create ports in your design using the same locations. However, the direction of the ports must match their expected directions when the software tries to reuse these ports.

Action

Verify and make sure the port directions are defined correctly in your design. The directions for UMR_IN[11:0] are inputs and UMR_OUT[11:0] are outputs as implied by their names.

BN631

@E: Instance <instanceName> linked incorrectly

This error occurs when all the following conditions apply:

- Distributed compilation is enabled.
- Design is created with standard Verilog or mixed HDL language.
- Instance configuration or cross-module referencing (XMR) is used for the design.

For this scenario, the instance clause could not be resolved when either configuration or XMR is defined for a design that is running the distributed compilation flow.

Action

Contact Synopsys support to report this case. As a workaround, you can disable the Distributed Compilation option and rerun this design using the non-distributed flow.

BN632

@E: Could not link instance <*instanceName*>

This error occurs when the software cannot find a white box definition for the specified instance of a module or entity in a design running the distributed compilation flow.

Action

Contact Synopsys support to report this case. As a workaround, you can disable the Distributed Compilation option and rerun this design using the non-distributed flow.

BN643

@E: Incorrect linking for instance <*instanceName*>.

The above error occurs only when distributed compilation is enabled and indicates that the named instance was unable to be linked correctly by the linker.

Action

Contact Synopsys support to report this case. As a workaround, you can disable the distributed compilation option (option set distributed_compile 0) and rerun this design using the non-distributed flow.

BN667

@E: UMR_DATA_BITWIDTH and SYSTEM_UMR_DATA_BITWIDTH do not match for UMR CAPIM <portName>.

The above error occurs when there is a mismatch between the UMR_DATA_BITWIDTH and the SYSTEM_UMR_DATA_BITWIDTH for UMR CAPIM.

Action

Make sure that the UMR_DATA_BITWIDTH and the SYSTEM_UMR_-DATA_BITWIDTH are equal.

CHAPTER 5

BP Messages 100 – 113

BP102

@E: Could not initialize CDPL parallel processing server.

This error occurs because the CDPL infrastructure failed to initialize.

Action

The required corrective action varies, according to the cause of the error:

1. Make sure that the protocol you are using to run CDPL is one that lets you log onto other machines; for example, ssh or rsh. csh is not supported because it is a shell utility that does not let you log on to other machines.
2. Check the CDPL master.log and worker.log files for information on where to find details. Check for these messages:
 - “@N BP100 : “*path/file.log*”|Log messages from CDPL parallel processing package can be found in this file”
 - “@N BP101 : “*path/file.err*”|Error messages from CDPL parallel processing package can be found in this file”

For specifics about fixing the error, refer to the CDPL documentation, which is included as a PDF in the tool installation.

BP103

@E: Could not start the master process for CDPL parallel processing.

This error occurs because the tool could not start the master process for parallel processing with CDPL.

Action

Check the CDPL master.log and worker.log files for information on where to find details. Check for these messages:

- “@N BP100 : “*path/file.log*”|Log messages from CDPL parallel processing package can be found in this file”
- “@N BP101 : “*path/file.err*”|Error messages from CDPL parallel processing package can be found in this file”

For specifics about fixing the error, refer to the CDPL documentation, which is included as a PDF in the tool installation.

BP104

@N: Successful launch of CDPL server and master

The tool successfully initialized the CDPL server started the master process for parallel processing with CDPL.

Action

Informative message; no action required.

BP105

@E: Launch of CDPL server and master was unsuccessful

This error message displays because the tool could not initialize the CDPL server (BP102) or because it could not start the master process for parallel processing with CDPL (BP103).

Action

Check the CDPL master.log and worker.log files for information on the underlying failure (BP102 or BP103). Check for these messages:

- “@N BP100 : “*path/file.log*”|Log messages from CDPL parallel processing package can be found in this file”
- “@N BP101 : “*path/file.err*”|Error messages from CDPL parallel processing package can be found in this file”

For specifics about fixing the error, refer to the CDPL documentation, which is included as a PDF in the tool installation.

BP106

@E: <filename> file not found. Reinstall the product.

The scllp executable is missing from the tool installation hierarchy.

Action

Download the tool again and re-install the product. Check that the scllp executable is in the installation.

BP107

@E: Multiprocessing terminated because the scllp executable for checking out licenses could not be started.

The scllp executable for checking out licenses cannot be started. This situation could be caused by any of the following:

- Installation issues.
- The machine cannot create a new process.
- You do not have write permission to the directory.

Action

Check the log file for details, and fix the problem. For more detail about the problem once you have identified it, check the CDPL documentation, which is available as a PDF in the tool installation hierarchy.

If there is a problem with the executable, download the software again and reinstall it.

BP109

@E: CDPL initialization failed

This error message displays because the tool could not initialize CDPL for multiprocessing. Non-initialization can be due to many causes, which are described in additional error messages.

Action

Check the CDPL master.log and worker.log files for information on where to find details. Check for these messages:

- “@N BP100 : “*path/file.log*”|Log messages from PDPL parallel processing package can be found in this file”

- “@N BP101 : “*path/file.err*”|Error messages from CDPL parallel processing package can be found in this file”

For specifics about fixing the error, refer to the CDPL documentation, which is included as a PDF in the tool installation.

BP111

@E: CDPL_FPGA_HOST environment not set.

You get this error message if you have not set up the CDPL_FPGA_HOST environment variable, which is required for running CDPL distributed processing.

Action

Set the CDPL_FPGA_HOST environment variable to point to the CDPL executable and libraries. See the *User Guide* and the CDPL documentation for details about CDPL setup.

BP113

@E: Failed to create directory <dirName>; make sure you have write permission.

This error message appears when you run distributed processing and do not have write permission to the specified directory.

Action

Reset the permissions for the directory so that you can write to it, and rerun distributed processing.

CHAPTER 6

CD Messages 100 – 178

CD100

@E: Attribute <left> is not supported yet

An attribute was applied to an identifier that is not supported by that attribute which prevents the compiler from evaluating the expression.

VHDL supports the attribute 'left' on scalar types or subtypes and 'base' only on scalar types. In the test case below, the attribute 'base' is not applied to a scalar type which causes it to return an unknown value. The attribute 'left' is then applied to the unknown return type which causes the compiler to error out.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (A: in std_logic_vector(1 downto 0);
          B: in std_logic_vector(1 downto 0);
          C: out std_logic_vector(1 downto 0) );
end test;
```

```

architecture rtl of test is
subtype memory is integer range 0 to 1;
signal mem : memory;
begin
  c(mem'base'left)<=a(0) and b(1);
  C(0) <= a(1) and b(0);
end rtl;

```

Action

Apply the attribute 'left' to a suitable scalar type identifier such as memory as shown in the corrected test case below.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
  port (A: in std_logic_vector(1 downto 0);
        B: in std_logic_vector(1 downto 0);
        C: out std_logic_vector(1 downto 0));
end test;

architecture rtl of test is
subtype memory IS integer range 0 to 1;
begin
  c(memory'left)<=a(0) and b(1);
  C(0) <= a(1) and b(0);
end rtl;

```

CD104

@E: reverse_range not supported yet

A reverse_range attribute was applied to an unconstrained object. In the test case below, the attribute reverse_range is applied on the type natural which is unconstrained and which causes the compiler to error out.

```

library ieee;
use ieee.std_logic_1164.all;

```

```

entity comb is
    port (A: in std_logic_vector(1 downto 0);
          B: in std_logic_vector(1 downto 0);
          C: out std_logic_vector(natural'reverse_range));
end comb;

architecture rtl of comb is
begin
    c<= a and b;
end rtl;

```

Action

Apply the reverse_range attribute only on constrained objects. As shown in the corrected test case below, the reverse_range attribute is applied on the constrained array object A.

```

library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (A: in std_logic_vector(1 downto 0);
          B: in std_logic_vector(1 downto 0);
          C: out std_logic_vector(A' reverse_range));
end comb;

architecture rtl of comb is
begin
    c<= a and b;
end rtl;

```

CD109

@E: Cannot implement 'rightof of right most element in enumeration type

The 'rightof attribute is being used with the right-most value defined in the enumerated type. VHDL has a predefined set of attributes that provides information about the values included for certain types. The attributes associated with discrete (i.e., enumerated types and integers) as well as physical types (i.e., time) are 'pos, 'val, 'succ, 'pred, 'leftof, and 'rightof.

`T'rightof(x)` is defined as the value in `T` at position one to the right of `x`. For enumeration types, the position numbers start at zero for the first element listed and increase by one for each element to the right. In the following test case, `state_values'rightof(s1)` is illegal because the right-most element of the enumerated type `state_values` is `s1`.

```

library ieee;
use ieee.std_logic_1164.all;

entity stmchl is
    port (clk, in1, rst: in std_logic;
          out1: out std_logic );
end stmchl;

architecture behave of stmchl is
type state_values is (sx, s0, s1);
signal state, next_state: state_values;
begin
    process (clk, rst)
    begin
        if rst = '1' then
            state <= s0;
        elsif rising_edge(clk) then
            state <= next_state;
        end if;
    end process;

    process (state, in1)
    begin
-- set defaults for output and state
        out1 <= '0';
        next_state <= sx; -- catch missing assignments to next_state
        case state is
            when s0 =>
                if in1 = '0' then
                    out1 <='1';
                    next_state <= s1;
                else
                    out1 <= '0';
                    next_state <= state_values'rightof (s1);
                end if;
            when s1 =>
                if in1 = '0' then
                    out1 <='0';
                    next_state <= s0;
                else
                    out1 <= '1';
                end if;
        end case;
    end process;
end;

```

```
        next_state <= s1;
    end if;
    when sx =>
        next_state <= sx;
    end case;
end process;
end behave;
```

Action

Make sure that the ‘rightof’ is not used in association with the right-most element of the enumeration type. To eliminate this error in the above test case, change the ‘rightof’ reference as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity stmchl is
    port (clk, in1, rst: in std_logic;
          out1: out std_logic );
end stmchl;

architecture behave of stmchl is
type state_values is (sx, s0, s1);
signal state, next_state: state_values;
begin
    process (clk, rst)
    begin
        if rst = '1' then
            state <= s0;
        elsif rising_edge(clk) then
            state <= next_state;
        end if;
    end process;

    process (state, in1)
    begin
-- set defaults for output and state
        out1 <= '0';
        next_state <= sx; -- catch missing assignments to next_state
        case state is
            when s0 =>
                if in1 = '0' then
                    out1 <='1';
                    next_state <= s1;
                else
```

```

        out1 <= '0';
        next_state <= state_values'rightof (s0);
    end if;
when s1 =>
    if in1 = '0' then
        out1 <='0';
        next_state <= s0;
    else
        out1 <= '1';
        next_state <= s1;
    end if;
when sx =>
    next_state <= sx;
end case;
end process;
end behave;

```

CD110

@E: Cannot implement 'succ of right most element in enumeration type

The 'succ attribute is being used with the right-most value defined in the enumerated type. VHDL has a predefined set of attributes that provide information about the values included for certain types. The attributes associated with discrete (i.e., enumerated types and integers) as well as physical types (i.e., time) are 'pos, 'val, 'succ, 'pred, 'leftof, and 'rightof.

T'succ(x) is defined as the value in T at position one greater than that of x where T is the data type and x is a value of that type. For enumeration types, the position numbers start at zero for the first element listed and increase by one for each element to the right. In the following test case, state_values'succ (s1) is illegal because there is no value at a position that is greater than that of s1.

```

library ieee;
use ieee.std_logic_1164.all;

entity stmchl is
    port (clk, in1, rst: in std_logic;
          out1: out std_logic );
end stmchl;

```

```
architecture behave of stmchl is
type state_values is (sx, s0, s1);
signal state, next_state: state_values;
begin
    process (clk, rst)
begin
    if rst = '1' then
        state <= s0;
    elsif rising_edge(clk) then
        state <= next_state;
    end if;
end process;

process (state, in1)
begin
-- set defaults for output and state
out1 <= '0';
next_state <= sx; -- catch missing assignments to next_state
case state is
    when s0 =>
        if in1 = '0' then
            out1 <='1';
            next_state <= s1;
        else
            out1 <= '0';
            next_state <= state_values'succ (s1);
        end if;
    when s1 =>
        if in1 = '0' then
            out1 <='0';
            next_state <= s0;
        else
            out1 <= '1';
            next_state <= s1;
        end if;
    when sx =>
        next_state <= sx;
end case;
end process;
end behave;
```

Action

Make sure that the 'succ' is not used in association with the right-most element of the enumeration type. To eliminate this error in the above test case, change the 'succ' reference as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity stmchl is
    port (clk, in1, rst: in std_logic;
          out1: out std_logic );
end stmchl;

architecture behave of stmchl is
type state_values is (sx, s0, s1);
signal state, next_state: state_values;
begin
    process (clk, rst)
begin
    if rst = '1' then
        state <= s0;
    elsif rising_edge(clk) then
        state <= next_state;
    end if;
end process;

process (state, in1)
begin
-- set defaults for output and state
out1 <= '0';
next_state <= sx; -- catch missing assignments to next_state
    case state is
        when s0 =>
            if in1 = '0' then
                out1 <='1';
                next_state <= s1;
            else
                out1 <= '0';
                next_state <= state_values'succ (s0);
            end if;
        when s1 =>
            if in1 = '0' then
                out1 <='0';
                next_state <= s0;
            else
                out1 <= '1';
                next_state <= s1;
            end if;
    end case;
end;
```

```
        when sx =>
            next_state <= sx;
        end case;
    end process;
end behave;
```

CD111

@E: Cannot implement 'leftof of left most element in enumeration type

The 'leftof attribute is being used with the left-most value defined in the enumerated type. VHDL has a predefined set of attributes that provides information about the values included for certain types. The attributes associated with discrete (i.e., enumerated types and integers) as well as physical types (i.e., time) are 'pos, 'val, 'succ, 'pred, 'leftof, and 'rightof.

T'leftof(x) is defined as the value in T at the position one to the left of x, where T is the data type and x is a value of that type. For enumeration types, the position numbers start at zero for the first element listed and increase by one for each element to the right. In the following test case, state_values'leftof(sx) is illegal because the left-most element of the enumerated type state_values is sx.

```
library ieee;
use ieee.std_logic_1164.all;

entity stmchl is
    port (clk, in1, rst: in std_logic;
          out1: out std_logic );
end stmchl;

architecture behave of stmchl is
type state_values is (sx, s0, s1);
signal state, next_state: state_values;
begin
    process (clk, rst)
    begin
        if rst = '1' then
            state <= s0;
        elsif rising_edge(clk) then
            state <= next_state;
        end if;
    end process;
```

```

process (state, in1)
begin
-- set defaults for output and state
out1 <= '0';
next_state <= sx; -- catch missing assignments to next_stat
case state is
when s0 =>
    if in1 = '0' then
        out1 <='1';
        next_state <= s1;
    else
        out1 <= '0';
next_state <= state_values'leftof(sx);
    end if;
when s1 =>
    if in1 = '0' then
        out1 <='0';
        next_state <= s0;
    else
        out1 <= '1';
        next_state <= s1;
    end if;
when sx =>
    next_state <= sx;
end case;
end process;
end behave;

```

Action

Make sure that the 'leftof' is not used in association with the left-most element of the enumeration type. To eliminate this error in the above test case, change the 'leftof' reference as shown in the corrected test case below.

```

library ieee;
use ieee.std_logic_1164.all;

entity stmchl is
    port (clk, in1, rst: in std_logic;
          out1: out std_logic );
end stmchl;

```

```
architecture behave of stmchl is
type state_values is (sx, s0, s1);
signal state, next_state: state_values;
begin
    process (clk, rst)
begin
    if rst = '1' then
        state <= s0;
    elsif rising_edge(clk) then
        state <= next_state;
    end if;
end process;

process (state, inl)
begin
-- set defaults for output and state
out1 <= '0';
next_state <= sx; -- catch missing assignments to next_stat
case state is
    when s0 =>
        if inl = '0' then
            out1 <='1';
            next_state <= s1;
        else
            out1 <= '0';
            next_state <= state_values'leftof(s0);
        end if;
    when s1 =>
        if inl = '0' then
            out1 <='0';
            next_state <= s0;
        else
            out1 <= '1';
            next_state <= s1;
        end if;
    when sx =>
        next_state <= sx;
end case;
end process;
end behave;
```

CD112

@E: Cannot implement 'pred of left most element in enumeration type

The 'pred attribute is being used with the left-most value defined in the enumerated type. VHDL has a predefined set of attributes that provides information about the values included for certain types. The attributes associated with discrete (i.e., enumerated types and integers) as well as physical types (i.e., time) are 'pos, 'val, 'succ, 'pred, 'leftof, and 'rightof.

T'pred(x) is defined as the value in T at a position that is one less than that of x. For enumeration types, the position numbers start at zero for the first element listed and increase by one for each element to the right. In the following test case, state_values'pred(sx) is illegal as there is no value present at the position one less than sx.

```
library ieee;
use ieee.std_logic_1164.all;

entity stmchl is
    port (clk, in1, rst: in std_logic;
          out1: out std_logic );
end stmchl;

architecture behave of stmchl is
type state_values is (sx, s0, s1);
signal state, next_state: state_values;
begin
    process (clk, rst)
    begin
        if rst = '1' then
            state <= s0;
        elsif rising_edge(clk) then
            state <= next_state;
        end if;
    end process;

    process (state, in1)
    begin
-- set defaults for output and state
        out1 <= '0';
        next_state <= sx; -- catch missing assignments to next_state
        case state is
            when s0 =>
                if in1 = '0' then
```

```

        out1 <='1';
        next_state <= s1;
    else
        out1 <= '0';
        next_state <= state_values'pred (sx);
    end if;
when s1 =>
    if in1 = '0' then
        out1 <='0';
        next_state <= s0;
    else
        out1 <= '1';
        next_state <= s1;
    end if;
when sx =>
    next_state <= sx;
end case;
end process;
end behave;

```

Action

Make sure that the 'pred' is not used in association with the left-most element of the enumeration type. To eliminate this error in the above test case, change the 'pred' reference as shown in the corrected test case below.

```

library ieee;
use ieee.std_logic_1164.all;

entity stmchl is
    port (clk, in1, rst: in std_logic;
          out1: out std_logic );
end stmchl;

architecture behave of stmchl is
type state_values is (sx, s0, s1);
signal state, next_state: state_values;
begin
    process (clk, rst)
begin
    if rst = '1' then
        state <= s0;
    elsif rising_edge(clk) then
        state <= next_state;
    end if;
end process;

```

```

process (state, in1)
begin
-- set defaults for output and state
out1 <= '0';
next_state <= sx; -- catch missing assignments to next_state
case state is
    when s0 =>
        if in1 = '0' then
            out1 <='1';
            next_state <= s1;
        else
            out1 <= '0';
            next_state <= state_values'pred (s0);
        end if;
    when s1 =>
        if in1 = '0' then
            out1 <='0';
            next_state <= s0;
        else
            out1 <= '1';
            next_state <= s1;
        end if;
    when sx =>
        next_state <= sx;
end case;
end process;
end behave;

```

CD117

@E: attribute <high1> is not defined for this object type

An undefined attribute was applied to a type for which that particular attribute is not predefined. In the following test case, integer'high1 is illegal as 'high1 is not a predefined attribute for the type Integer.

```

package my_pack is
subtype my_type is integer range 14 to integer'high1;
end my_pack;

```

```
library ieee;
use ieee.std_logic_1164.all;
use work.my_pack.all;

entity test is
generic (k: my_type:=2);
    port (q: out std_logic_vector(k-1 downto 0);
          clk : in std_logic;
          d : in std_logic_vector(k-1 downto 0));
end test;

architecture test of test is
begin
    process(clk)
    begin
        if(clk'event and clk='1') then
            q<=d;
        end if;
    end process;
end test;
```

Action

Make sure that the attribute being applied to a type has been previously defined. To eliminate this error in the above test case, change the attribute 'high1' to some predefined attribute for Integer such as 'high' as shown in the corrected test case below.

```
package my_pack is
subtype my_type is integer range 14 to integer'high;
end my_pack;

library ieee;
use ieee.std_logic_1164.all;
use work.my_pack.all;

entity test is
generic (k: my_type:=2);
port (q: out std_logic_vector(k-1 downto 0);
      clk : in std_logic;
      d : in std_logic_vector(k-1 downto 0));
end test;
```

```

architecture test of test is
begin
  process(clk)
  begin
    if(clk'event and clk='1') then
      q<=d;
    end if;
  end process;
end test;

```

CD118

@E: event and stable attributes apply only to signals

The ‘event attribute or the ‘stable attribute is being applied to any identifier that is not a signal. In VHDL, the attributes ‘event and ‘stable are defined only for signals and not for variables. In the following test case, temp’event is illegal as temp is a variable and ‘event is not defined for variables.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
port (din: in std_logic_vector(1 downto 0);
      clk: in std_logic;
      q: out std_logic_vector(1 downto 0));
end test;

architecture dff of test is
begin
  process(clk)
  variable temp: std_logic;
  begin
    temp:=clk;
    if(temp'event and temp='1') then
      q<=din;
    end if;
  end process;
end dff;

```

Action

Make sure that any 'event' or 'stable' attribute is applied to a signal. To eliminate this error in the above test case, declare temp as a signal as shown below in the corrected test case.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
port (din: in std_logic_vector(1 downto 0);
      clk: in std_logic;
      q: out std_logic_vector(1 downto 0));
end test;

architecture dff of test is
signal temp: std_logic;
begin
temp<=clk;
process(temp)
begin
  if(temp'event and temp='1') then
    q<=din;
  end if;
end process;
end dff;
```

CD120

@E: Could not implement attribute 'rightof'

The attribute 'rightof' is being applied with the value V not being of type T. If T is a discrete type, physical type, or a subtype, the attribute T'rightof(V) returns the value that is to the right of the value V in T. In the following test case, the attribute 'rightof' is applied with the value of V being equal to 0, but the identifier 0 does not belong to the type my_type which causes the compiler to error out.

```
library ieee;
use ieee.std_logic_1164.all;
```

```

entity test is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic_vector(2 downto 0));
end test;

architecture rtl of test is
subtype my_type is integer range 0 to 2;
begin
    c<= (2=>a,0=> b,my_type'rightof('0')=>a);
end rtl;

```

Action

Change the value of V to a legal element such as 0 belonging to the type my_type as shown in the corrected test case below.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic_vector(2 downto 0));
end test;

architecture rtl of test is
subtype my_type is integer range 0 to 2;
begin
    c<= (2=>a,0=> b,my_type'rightof(0)=>a);
end rtl;

```

CD121

@E: Could not implement attribute 'leftof'

The 'leftof' attribute is being applied with the value V not being of type T. When T is a discrete type, physical type, or subtype, the attribute T'leftof(V) returns the value that is to the left of the value V in T. In the test case below, the 'leftof' attribute is applied with the value of V being equal to 1. Because the identifier '1' does not belong to the type my_type, the compiler errors out.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic_vector(2 downto 0));
end test;

architecture rtl of test is
subtype my_type is integer range 0 to 2;
begin
    c<= (2=>a,1=> b,my_type'leftof('1')=>a);
end rtl;
```

Action

Change the value of V to a legal element such as 1 belonging to the type my_type as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic_vector(2 downto 0));
end test;

architecture rtl of test is
subtype my_type is integer range 0 to 2;
begin
    c<= (2=>a,1=> b,my_type'leftof(1)=>a);
end rtl;
```

CD122

@E: Attribute of design unit must be in declaration section of design unit

An attribute to be applied to an entity (design unit) is declared outside of the corresponding entity declaration. In VHDL, the attribute being applied to a design unit such as an entity must be declared and used only within that design unit. In the following test case, the `syn_global_buffers` attribute, which applies to the entity `seq`, is declared inside the architecture which causes the compiler to error out.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (q: out std_logic_vector(19 downto 0);
          d : in std_logic_vector(19 downto 0);
          reset: in std_logic_vector(19 downto 0);
          clk : in std_logic_vector( 19 downto 0));
end seq;

architecture test of seq is
attribute syn_global_buffers: integer;
attribute syn_global_buffers of seq:entity  is 10;
begin
    process(clk,reset)
    begin
        for i in 0 to 19 loop
            if (reset(i) ='1') then
                q(i)<='0';
            elsif clk(i) ='1' and clk(i)'event then
                q(i)<=d(i);
            end if;
        end loop;
    end process;
end test;
```

Action

Make sure that any attribute to be applied to an entity is declared and used only within the entity declaration. In the corrected test case below, the `syn_global_buffers` attribute appears within the entity declaration which eliminates the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (q: out std_logic_vector(19 downto 0);
          d : in std_logic_vector(19 downto 0);
          reset: in std_logic_vector(19 downto 0);
          clk : in std_logic_vector( 19 downto 0));

attribute syn_global_buffers: integer;
attribute syn_global_buffers of seq:entity  is 10;
end seq;

architecture test of seq is
begin
    process(clk,reset)
    begin
        for i in 0 to 19 loop
            if (reset(i) ='1') then
                q(i)<='0';
            elsif clk(i) ='1' and clk(i)'event then
                q(i)<=d(i);
            end if;
        end loop;
    end process;
end test;
```

CD123

@E: attribute of architecture must be in declaration section of architecture

The attribute being applied to the architecture is not present in the architecture declaration. In VHDL, when an attribute is being applied to an architecture, the attribute cannot appear outside of the architecture declaration. The compiler checks this error by crosschecking the class of the attribute with its context. In the following test case, the attribute `syn_black_box`, which applies to architecture `test`, erroneously appears in the entity declaration.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity test is
port (q: out std_logic_vector(7 downto 0);
      d : in std_logic_vector(7 downto 0);
      clk : in std_logic);
attribute syn_black_box: boolean;
attribute syn_black_box of test:architecture  is true;
end test;

architecture test of test is
begin
  process(clk)
  begin
    if(clk'event and clk='1') then
      q<=d;
    end if;
  end process;
end test;
```

Action

Make sure that any attribute being applied to the architecture is declared as well as applied to the architecture only in the architecture declaration. To eliminate this error in the above test case, apply the attribute in the architecture declaration as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
port (q: out std_logic_vector(7 downto 0);
      d : in std_logic_vector(7 downto 0);
      clk : in std_logic);
end test;

architecture test of test is
attribute syn_black_box: boolean;
attribute syn_black_box of test:architecture  is true;

begin
  process(clk)
  begin
    if(clk'event and clk='1') then
      q<=d;
    end if;
  end process;
end test;
```

CD124

@E: Expecting attribute name

An attribute is declared, but the attribute name is not in the specification. The syntax for attribute declaration and specification is as follows:

```
--attribute declaration
--attribute attribute-name : value_type;

--attribute specification
--attribute attribute-name of item-names : name-class is
expression;
```

In the following test case, the attribute name on signal temp in the attribute specification is missing which results in the above message.

```
library ieee;
use ieee.std_logic_1164.all;
library synplify;
use synplify.attributes.all;

entity adder is
    port (a, b, cin: in std_logic;
          sum, cout:out std_logic );
end adder;

architecture behave of adder is
signal temp:std_logic;
attribute of temp:signal is true;
begin
    sum <= (a xor b xor cin;
    cout <= (a and b) or (a and cin) or (b and cin);
end temp;
```

Action

Make sure that the syntax for attributes in VHDL contains the attribute name. To eliminate the error in the above test case, insert the attribute name in the attribute specification as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;
library synplify;
use synplify.attributes.all;
```

```

entity adder is
    port ( a, b, cin: in std_logic;
           sum, cout:out std_logic );
end adder;

architecture behave of adder is
signal temp:std_logic;
attribute syn_keep of temp:signal is true;
begin
    sum <= (a xor b xor cin);
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;

```

If you are using any of the predefined Synopsys FPGA synthesis directives, include the following library-use clause to eliminate having to redefine the directive or attribute each time it is used.

```

library synplify;
use synplify.attributes.all;

```

CD125

@E: Reference to unknown attribute definition <*syn_noclockbuf*>

The compiler found an attribute specification without a corresponding attribute declaration. In VHDL, all attributes must be declared using an attribute declaration before applying the attribute specification. The attribute declaration is of the form

attribute attributeName : valueType;

and the attribute specification is of the form:

attribute attributeName of itemNames : nameClass is expression;

In the test case below, the *syn_noclockbuf* attribute does not have a corresponding declaration which causes the error.

```

library ieee;
use ieee.std_logic_1164.all;

```

```
entity test is
    port (q: out std_logic_vector(7 downto 0);
          d : in std_logic_vector(7 downto 0);
          clk : in std_logic);
attribute syn_noclockbuf of clk :signal is true;
end test;

architecture test of test is
begin
    process(clk)
    begin
        if(clk'event and clk='1') then
            q<=d;
        end if;
    end process;
end test;
```

Action

Be sure to declare the attribute before the attribute specification as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (q: out std_logic_vector(7 downto 0);
          d : in std_logic_vector(7 downto 0);
          clk : in std_logic);
attribute syn_noclockbuf: Boolean;
attribute syn_noclockbuf of clk :signal is true;
end test;

architecture test of test is
begin
    process(clk)
    begin
        if(clk'event and clk='1') then
            q<=d;
        end if;
    end process;
end test;
```

CD126

@E: Expecting identifier

Record elements were assigned explicitly in an aggregate statement and each element used record tags to reference its elements. The following test case generates the above message.

```
library ieee;
use ieee.std_logic_1164.all;

entity error25 is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic_vector(1 downto 0) );
end error25;

architecture rtl of error25 is
TYPE color IS (red, yellow, blue, white);
TYPE rec_ptr IS
RECORD
    d1 : color;
    d2 : std_logic;
END RECORD;
signal rp : rec_ptr;
begin
    rp <= (rp.d1 => red, rp.d2 => A);
    C <= (B, rp.d2);
end rtl;
```

Action

Remove the record tag in the assignment statement. To eliminate this error in the above test case, change the record tags in the aggregate statement as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity error25 is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic_vector(1 downto 0) );
end error25;
```

```

architecture rtl of error25 is
TYPE color IS (red, yellow, blue, white);
TYPE rec_ptr IS
RECORD
    d1 : color;
    d2 : std_logic;
END RECORD;
signal rp : rec_ptr;
begin
    rp <= (d1 => red, d2 => A);
    C <= (B, rp.d2);
end rtl;

```

Record types belong to a composite class that consists of elements of different types. The element declaration for each record declares one or more identifiers and their types. The following code segments illustrate record definition and assignment.

```

--declaration of identifiers and their types
type opcode is (add, sub, mult, div, and, or, nop);
type mode is range 0 to 3;
type address is std_logic_vector (7 downto 0);

--record declaration containing 3 fields of different types
type instruction_format is record
    op:opcode;
    md:mode;
    addr:address;
end record;

--signal of type instruction_format initialized
signal instr:instruction_format := (nop, 0, "00000000");

--assignment of record fields
instr.op <= add;
instr.md <= 1;
instr.addr <= "00000001";

--complete record assignment using named association
instr <= (op => add, md => 1, addr => "00000001");

--complete record assignment using positional association
instr <= (add, 1, "00000001");

```

CD127

@E: Expecting all, others or identifier list

The identifier list (or an all or others keyword) followed the object. In the test case below, object type signal precedes its identifier (tmp) which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity reg_reg is
    port (data, clk, a: in std_logic;
          qb: out std_logic);
end reg_reg;

architecture async_set_reset of reg_reg is
signal qa: std_logic;
signal tmp: std_logic;
attribute syn_probe : boolean;
attribute syn_probe of signal : tmp is true;
begin
    tmp <= data and a;
    setreset: process (clk)
    begin
        if rising_edge(clk) then
            qb <= qa;
        end if;
    end process setreset;

    process (clk)
    begin
        if rising_edge(clk) then
            qa <= tmp;
        end if;
    end process;
end;
```

Action

Make sure that an identifier list precedes its objects.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity reg_reg is
    port (data, clk, a: in std_logic;
          qb: out std_logic);
end reg_reg;

architecture async_set_reset of reg_reg is
signal qa: std_logic;
signal tmp: std_logic;
attribute syn_probe : boolean;
attribute syn_probe of tmp : signal is true;

begin
tmp <= data and a;
setreset: process (clk)
begin
    if rising_edge(clk) then
        qb <= qa;
    end if;
end process setreset;

process (clk)
begin
    if rising_edge(clk) then
        qa <= tmp;
    end if;
end process;
end;
```

CD128

@E: Expecting : before class of object(s)

The attribute specification did not have a colon between the class of the object (*nameClass*) and the *itemNames*. In VHDL, the syntax for an attribute specification is of the form:

attribute attributeName of itemNames : nameClass is expression;

In the test case below, the attribute specification is missing the colon separator between the *itemName* *clk* and the *nameClass* signal which causes the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (q: out std_logic_vector(7 downto 0);
          d : in std_logic_vector(7 downto 0);
          clk : in std_logic );
attribute syn_noclockbuf : boolean;
attribute syn_noclockbuf of clk signal is true;
end test;

architecture test of test is
begin
    process(clk)
    begin
        if(clk'event and clk='1') then
            q<=d;
        end if;
    end process;
end test;

```

Action

Be sure to include the colon between the class of the *itemNames* and the object (*nameClass*) as shown in the corrected test case given below.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (q: out std_logic_vector(7 downto 0);
          d : in std_logic_vector(7 downto 0);
          clk : in std_logic );
attribute syn_noclockbuf : boolean;
attribute syn_noclockbuf of clk : signal is true;
end test;

architecture test of test is
begin
    process(clk)
    begin
        if(clk'event and clk='1') then
            q<=d;
        end if;
    end process;
end test;

```

CD129

@E: Attributes on literals are not supported

The compiler encountered an attribute specification in which the class of the item is literal (attributes on literals are not supported). In VHDL the syntax for an attribute specification is

```
attribute attributeName of itemNames : nameClass is expression;
```

In the test case below, the compiler encounters an attribute specification in which the class of item `my_literal` is specified as a literal.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (q: out std_logic_vector(7 downto 0);
          d : in std_logic_vector(7 downto 0);
          clk : in std_logic);
end test;

architecture test of test is
type mvl is ('U','0','1','Z');
signal temp:mvl;
attribute my_literal : mvl;
attribute my_literal of temp : literal is '1';
begin
    process(clk)
    begin
        if(clk'event and clk=temp) then
            q<=d;
        end if;
    end process;
end test;
```

Action

Make sure that the design has no attributes on literals as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;
```

```

entity test is
    port (q: out std_logic_vector(7 downto 0);
          d : in std_logic_vector(7 downto 0);
          clk : in std_logic);
end test;

architecture test of test is
type mvl is ('U','0','1','Z');
begin
    process(clk)
    begin
        if(clk'event and clk=std_logic(mvl'rightof('0'))) then
            q<=d;
        end if;
    end process;
end test;

```

CD131

@E: Expecting type of object to add attribute to

The compiler found an attribute specification that does not mention the class (type) of the object. In VHDL, the syntax for an attribute specification is of the form:

attribute attributeName of itemName : nameClass is expression;

In the test case below, the attribute specification does not have the class name for the object `clk` following the colon which causes the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (q: out std_logic_vector(7 downto 0);
          d : in std_logic_vector(7 downto 0);
          clk : in std_logic);
attribute syn_noclockbuf : boolean;
attribute syn_noclockbuf of clk : is true;
end test;

```

```
architecture test of test is
begin
    process(clk)
    begin
        if(clk'event and clk='1') then
            q<=d;
        end if;
    end process;
end test;
```

Action

Be sure to define the object type for the applied attribute as shown in the corrected test case below (object clk is assigned type signal).

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (q: out std_logic_vector(7 downto 0);
          d : in std_logic_vector(7 downto 0);
          clk : in std_logic);
attribute syn_noclockbuf : boolean;
attribute syn_noclockbuf of clk : signal is true;
end test;

architecture test of test is
begin
    process(clk)
    begin
        if(clk'event and clk='1') then
            q<=d;
        end if;
    end process;
end test;
```

CD132

@E: Expecting keyword is

The keyword is, which is required between *nameClass* and *expression*, was omitted in the attribute specification. In VHDL, the syntax for an attribute specification is of the form:

```
attribute attributeName of itemName : nameClass is expression;
```

In the test case below, the keyword is in the attribute specification is missing from bewteen the class signal and the expression true which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (q: out std_logic_vector(7 downto 0);
          d : in std_logic_vector(7 downto 0);
          clk : in std_logic);
attribute syn_noclockbuf : boolean;
attribute syn_noclockbuf of clk :signal true;
end test;

architecture test of test is
begin
    process(clk)
    begin
        if(clk'event and clk='1') then
            q<=d;
        end if;
    end process;
end test;
```

Action

Make sure to include the keyword is after the class signal as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;
```

```

entity test is
    port (q: out std_logic_vector(7 downto 0);
          d : in std_logic_vector(7 downto 0);
          clk : in std_logic);
attribute syn_noclockbuf : boolean;
attribute syn_noclockbuf of clk :signal is true;
end test;

architecture test of test is
begin
    process(clk)
    begin
        if(clk'event and clk='1') then
            q<=d;
        end if;
    end process;
end test;

```

CD133

@E: Expecting : for attribute declaration or "of" for value

The above error occurs when either

- a colon is omitted after the attribute name in the attribute declaration
- the keyword of is omitted after the attribute name in the attribute specification

In VHDL, the syntax for an attribute declaration is

attribute attributeName : valueType;

and the attribute specification is of the form:

attribute attributeName of itemNames : nameClass is expression;

In the test case below, the keyword of following the attribute name `syn_noclockbuf` is missing which causes the error.

```

library ieee;
use ieee.std_logic_1164.all;

```

```
entity simuledff is
    port (q: out std_logic_vector(7 downto 0);
          d : in std_logic_vector(7 downto 0);
          clk : in std_logic);
attribute syn_noclockbuf : boolean;
attribute syn_noclockbuf  clk :signal is true;
end simuledff;

architecture test of simuledff is
begin
    process(clk)
    begin
        if(clk'event and clk='1') then
            q<=d;
        end if;
    end process;
end test;
```

Action

Make sure that either a colon follows the attribute name in the attribute declaration or the keyword of follows the attribute name in the attribute specification. The corrected test case is shown below.

```
library ieee;
use ieee.std_logic_1164.all;

entity simuledff is
    port (q: out std_logic_vector(7 downto 0);
          d : in std_logic_vector(7 downto 0);
          clk : in std_logic);
attribute syn_noclockbuf : boolean;
attribute syn_noclockbuf of clk :signal is true;
end simuledff;

architecture test of simuledff is
begin
    process(clk)
    begin
        if(clk'event and clk='1') then
            q<=d;
        end if;
    end process;
end test;
```

CD134

@W: No such identifier, <q2>, of proper type in current declarative region

A signal in an attribute is not declared within the corresponding declarative region such as between the architecture definition line and the begin. In the test case below, qrs is not defined within the declarative region.

```
architecture async_set_reset of dff1 is
.....
begin
```

but is used in the attribute specification:

```
attribute syn_preserve : boolean;
attribute syn_preserve of qrs2 : signal is true;
```

The syn_preserve attribute is not honored in this case.

```
-- test case
library ieee;
use ieee.std_logic_1164.all;

entity dff1 is
    port (data, clk, reset, set: in std_logic;
          qrs1,qrs2: out std_logic );
end dff1;

architecture async_set_reset of dff1 is
attribute syn_preserve : boolean;
attribute syn_preserve of qrs2 : signal is true;
begin
setreset:
    process (clk, reset, set)
begin
    if reset = '1' then
        qrs1 <= '0';
        qrs2 <= '0';
    elsif set = '1' then
        qrs1 <= '1';
        qrs2 <= '1';
    elsif rising_edge(clk) then
```

```
        qrs1 <= data;
        qrs2 <= data;
    end if;
end process setreset;
end async_set_reset;
```

Action

Make sure that the HDL code declares identifiers before using them. To eliminate the error in the above test case, apply the `syn_preserve` synthesis directive to port `qrs2` in the entity declaration as shown in the corrected test case below.

```
-- test case
library ieee;
use ieee.std_logic_1164.all;

entity dff1 is
    port (data, clk, reset, set: in std_logic;
          qrs1,qrs2: out std_logic );
attribute syn_preserve : boolean;
attribute syn_preserve of qrs2 : signal is true;
end dff1;

architecture async_set_reset of dff1 is
begin

setreset:
process (clk, reset, set)
begin
    if reset = '1' then
        qrs1 <= '0';
        qrs2 <= '0';
    elsif set = '1' then
        qrs1 <= '1';
        qrs2 <= '1';
    elsif rising_edge(clk) then
        qrs1 <= data;
        qrs2 <= data;
    end if;
end process setreset;
end async_set_reset;
```

The synthesis attribute `syn_preserve` is honored for port `qrs`.

CD135

@W: syn_encoding attribute has been renamed to syn_enum_encoding.
Change syn_encoding usage to:
synthesis syn_enum_encoding for upward compatibility

The synthesis directive syn_encoding is being used for encoding enumerated types. To encode enumerated types, use the syn_enum_encoding directive in the following format:

```
attribute syn_enum_encoding: string;
attribute syn_enum_encoding of state_type : type is "001 010 101";
```

In the test case below, the syn_encoding attribute is incorrectly used.

```
library ieee;
use ieee.std_logic_1164.all;
library ieee;
use ieee.std_logic_1164.all;

entity shift_enum is
    port (clk, rst: bit;
          O: out std_logic_vector(2 downto 0) );
end shift_enum;

architecture behave of shift_enum is
type state_type is (S0, S1, S2);
attribute syn_encoding: string;
attribute syn_encoding of state_type : type is "001 010 101";
signal machine : state_type;
begin
    process (clk, rst)
    begin
        if rst = '1' then
            machine <= S0;
        elsif clk = '1' and clk'event then
            case machine is
                when S0 => machine <= S1;
                when S1 => machine <= S2;
                when S2 => machine <= S0;
                when others => null;
            end case;
        end if;
    end process;
```

```

with machine select
    O <= "001" when S0,
    "010" when S1,
    "101" when S2;
end behave;

```

Action

Make sure that `syn_enum_encoding` is used instead of `syn_encoding` when encoding enumerated types. The `syn_enum_encoding` directive is honored only when the FSM compiler is turned off. To eliminate the warning in the above test case, replace `syn_encoding` attribute references with `syn_enum_encoding` as shown in the corrected test case below.

```

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

entity shift_enum is
    port (clk, rst: bit;
          O: out std_logic_vector(2 downto 0) );
end shift_enum;

architecture behave of shift_enum is
type state_type is (S0, S1, S2);
attribute syn_enum_encoding: string;
attribute syn_enum_encoding of state_type : type is "001 010 101";

signal machine : state_type;
begin
    process (clk, rst)
    begin
        if rst = '1' then
            machine <= S0;
        elsif clk = '1' and clk'event then
            case machine is
                when S0 => machine <= S1;
                when S1 => machine <= S2;
                when S2 => machine <= S0;
                when others => null;
            end case;
        end if;
    end process;

```

```

with machine select
    O <= "001" when S0,
    "010" when S1,
    "101" when S2;
end behave;
```

CD137

@W: Enum count mismatch

The synthesis directive `syn_enum_encoding` is being used for encoding enumerated types and a mismatch exists between the number of states and the encodings specified. In the following test case, `state_type` is a type of three enumerations `S0`, `S1`, and `S2`, but only two enumerated encodings are associated with these states.

```

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

entity shift_enum is
    port (clk, rst: bit;
          O: out std_logic_vector(2 downto 0) );
end shift_enum;

architecture behave of shift_enum is
type state_type is (S0, S1, S2);
attribute syn_enum_encoding: string;
attribute syn_enum_encoding of state_type :
    type is "001 010 ";
signal machine : state_type;
begin
    process (clk, rst)
    begin
        if rst = '1' then
            machine <= S0;
        elsif clk = '1' and clk'event then
            case machine is
                when S0 => machine <= S1;
                when S1 => machine <= S2;
```

```

        when S2 => machine <= S0;
        when others => null;
    end case;
end if;
end process;

-- with machine select
  O <= "001" when S0,
  "010" when S1,
  "101" when S2;
end behave;

```

Action

Make sure that the number of enumerations and the encodings associated with them specified by the `syn_enum_encoding` attribute match. To eliminate the warning for the above test case, correct the number of encodings in the `syn_enum_encoding` of `state_type` statement as shown below.

```

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

entity shift_enum is
  port (clk, rst: bit;
        O: out std_logic_vector(2 downto 0) );
end shift_enum;

architecture behave of shift_enum is
type state_type is (S0, S1, S2);
attribute syn_enum_encoding: string;
attribute syn_enum_encoding of state_type : type is "001 010 011";
signal machine : state_type;
begin
  process (clk, rst)
begin
  if rst = '1' then
    machine <= S0;
  elsif clk = '1' and clk'event then
    case machine is
      when S0 => machine <= S1;
      when S1 => machine <= S2;

```

```

        when S2 => machine <= S0;
        when others => null;
    end case;
end if;
end process;

-- with machine select
O <= "001" when S0,
"010" when S1,
"101" when S2;
end behave;

```

The enumerated type is honored when the FSM compiler is turned off.

CD138

@W: Enum code width should be the same.

When using user-defined encoding for the `syn_enum_encoding` directive, the code is of a different width. In VHDL designs, the `syn_enum_encoding` directive defines how enumerated data types are implemented when the FSM compiler is disabled. For this type of enumerated code, a state machine cannot be implemented and either sequential, gray, or one-hot encoding is used according to the number of states. In the test case below, while defining the value of the `syn_enum_encoding` directive (001 0100 101), the code widths are different which causes the warning.

```

library ieee;
use ieee.std_logic_1164.all;

entity test2 is
    port (clk, rst : bit;
          O : out std_logic_vector(2 downto 0));
end cd138;

architecture behave of test2 is
type state_type is (S0, S1, S2);
attribute syn_enum_encoding: string;
attribute syn_enum_encoding of state_type :
type is "001 0100 101";
signal machine : state_type;
begin
    process (clk, rst)

```

```

begin
    if rst = '1' then
        machine <= S0;
    elsif clk = '1' and clk'event then
        case machine is
            when S0 => machine <= S1;
            when S1 => machine <= S2;
            when S2 => machine <= S0;
        end case;
    end if;
end process;

with machine select
    O <= "001" when S0,
    "010" when S1,
    "101" when S2;
end behave;

```

Action

Make sure that the same code width is specified within the enumerated code. In the corrected test case below, the enumerated code has a width of three bits which allows the compiler to correctly implement the states.

```

library ieee;
use ieee.std_logic_1164.all;

entity test2 is
    port (clk, rst : bit;
          O : out std_logic_vector(2 downto 0));
end cd138;

architecture behave of test2 is
type state_type is (S0, S1, S2);
attribute syn_enum_encoding: string;
attribute syn_enum_encoding of state_type : type is "001 100 101";
signal machine : state_type;
begin
    process (clk, rst)
    begin
        if rst = '1' then
            machine <= S0;
        elsif clk = '1' and clk'event then
            case machine is
                when S0 => machine <= S1;

```

```

        when S1 => machine <= S2;
        when S2 => machine <= S0;
    end case;
end if;
end process;

with machine select
  O <= "001" when S0,
  "010" when S1,
  "101" when S2;
end behave;

```

CD139

@W: Unknown encoding value <one cold>

The syn_enum_encoding (or syn_encoding) synthesis directive was assigned a value other than sequential, onehot, or gray. The syn_enum_encoding directive can be used to specify any pattern of bits such as:

```
attribute syn_enum_encoding of state_type : type is "001 010 0011";
```

However, when specified as the name of the encoding, only these encodings are supported: sequential, onehot, or gray.

```

library ieee;
use ieee.std_logic_1164.all;

entity shift_enum is
  port (clk, rst: bit;
        O: out std_logic_vector(2 downto 0) );
end shift_enum;

architecture behave of shift_enum is
type state_type is (S0, S1, S2);
attribute syn_enum_encoding: string;
attribute syn_enum_encoding of state_type :
type is "one cold";
signal machine : state_type;
begin
  process (clk, rst)
begin
  if rst = '1' then

```

```

        machine <= S0;
      elsif clk = '1' and clk'event then
        case machine is
          when S0 => machine <= S1;
          when S1 => machine <= S2;
          when S2 => machine <= S0;
          when others => null;
        end case;
      end if;
    end process;
  with machine select
    O <= "001" when S0,
    "010" when S1,
    "101" when S2;
  end behave;

```

Action

Make sure that only the values sequential, gray, and onehot are used when specifying encoding. If you need to specify an encoding other than these, use the enumerated type of encoding. For example, if the state machine needs an enumeration of one_cold, the syn_enum_encoding statement would appear as follows in the corrected test case.

```

library ieee;
use ieee.std_logic_1164.all;

entity shift_enum is
  port (clk, rst: bit;
        O: out std_logic_vector(2 downto 0) );
end shift_enum;

architecture behave of shift_enum is
type state_type is (S0, S1, S2);
attribute syn_enum_encoding: string;
attribute syn_enum_encoding of state_type : type is "110 101 011";
signal machine : state_type;
begin
  process (clk, rst)
begin
  if rst = '1' then
    machine <= S0;
  elsif clk = '1' and clk'event then
    case machine is
      when S0 => machine <= S1;
      when S1 => machine <= S2;

```

```

        when S2 => machine <= S0;
        when others => null;
    end case;
end if;
end process;
with machine select
    O <= "001" when S0,
    "010" when S1,
    "101" when S2;
end behave;

```

The `syn_enum_encoding` directive is only honored when the FSM compiler is turned off.

CD145

@E: Formal, "<q>", and actual agree on type but not on size

The widths of the ports in the entities of the components (formal) did not match their corresponding signals in the entity (actual). The nomenclature for actual and formal is shown in the following port examples. In the following test case, `q`, `data`, `clk`, `rst`, and `r_l` are formal ports (ports of the component/sub-level entity `rotate`), and `q_out`, `reg_out`, `clk`, `rst`, and `r_l` are actuals (signals used in the instantiation of the entity `rotate`).

```

library ieee;
use ieee.std_logic_1164.all;

entity rotate is
    port (q: buffer std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst, r_l: in std_logic );
end rotate;

library ieee;
use ieee.std_logic_1164.all;

entity top_level is
    port (q_out: inout std_logic_vector (7 downto 0);
          a, b: in std_logic_vector (7 downto 0);
          sel, r_l, clk, rst: in std_logic );
end top_level;

```

architecture structural of top_level is

```
component rotate -- component declaration for rotate
    port (q: buffer std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst, r_l: in std_logic );
end component;

-- declare the internal signals here
signal mux_out: std_logic_vector (7 downto 0);
signal reg_out: std_logic_vector (6 downto 0);
begin -- structural description begins
...
--instantiate a rotate, name it inst2, and wire it up
inst2: rotate port map (q_out, reg_out, clk, rst, r_l);
```

The width mismatch between the formal (q) and actual(`reg_out`) for the instantiation `reg8` in the entity `top_level` in the following test case causes the compiler to error out with the above message.

```
--Example of a hierarchical design
-----
--First define the lower level modules: muxhier, reg8, & rotate
-----

library ieee;
use ieee.std_logic_1164.all;

entity muxhier is
    port (outvec: out std_logic_vector (7 downto 0);
          a_vec, b_vec: in std_logic_vector (7 downto 0);
          sel: in std_logic );
end muxhier;

architecture mux_design of muxhier is -- mux
begin
with sel select
    outvec <= a_vec when '1',
                  b_vec when '0',
                  "XXXXXXXX" when others;
end mux_design;

library ieee;
use ieee.std_logic_1164.all;
```

```
entity reg8 is
    port (q: buffer std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst: in std_logic );
end reg8;

architecture reg8_design of reg8 is -- eight bit register
begin
    process (clk, rst)
    begin
        if rst = '1' then
            q <= X"00";
        elsif rising_edge(clk) then
            q <= data;
        end if;
    end process;
end reg8_design;

library ieee;
use ieee.std_logic_1164.all;
entity rotate is
    port (q: buffer std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst, r_l: in std_logic );
end rotate;

architecture rotate_design of rotate is
-- rotates bits or loads
-- when r_l is high, it rotates; if low, it loads data
begin
    process (clk, rst)
    begin
        if rst = '1' then
            q <= X"00";
        elsif rising_edge(clk) then
            if r_l = '1' then
                q <= q (6 downto 0 ) & q (7);
            else
                q <= data;
            end if;
        end if;
    end process;
end rotate_design;
```

```
-- Top level
-----
library ieee;
use ieee.std_logic_1164.all;

entity top_level is
    port (q: inout std_logic_vector (7 downto 0);
          a, b: in std_logic_vector (7 downto 0);
          sel, r_l, clk, rst: in std_logic);
end top_level;

architecture structural of top_level is

component muxhier -- component declaration for mux
    port (outvec: out std_logic_vector (7 downto 0);
          a_vec, b_vec: in std_logic_vector (7 downto 0);
          sel: in std_logic);
end component;

component reg8 -- component declaration for reg8
    port (q: buffer std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst: in std_logic);
end component;

component rotate -- component declaration for rotate
    port (q: buffer std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst, r_l: in std_logic);
end component;

-- declare the internal signals here
signal mux_out: std_logic_vector (7 downto 0);
signal reg_out: std_logic_vector (6 downto 0);
begin -- structural description begins

-- instantiate a mux, name it inst1, and wire it up
-- here we connect the mux with positional port mapping (by
position)
inst1: muxhier port map (mux_out, a, b, sel);

-- instantiate a rotate, name it inst2, and wire it up
inst2: rotate port map (q, reg_out, clk, rst, r_l);
```

```
-- instantiate a reg8, name it inst3, and wire it up
-- reg8 is connected with named port mapping (by name)
-- the port connections can be given in any order
-- Note that the local signal names are on the right of the
-- '>=' mapping operators, and the signal names from the
-- component declaration are on the left.

--instantiate a data reg, name it inst3, and wire it up
inst3: reg8
    port map (clk => clk, data => mux_out,
               q => reg_out, rst => rst);

end structural;
```

Action

Make sure that the width and type of the formal data type and actual ports in the component declaration and instantiation match. To eliminate the error in the above test case, change the width `reg_out` in the signal statement from (6 downto 0) to (7 downto 0).

```
-----
--Example of a hierarchical design
-----
--First define the lower level modules: muxhier, reg8, & rotate
-----

library ieee;
use ieee.std_logic_1164.all;

entity muxhier is
    port (outvec: out std_logic_vector (7 downto 0);
          a_vec, b_vec: in std_logic_vector (7 downto 0);
          sel: in std_logic );
end muxhier;

architecture mux_design of muxhier is -- mux
begin
with sel select
    outvec <= a_vec when '1',
                  b_vec when '0',
                  "XXXXXXXX" when others;
end mux_design;

library ieee;
use ieee.std_logic_1164.all;
```

```
entity reg8 is
    port (q: buffer std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst: in std_logic );
end reg8;

architecture reg8_design of reg8 is -- eight bit register
begin
    process (clk, rst)
    begin
        if rst = '1' then
            q <= X"00";
        elsif rising_edge(clk) then
            q <= data;
        end if;
    end process;
end reg8_design;

library ieee;
use ieee.std_logic_1164.all;
entity rotate is
    port (q: buffer std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst, r_l: in std_logic );
end rotate;

architecture rotate_design of rotate is
-- rotates bits or loads
-- when r_l is high, it rotates; if low, it loads data
begin
    process (clk, rst)
    begin
        if rst = '1' then
            q <= X"00";
        elsif rising_edge(clk) then
            if r_l = '1' then
                q <= q (6 downto 0 ) & q (7);
            else
                q <= data;
            end if;
        end if;
    end process;
end rotate_design;
```

```
-----  
-- Top level  
-----  
  
library ieee;  
use ieee.std_logic_1164.all;  
  
entity top_level is  
    port (q: inout std_logic_vector (7 downto 0);  
          a, b: in std_logic_vector (7 downto 0);  
          sel, r_l, clk, rst: in std_logic);  
end top_level;  
  
architecture structural of top_level is  
  
component muxhier -- component declaration for mux  
    port (outvec: out std_logic_vector (7 downto 0);  
          a_vec, b_vec: in std_logic_vector (7 downto 0);  
          sel: in std_logic);  
end component;  
  
component reg8 -- component declaration for reg8  
    port (q: buffer std_logic_vector (7 downto 0);  
          data: in std_logic_vector (7 downto 0);  
          clk, rst: in std_logic);  
end component;  
  
component rotate -- component declaration for rotate  
    port (q: buffer std_logic_vector (7 downto 0);  
          data: in std_logic_vector (7 downto 0);  
          clk, rst, r_l: in std_logic);  
end component;  
  
-- declare the internal signals here  
signal mux_out: std_logic_vector (7 downto 0);  
signal reg_out: std_logic_vector (7 downto 0);  
begin -- structural description begins  
  
-- instantiate a mux, name it inst1, and wire it up  
-- here we connect the mux with positional port mapping (by position)  
inst1: muxhier port map (mux_out, a, b, sel);  
  
-- instantiate a rotate, name it inst2, and wire it up  
inst2: rotate port map (q, reg_out, clk, rst, r_l);
```

```
-- instantiate a reg8, name it inst3, and wire it up
-- reg8 is connected with named port mapping (by name)
-- the port connections can be given in any order
-- Note that the local signal names are on the right of the
-- '>=' mapping operators, and the signal names from the
-- component declaration are on the left.

--instantiate a data reg, name it inst3, and wire it up
inst3: reg8
    port map (clk => clk, data => mux_out,
               q => reg_out, rst => rst);

end structural;
```

When this error occurs, a warning appears indicating which line to edit.

```
@W: Port map width mismatch (8 => 7) on port data of component reg8
```

CD146

@E: Formal, "<*data*>", and actual disagree on type

The port type in the entities of the components (formal) did not match the type for the signals in the entity (actual). The nomenclature for actual and formal is shown in the following port examples. In the following test case, q, data, clk, rst, and r_l are formal ports (ports of the component/sub-level entity rotate), and q_out, reg_out, clk, rst, and r_l are actuals (signals used in the instantiation of the entity rotate).

```
library ieee;
use ieee.std_logic_1164.all;

entity rotate is
    port (q: buffer std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst, r_l: in std_logic );
end rotate;

library ieee;
use ieee.std_logic_1164.all;
```

```

entity top_level is
    port (q_out: inout std_logic_vector (7 downto 0);
          a, b: in std_logic_vector (7 downto 0);
          sel, r_l, clk, rst: in std_logic );
end top_level;

architecture structural of top_level is
component rotate -- component declaration for rotate
    port (q: buffer std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst, r_l: in std_logic );
end component;

-- declare the internal signals here
signal mux_out: std_logic_vector (7 downto 0);
signal reg_out: std_logic_vector (6 downto 0);
begin -- structural description begins
    .....
    -- instantiate a rotate, name it inst2, and wire it up
    inst2: rotate port map (q_out, reg_out, clk, rst, r_l);

```

The following test case causes the above error due to the type mismatch between the formal (q) and actual (reg_out) for the instantiation rotate in the entity top_level.

```

-- Example of a hierarchical design
-----
-- First define the lower level modules: muxhier, reg8, & rotate
-----

library ieee;
use ieee.std_logic_1164.all;

entity muxhier is
    port (outvec: out std_logic_vector (7 downto 0);
          a_vec, b_vec: in std_logic_vector (7 downto 0);
          sel: in std_logic );
end muxhier;

architecture mux_design of muxhier is -- mux
begin
with sel select
    outvec <= a_vec when '1',
                  b_vec when '0',

```

```
        "XXXXXXXX" when others;
end mux_design;

library ieee;
use ieee.std_logic_1164.all;

entity reg8 is
    port (q: buffer std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst: in std_logic );
end reg8;

architecture reg8_design of reg8 is -- 8-bit register
begin
    process (clk, rst)
    begin
        if rst = '1' then
            q <= X"00";
        elsif rising_edge(clk) then
            q <= data;
        end if;
    end process;
end reg8_design;

library ieee;
use ieee.std_logic_1164.all;

entity rotate is
    port (q: buffer std_logic_vector (7 downto 0);
          data: in bit_vector (7 downto 0);
          clk, rst, r_l: in std_logic );
end rotate;

architecture rotate_design of rotate is
-- rotates bits or loads
-- when r_l is high, it rotates; if low, it loads data
begin
    process (clk, rst)
    begin
        if rst = '1' then
            q <= X"00";
        elsif rising_edge(clk) then
            if r_l = '1' then
                q <= q (6 downto 0 ) & q (7);
            else
                q <= data;
            end if;
        end if;
    end process;
end rotate_design;
```

```
        end if;
    end process;
end rotate_design;

-----
--      Top level
-----

library ieee;
use ieee.std_logic_1164.all;

entity top_level is
    port (q: inout std_logic_vector (7 downto 0);
          a, b: in std_logic_vector (7 downto 0);
          sel, r_l, clk, rst: in std_logic);
end top_level;

architecture structural of top_level is

component muxhier -- component declaration for mux
    port (outvec: out std_logic_vector (7 downto 0);
          a_vec, b_vec: in std_logic_vector (7 downto 0);
          sel: in std_logic );
end component;

component reg8 -- component declaration for reg8
    port (q: buffer std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst: in std_logic );
end component;

component rotate -- component declaration for rotate
    port (q: buffer std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst, r_l: in std_logic );
end component;

-- declare the internal signals here
signal mux_out: std_logic_vector (7 downto 0);
signal reg_out: std_logic_vector (7 downto 0);
begin -- structural description begins

--instantiate a mux, name it inst1, and wire it up
--here we connect the mux with positional port mapping (by position)
inst1:muxhier port map (mux_out, a, b, sel);

--instantiate a rotate, name it inst2, and wire it up
inst2: rotate port map (q, reg_out, clk, rst, r_l);
```

```
--instantiate a data reg, name it inst3, and wire it up
inst3: reg8
    port map (clk => clk, data => mux_out,
               q => reg_out, rst => rst);
end structural;
```

Action

Make sure that the type and width of the formal data type and actual ports in the component declaration and instantiation match. To eliminate the error in the above test case, match the port type (`std_logic_vector`) of the actual (`reg_out`) by editing the entity `rotate` is statement that defines the width of the formal data.

```
-- Example of a hierarchical design
-----
-- First define the lower level modules: muxhier, reg8, & rotate
-----

library ieee;
use ieee.std_logic_1164.all;

entity muxhier is
    port (outvec: out std_logic_vector (7 downto 0);
          a_vec, b_vec: in std_logic_vector (7 downto 0);
          sel: in std_logic );
end muxhier;

architecture mux_design of muxhier is -- mux
begin
with sel select
    outvec <= a_vec when '1',
                b_vec when '0',
                "XXXXXXXX" when others;
end mux_design;

library ieee;
use ieee.std_logic_1164.all;

entity reg8 is
    port (q: buffer std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst: in std_logic );
end reg8;
```

```
architecture reg8_design of reg8 is -- 8-bit register
begin
    process (clk, rst)
    begin
        if rst = '1' then
            q <= X"00";
        elsif rising_edge(clk) then
            q <= data;
        end if;
    end process;
end reg8_design;

library ieee;
use ieee.std_logic_1164.all;

entity rotate is
    port (q: buffer std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst, r_l: in std_logic );
end rotate;

architecture rotate_design of rotate is
-- rotates bits or loads
-- when r_l is high, it rotates; if low, it loads data
begin
    process (clk, rst)
    begin
        if rst = '1' then
            q <= X"00";
        elsif rising_edge(clk) then
            if r_l = '1' then
                q <= q (6 downto 0 ) & q (7);
            else
                q <= data;
            end if;
        end if;
    end process;
end rotate_design;

-----
-- Top level
-----

library ieee;
use ieee.std_logic_1164.all;
```

```
entity top_level is
    port (q: inout std_logic_vector (7 downto 0);
          a, b: in std_logic_vector (7 downto 0);
          sel, r_l, clk, rst: in std_logic);
end top_level;

architecture structural of top_level is

component muxhier -- component declaration for mux
    port (outvec: out std_logic_vector (7 downto 0);
          a_vec, b_vec: in std_logic_vector (7 downto 0);
          sel: in std_logic );
end component;

component reg8 -- component declaration for reg8
    port (q: buffer std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst: in std_logic );
end component;

component rotate -- component declaration for rotate
    port (q: buffer std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst, r_l: in std_logic );
end component;

-- declare the internal signals here
signal mux_out: std_logic_vector (7 downto 0);
signal reg_out: std_logic_vector (7 downto 0);
begin -- structural description begins

--instantiate a mux, name it inst1, and wire it up
--here we connect the mux with positional port mapping (by
position)
inst1: muxhier port map (mux_out, a, b, sel);

--instantiate a rotate, name it inst2, and wire it up
inst2: rotate port map (q, reg_out, clk, rst, r_l);

--instantiate a data reg, name it inst3, and wire it up
inst3: reg8
    port map (clk => clk, data => mux_out,
              q => reg_out, rst => rst);

end structural;
```

When this error occurs, a warning like the one below appears indicating which line to edit.

@W: Port type mismatch between component and entity

CD147

@E: Variable <data> should have a constrained type

An array data type was defined, but did not have a specified range. In the test case below, the port data is defined as a std_logic_vector, but has no range specified.

```
-- 4 bit up-counter with load and asynchronous low reset
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity counter is
    port (clk, reset, load: in std_logic;
          data: in std_logic_vector;
          count: out std_logic_vector (7 downto 0) );
end counter;

architecture behave of counter is
signal count_i : std_logic_vector (7 downto 0);
begin
    process (clk, reset)
    begin
        if (reset = '0') then
            count_i <= "00000000";
        elsif rising_edge(clk) then
            if load = '1' then
                count_i <= data;
            else
                count_i <= count_i + '1';
            end if;
        end if;
    end process;
    count <= count_i;
end behave;
```

Action

Make sure that all vectors are defined with definite bounds. If the width of a vector is 1, use (0 to 0) to describe the vector. To eliminate the error in the above test case, change the data port to contain a specified range.

```
-- 4 bit up-counter with load and asynchronous low reset
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity counter is
    port (clk, reset, load: in std_logic;
          data: in std_logic_vector (7 downto 0);
          count: out std_logic_vector (7 downto 0) );
end counter;

architecture behave of counter is
signal count_i : std_logic_vector (7 downto 0);
begin
    process (clk, reset)
    begin
        if (reset = '0') then
            count_i <= "00000000";
        elsif rising_edge(clk) then
            if load = '1' then
                count_i <= data;
            else
                count_i <= count_i + '1';
            end if;
        end if;
    end process;
    count <= count_i;
end behave;
```

CD148

@E: Deferred constant <compare_width> was never given a value

A deferred constant was encountered in the HDL code. Deferred constants are constants defined in package declarations with no value associated with them. In the following test case, compare_width is declared in a package, but is not assigned a value which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;

package my_package is
    constant compare_width : integer;
end;

-- Comparator
library ieee;
use ieee.std_logic_1164.all;
use work.my_package.all;

entity compare is
    port (a, b: in std_logic_vector ((compare_width-1) downto 0);
          equal: out std_logic );
end compare;

architecture behave of compare is
begin
    equal <= '1' when a = b else
        '0';
end behave;
```

Action

Make sure that all declared constants have definite values. Deferred constants (constants defined in package declarations with no value specified) must be assigned a value either in the package declaration or in the package body. To eliminate the error in the above test case, assign a value to the deferred constant.

```
library ieee;
use ieee.std_logic_1164.all;
```

```

package my_package is
  constant compare_width : integer := 16;
end;

-- Comparator
library ieee;
use ieee.std_logic_1164.all;
use work.my_package.all;

entity compare is
  port (a, b: in std_logic_vector ((compare_width-1) downto 0);
        equal: out std_logic );
end compare;

architecture behave of compare is
begin
  equal <= '1' when a = b else
    '0';
end behave;

```

CD150

@E: Width mismatch, variable <variableName> has width <width>, value <entries>

The number of entries defined for the variable is greater than or less than the expected width. The bus variable, bus width, and the number of entries are reported in the message. The example below illustrates a width mismatch error. The bus variable (`err`) has a width of 8, but only seven entries.

```

library IEEE;
use IEEE.std_logic_1164.all;

entity and2 is
  port (A0: in STD_LOGIC;
        A1: in STD_LOGIC;
        Y: out STD_LOGIC );
end and2;

```

```
architecture and2 of and2 is
  signal err : std_logic_vector (7 downto 0):=
    '1'&'0'&'1'&'0'&'1'&'0'&'1';
begin
  Y <= A0 and A1;
end and2;
```

Action

Make sure that the width of the variable and the number of entries match.

CD155

@E: No such builtin <*and2*>

A user-defined function is mapped to a builtin which is not a predefined function name or type. The => operator specifies a builtin implementation for a type or function. Builtins are the predefined functions and types defined by the language and, as such, do not require the function body to be explicitly specified. In the test case below, the function my_and is mapped to a non-existent builtin name and2 which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

package my_pack is
  function my_and (a, b: std_logic_vector)
    return std_logic_vector =>"and2";
end my_pack;

library ieee;
use ieee.std_logic_1164.all;
use work.my_pack.all;

entity test is
  port (clk: in std_logic;
        a_in, b_in: in std_logic_vector(3 downto 0);
        d_out: out std_logic_vector(3 downto 0) );
end test;
```

```
architecture beh of test is
begin
    process (clk)
    begin
        if falling_edge(clk) then
            d_out <= my_and(a_in, b_in);
        end if;
    end process;
end beh;
```

Action

Make sure to provide a predefined builtin name such as and as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;
package my_pack is
function my_and (a, b: std_logic_vector)
    return std_logic_vector =>"and";
end my_pack;

library ieee;
use ieee.std_logic_1164.all;
use work.my_pack.all;

entity test is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0) );
end test;

architecture beh of test is
begin
    process (clk)
    begin
        if falling_edge(clk) then
            d_out <= my_and(a_in, b_in);
        end if;
    end process;
end beh;
```

CD156

@E: Expecting subprogram or operator name

A subprogram specification does not have a subprogram name (i.e., the subprogram is a function or procedure) or operator name (i.e., the subprogram is an overloaded operator specification). The general syntax of a subprogram specification for a function body is:

```
function functionName (parameterList) return returnType
```

In the syntax statement, *functionName* is a regular function or an overloaded operator name. The general syntax of a subprogram specification for a procedure body is:

```
procedure procedureName (parameterList)
```

In the syntax statement, *procedureName* is the name of the procedure.

In the test case below, the error results from the function body not having a function name.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0));
end test;

architecture beh of test is
function (a, b: std_logic_vector) return std_logic_vector
    is variable temp: std_logic_vector(a'range);
begin
    temp:= a and b;
    return temp;
end ;

begin
    process (clk)
    begin
        if rising_edge(clk) then
            d_out <= my_and(a_in, b_in);
        end if;
    end process;
```

```
end beh;
```

Action

Be sure to provide the subprogram name (`my_and`) in the subprogram specification as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0) );
end test;

architecture beh of test is
function my_and (a, b: std_logic_vector) return std_logic_vector
    is variable temp: std_logic_vector(a'range);
begin
    temp:= a and b;
    return temp;
end ;

begin
    process (clk)
    begin
        if rising_edge(clk) then
            d_out <= my_and(a_in, b_in);
        end if;
    end process;
end beh;
```

CD157

@E: Expecting return type name

A subprogram specification or declaration does not have a *returnType*. The general syntax of a subprogram specification for a function body is:

function *functionName* (*parameterList*) return *returnType*

In the test case below, the function specification for the function `my_and` does not have a *returnType* which causes the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0) );
end test;

architecture beh of test is
function my_and (a, b: std_logic_vector) return
    is variable temp : std_logic_vector(a'range);
begin
    temp := a and b;
    return temp;
end function;

begin
    process (clk)
    begin
        if rising_edge(clk) then
            d_out <= my_and(a_in, b_in);
        end if;
    end process;
end beh;
```

Action

Be sure to provide a *returnType* in the subprogram specification as shown in the corrected test case (*returnType* is `std_logic_vector`).

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0) );
end test;
```

```

architecture beh of test is
function my_and (a, b: std_logic_vector) return std_logic_vector
  is variable temp : std_logic_vector(a'range);
begin
  temp := a and b;
  return temp;
end function;

begin
  process (clk)
  begin
    if rising_edge(clk) then
      d_out <= my_and(a_in, b_in);
    end if;
  end process;
end beh;

```

CD158

@E: Expecting builtin name

A user-defined function is mapped to a builtin function and the builtin name following the => operator is not enclosed in double quotes. The => operator specifies a builtin implementation for a type or function. Builtins are the predefined functions and types defined by the language and are enclosed between double quote marks (""). In the test case below, the function declaration of the function my_and specifies the builtin name (and) without the enclosing quote marks which causes the error.

```

library ieee;
use ieee.std_logic_1164.all;

package my_pack is
  function my_and (a, b: std_logic_vector)
    return std_logic_vector => and;
end my_pack;

library ieee;
use ieee.std_logic_1164.all;
use work.my_pack.all;

```

```
entity test is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0) );
end test;

architecture beh of test is
begin
    process (clk)
    begin
        if falling_edge(clk) then
            d_out <= my_and(a_in, b_in);
        end if;
    end process;
end beh;
```

Action

Be sure to enclose the builtin name (and) in double quotes as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

package my_pack is
function my_and (a, b: std_logic_vector)
    return std_logic_vector => "and";
end my_pack;

library ieee;
use ieee.std_logic_1164.all;
use work.my_pack.all;

entity test is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0) );
end test;
```

```

architecture beh of test is
begin
  process (clk)
  begin
    if falling_edge(clk) then
      d_out <= my_and(a_in, b_in);
    end if;
  end process;
end beh;

```

CD159

@E: Expecting subprogram name <*my_and*>

The *subprogramName* (if present) at the end of the statement did not match the *subprogramName* in the subprogram specification. The typical format for a subprogram body is:

```

subprogram_specification is
  subprogram_item_declarations
begin
  subprogram_statements
end [function] [procedure] [subprogramName];

```

In the test case below, the function name *my_and2* does not match the function name *my_and* in the function specification which causes the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
  port (clk: in std_logic;
        a_in, b_in: in std_logic_vector(3 downto 0);
        d_out: out std_logic_vector(3 downto 0));
end test;

```

```

architecture beh of test is
function my_and (a, b: std_logic_vector) return std_logic_vector
    is variable temp: std_logic_vector(a'range);
begin
    temp:= a and b;
    return temp;
end my_and2;

begin
    process (clk)
begin
    if falling_edge(clk) then
        d_out <= my_and(a_in, b_in);
    end if;
end process;

end beh;

```

Action

Make sure that the subprogram name at the end of the subprogram body is the same name used in the subprogram specification. In the corrected test case below, the function name (`my_and`) in the subprogram specification and in the `end` statement are the same.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0));
end test;

architecture beh of test is
function my_and (a, b: std_logic_vector) return std_logic_vector
    is variable temp: std_logic_vector(a'range);
begin
    temp:= a and b;
    return temp;
end my_and;

```

```

begin
  process (clk)
  begin
    if falling_edge(clk) then
      d_out <= my_and(a_in, b_in);
    end if;
  end process;

end beh;

```

CD161

@E: Can't apply subtype to non object alias

An attempt was made to create an alias for a subtype of a non-object. The syntax for an alias declaration is:

```
alias identifier [: identifierType] is itemName;
```

In the syntax statement, *itemName* can be either:

- an object such as a constant, signal, variable, or file
- a non-object such as a function name, literal, type name, or attribute name.

For a non-object alias, the type information (*identifierType*) is not required.

In the test case below, identifier k is not declared in the scope which causes the compiler to treat the identifier as a non-object; when (1 downto 0) is encountered following the identifier, k(1 downto 0) is considered a subtype of k which results in the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
  port (a : in std_logic_vector(3 downto 0);
        b : in std_logic_vector(3 downto 0);
        c: out std_logic_vector(3 downto 0));
end test ;

```

```
architecture arc of test is
alias a_lower: std_logic_vector(1 downto 0) is k(1 downto 0);
begin
    c(1 downto 0)<= a_lower and b(1 downto 0);
    c(3 downto 2)<= a(3 downto 2) or b(3 downto 2);
end arc;
```

Action

Make sure to declare the alias according to the syntax. In the corrected test case below, the alias a_lower is used for the lower half of signal a which can have a subtype.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (a : in std_logic_vector(3 downto 0);
          b : in std_logic_vector(3 downto 0);
          c: out std_logic_vector(3 downto 0));
end test ;

architecture arc of test is
alias a_lower: std_logic_vector(1 downto 0)is a(1 downto 0);
begin
    c(1 downto 0)<= a_lower and b(1 downto 0);
    c(3 downto 2)<= a(3 downto 2) or b(3 downto 2);
end arc;
```

CD164

@E: <For> declaration is not legal here

An illegal declaration of a VHDL construct was encountered. In the following test case, the construct for is declared immediately after the process statement without an intervening begin statement which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity check is
    port (a,b: in std_logic_vector(1 downto 0);
          q:out std_logic_vector(1 downto 0));
end check;

architecture test of check is
begin
    process(a,b)
        for i in b'reverse_range loop
            q(i)<= a(i) and b(i);
        end loop;
    end process;
end test;
```

Action

Make sure that the VHDL constructs are legal. In the corrected test case below, a for construct is included after the begin statement inside the process block to eliminate the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity check is
    port (a,b: in std_logic_vector(1 downto 0);
          q:out std_logic_vector(1 downto 0));
end check;

architecture test of check is
begin
    process(a,b)
        begin
            for i in b'reverse_range loop
                q(i)<= a(i) and b(i);
            end loop;
        end process;
    end test;
```

CD166

@E: function body is not legal here

A function body has been defined within a package declaration. In VHDL, a function body only can be defined in either a package body or in an architectural declaration. In the test case below, the function my_and is defined inside the package declaration which causes error.

```
library ieee;
use ieee.std_logic_1164.all;

package my_pack is
    function my_and (a, b: std_logic_vector) return
    std_logic_vector
        is variable temp: std_logic_vector(a'range);
    begin
        temp:= a and b;
        return temp;
        end my_and;
end my_pack;

library ieee;
use ieee.std_logic_1164.all;
use work.my_pack.all;

entity seq is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0));
end seq;

architecture beh of seq is
begin
    process (clk)
    begin
        if falling_edge(clk) then
            d_out <= my_and(a_in, b_in);
        end if;
    end process;
end beh;
```

Action

Make sure to first declare the function specification in the package declaration and then define the function body in the package body as shown in the corrected test case given below.

```
library ieee;
use ieee.std_logic_1164.all;

package my_pack is -- package declaration
    function my_and (a, b: std_logic_vector) return
std_logic_vector;
-- above line is function specification
end my_pack;

package body my_pack is -- package body
function my_and (a, b: std_logic_vector) return std_logic_vector
is
    variable temp: std_logic_vector(a'range); -- function body
begin
    temp:= a and b;
    return temp;
end my_and;
end my_pack;

library ieee;
use ieee.std_logic_1164.all;
use work.my_pack.all;

entity seq is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0) );
end seq;

architecture beh of seq is
begin
    process (clk)
begin
    if falling_edge(clk) then
        d_out <= my_and(a_in, b_in);
    end if;
end process;
end beh;
```

CD169

@E: Illegal declaration

The compiler found multiple instances with the same instance name. Here is an example that results in the above error.

```
library ieee;
use ieee.std_logic_1164.all;

entity fa is
port (a,b,cin : in std_logic ;
      sum, carry : out std_logic);
end fa;

architecture df of fa is
begin
    sum <= a xor b xor cin;
    carry <= ((a and b) or (a and cin) or (b and cin));
end;

entity fa4 is
generic( N : integer := 4);
port (a,b : in std_logic_vector(N-1 downto 0);
      cin : in std_logic_vector(0 to 0);
      sum : out std_logic_vector(N-1 downto 0);
      carry : out std_logic);
end fa4;

architecture structural of fa4 is
component fa
port (a,b,cin : in std_logic;
      sum, carry : out std_logic);
end component;

signal temp : std_logic_vector(N downto 0);
begin
    temp(0) <= cin(0);
    u1 : fa port map(a=>a(0), b=>b(0),
                      cin=>temp(0), sum=>sum(0), carry=>temp(1));
    u1 : fa port map(a=>a(1), b=>b(1), cin=>temp(1),
                      sum=>sum(1), carry=>temp(2));
    u3 : fa port map(a=>a(2), b=>b(2), cin=>temp(2),
```

```

        sum=>sum(2), carry=>temp(3));
u4 : fa port map(a=>a(3), b=>b(3), cin=>temp(3),
                  sum=>sum(3), carry=>temp(4));
                  carry<=temp(4);
end structural;

```

In the above example, the same label u1 is used twice causes the error.

Action

Use different label names during instantiation of same components. In the above code, changing the second label to u2 removes the error:

```

u2 : fa port map(a=>a(1), b=>b(1), cin=>temp(1),
                  sum=>sum(1), carry=>temp(2));

```

CD170

@E: Expecting simple name

While instantiating a component using named association, there is a string of characters enclosed in single or double quotes that is used as a formal. In the test case below, the string “1111” is used as a formal which causes the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity and1 is
    port (a_in, b_in: in std_logic_vector(3 downto 0);
          c_out: out std_logic_vector(3 downto 0) );
end and1;

architecture and1 of and1 is
begin
    c_out<=a_in and b_in;
end and1;

library ieee;
use ieee.std_logic_1164.all;
use work.and1.all;

```

```
entity top is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0) );
end top;

architecture beh of top is
component andl is
    port (a_in, b_in: in std_logic_vector(3 downto 0);
          c_out: out std_logic_vector(3 downto 0) );
end component;

signal temp:std_logic_vector(3 downto 0);
begin


- ul: andl port map(a_in=>a_in,"1111"=>b_in ,c_out=>temp);


    process (clk)
    begin
        if falling_edge(clk) then
            d_out <=temp;
        end if;
    end process;
end beh;
```

Action

Change the formal as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity andl is
    port (a_in, b_in: in std_logic_vector(3 downto 0);
          c_out: out std_logic_vector(3 downto 0) );
end andl;

architecture andl of andl is
begin
    c_out<=a_in and b_in;
end andl;

library ieee;
use ieee.std_logic_1164.all;
use work.andl.all;
```

```

entity top is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0) );
end top;

architecture beh of top is
component and1 is
    port (a_in, b_in: in std_logic_vector(3 downto 0);
          c_out: out std_logic_vector(3 downto 0) );
end component;

signal temp:std_logic_vector(3 downto 0);
begin
u1: and1 port map(a_in=>a_in,b_in=>b_in ,c_out=>temp);
process (clk)
begin
    if falling_edge(clk) then
        d_out <=temp;
    end if;
end process;
end beh;

```

CD173

@E: Can't associate actual <k>

The compiler encountered a non-existent actual that was mapped to a value. In a component instantiation, the actual specified in the named port mapping must be present in the entity declaration of the component being mapped. In the test case below, the non-existent generic k is being mapped which causes the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity HA is
    port (U,V: in bit;
          X,Y: out bit);
end HA;

```

```
architecture beh of Ha is
begin
    x<= u xor v;
    y<= u and v;
end beh;

entity test is
    port (A, B: in bit;
          SUM, CARRY: out bit);
end test;

architecture beh of test is
component HA
    port (U,V: in bit;
          X,Y: out bit);
end component;

begin

```

Action

Be sure to use only valid actuals for named mappings in a component instantiation. This association is illustrated in the corrected test cases below where only the actuals that are declared in the entity are used for named mappings in the component instantiation.

Test Case Without Generic Map

```
library ieee;
use ieee.std_logic_1164.all;

entity HA is
    port (U,V: in bit;
          X,Y: out bit);
end HA;

architecture beh of Ha is
begin
    x<= u xor v;
    y<= u and v;
end beh;
```

```

entity test is
    port (A, B: in bit;
          SUM, CARRY: out bit);
end test;

architecture beh of test is
component HA
    port (U,V: in bit;
          X,Y: out bit);
end component;

begin
    u1: HA port map (A, B, SUM, CARRY);
end beh;

```

Test Case With Generic Map

```

library ieee;
use ieee.std_logic_1164.all;

entity HA is
generic (k: integer);
    port (U,V: in bit_vector(k-1 downto 0);
          X,Y: out bit_vector(k-1 downto 0));
end HA;

architecture beh of Ha is
begin
    x<= u xor v;
    y<= u and v;
end beh;

entity test is
    port (A, B: in bit_vector(2 downto 0);
          SUM, CARRY: out bit_vector(2 downto 0) );
end test;

architecture beh of test is
component HA
generic (k: integer);
    port (U,V: in bit_vector(k-1 downto 0);
          X,Y: out bit_vector(k-1 downto 0) );
end component;

begin
    u1: HA generic map (k=>3)
        port map (A, B, SUM, CARRY);
end beh;

```

CD174

@E: Duplicate specification of formal <b_in>

A formal was mapped more than once during port mapping. When named mapping is used in component instantiation, a formal can only be connected to a single actual. In the u1 component instantiation in the test case below, formal b_in is mapped to both b_in and clk which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity and1 is
    port (a_in, b_in: in std_logic_vector(3 downto 0);
          c_out: out std_logic_vector(3 downto 0) );
end and1;

architecture and1 of and1 is
begin
    c_out<=a_in and b_in;
end and1;

library ieee;
use ieee.std_logic_1164.all;
use work.and1.all;

entity test is
    port (clk: in std_logic;
          a_in_top, b_in_top: in std_logic_vector(3 downto 0);
          d_out_top: out std_logic_vector(3 downto 0) );
end test;

architecture beh of test is
component and1 is
    port (a_in, b_in: in std_logic_vector(3 downto 0);
          c_out: out std_logic_vector(3 downto 0) );
end component;

signal temp:std_logic_vector(3 downto 0);
begin
    u1: and1 port map
        (a_in=>a_in_top,b_in=>b_in_top,b_in=>clk,c_out=>temp);
    process (clk)
    begin
```

```

        if falling_edge(clk) then
            d_out <=temp;
        end if;
    end process;
end beh;
```

Action

Make sure that each formal is specified only once in the component instantiation as shown in the corrected test case below where formal `b_in` is specified only once.

```

library ieee;
use ieee.std_logic_1164.all;

entity and1 is
    port (a_in, b_in: in std_logic_vector(3 downto 0);
          c_out: out std_logic_vector(3 downto 0) );
end and1;

architecture and1 of and1 is
begin
    c_out<=a_in and b_in;
end and1;

library ieee;
use ieee.std_logic_1164.all;
use work.and1.all;

entity test is
    port (clk: in std_logic;
          a_in_top, b_in_top: in std_logic_vector(3 downto 0);
          d_out_top: out std_logic_vector(3 downto 0) );
end test;

architecture beh of test is
component and1 is
    port (a_in, b_in: in std_logic_vector(3 downto 0);
          c_out: out std_logic_vector(3 downto 0) );
end component;

signal temp:std_logic_vector(3 downto 0);
begin
u1: and1 port map(a_in=>a_in_top,b_in=>b_in_top,c_out=>temp);
process (clk)
begin
```

```
        if falling_edge(clk) then
            d_out_top <=temp;
        end if;
    end process;
end beh;
```

CD175

@E: Port map of component instantiation has more parameters than listed in the component declaration

During component instantiation, the number of parameters was greater than the number of parameters specified in the component declaration. When using positional mapping for instantiating a component, the number of parameters should always be the same as declared in the component declaration. If the number of parameters is less than the number declared, an actual is not mapped to a value, and if the number of parameters is more than the number declared, the compiler is unable to find an actual to associate with the additional parameter.

In the test case below, component instantiation u1 includes one parameter more than is specified in the and1 component declaration which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity and1 is
    port (a_in, b_in: in std_logic_vector(3 downto 0);
          c_out: out std_logic_vector(3 downto 0) );
end and1;

architecture test of and1 is
begin
    c_out<=a_in and b_in;
end test;

library ieee;
use ieee.std_logic_1164.all;
use work.and1.all;
```

```

entity test is
    port (clk: in std_logic;
          a_in_top, b_in_top: in std_logic_vector(3 downto 0);
          d_out_top: out std_logic_vector(3 downto 0) );
end test;

architecture beh of test is
component and1 is
    port (a_in, b_in: in std_logic_vector(3 downto 0);
          c_out: out std_logic_vector(3 downto 0) );
end component;

signal temp:std_logic;
begin
u1: and1 port map(a_in_top,b_in_top,temp,clk);
    process (clk)
    begin
        if falling_edge(clk) then
            d_out_top <=temp;
        end if;
    end process;
end beh;

```

Action

Make sure to specify the same number of parameters in the component instantiation as present in the component declaration. In the corrected test case below, the number of parameters in component instantiation u1 and component declaration and1 is the same.

```

library ieee;
use ieee.std_logic_1164.all;

entity and1 is
    port (a_in, b_in: in std_logic_vector(3 downto 0);
          c_out: out std_logic_vector(3 downto 0) );
end and1;

architecture test of and1 is
begin
    c_out<=a_in and b_in;
end test;

library ieee;
use ieee.std_logic_1164.all;
use work.and1.all;

```

```
entity test is
    port (clk: in std_logic;
          a_in_top, b_in_top: in std_logic_vector(3 downto 0);
          d_out_top: out std_logic_vector(3 downto 0) );
end test;

architecture beh of test is
component andl is
    port (a_in, b_in: in std_logic_vector(3 downto 0);
          c_out: out std_logic_vector(3 downto 0) );
end component;

signal temp:std_logic_vector( 3 downto 0);
begin
u1: andl port map(a_in_top,b_in_top,temp);
    process (clk)
    begin
        if falling_edge(clk) then
            d_out_top <=temp;
        end if;
    end process;
end beh;
```

CD176

@E: invalid function call for a port formal (only one parameter allowed)

Named mapping is being used in the component instantiation and a function call for a port formal with more than one argument was encountered. When named port mapping is used in a component instantiation, the formal name can be an original port name or a single-argument function on that port name. If the function is a multi parameter function, the result of this function is a new signal which is not available in the original entity. If the function is a single parameter function, the hardware equivalent to the function can be inserted before the actual port while developing the hierarchy.

In component instantiation u1 in the test case below, the compiler finds function and with two parameters for the formal corresponding to actual d_out which causes the error.

```

library ieee;
use ieee.std_logic_1164.all;
entity and1 is
    port (a_in, b_in: in std_ulogic_vector(3 downto 0);
          c_out: out std_ulogic_vector(3 downto 0) );
end and1;

architecture and1 of and1 is
begin
    c_out<=a_in and b_in;
end and1;

library ieee;
use ieee.std_logic_1164.all;
use work.and1.all;

entity test is
    port (a_in_top, b_in_top: in std_ulogic_vector(3 downto 0);
          d_out_top: out std_ulogic_vector(3 downto 0) );
end test;

architecture beh of test is
component and1 is
    port (a_in, b_in: in std_ulogic_vector(3 downto 0);
          c_out: out std_ulogic_vector(3 downto 0) );
end component;

begin

```

Action

Make sure to use a function of only one argument for a formal as shown in the corrected test case below. In the test case, the functional call is a not function of formal c_in and, as a single-parameter function, is valid.

```

library ieee;
use ieee.std_logic_1164.all;

entity and1 is
    port (a_in, b_in: in std_ulogic_vector(3 downto 0);
          c_out: out std_ulogic_vector(3 downto 0) );
end and1;
```

```

architecture and1 of and1 is
begin
    c_out<=a_in and b_in;
end and1;

library ieee;
use ieee.std_logic_1164.all;
use work.and1.all;

entity test is
    port (a_in_top, b_in_top: in std_ulogic_vector(3 downto 0);
          d_out_top: out std_ulogic_vector(3 downto 0) );
end test;

architecture beh of test is
component and1 is
    port (a_in, b_in: in std_ulogic_vector(3 downto 0);
          c_out: out std_ulogic_vector(3 downto 0) );
end component;

begin
u1: and1 port map(a_in=>a_in_top, b_in =>b_in_top,
                    not (c_out ) =>d_out_top);
end beh;

```

CD177

@E: invalid port formal

An invalid formal was encountered while instantiating a component using named association. While instantiating the component and1 in the test case below, the last formal specified as `not("1111")` is invalid which causes the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity and1 is
    port (a_in, b_in: in std_logic_vector(3 downto 0);
          c_out: out std_logic_vector(3 downto 0) );
end and1;

```

```

architecture and1 of and1 is
begin
    c_out<=a_in and b_in;
end and1;

library ieee;
use ieee.std_logic_1164.all;
use work.and1.all;

entity top is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0) );
end top;

architecture beh of top is
component and1 is
    port (a_in, b_in: in std_logic_vector(3 downto 0);
          c_out: out std_logic_vector(3 downto 0) );
end component;

signal temp:std_logic_vector(3 downto 0);
begin
u1: and1 port map(a_in=>a_in, b_in =>b_in, not ("1111")
=>temp);
    process (clk)
    begin
        if falling_edge(clk) then
            d_out <=temp;
        end if;
    end process;
end beh;

```

Action

Make sure that the formal is valid when instantiating a component. In the test case below, a valid formal is provided for the and1 component.

```

library ieee;
use ieee.std_logic_1164.all;

entity and1 is
    port (a_in, b_in: in std_logic_vector(3 downto 0);
          c_out: out std_logic_vector(3 downto 0) );
end and1;

```

```
architecture and1 of and1 is
begin
    c_out<=a_in and b_in;
end and1;

library ieee;
use ieee.std_logic_1164.all;
use work.and1.all;

entity top is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0) );
end top;

architecture beh of top is
component and1 is
    port (a_in, b_in: in std_logic_vector(3 downto 0);
          c_out: out std_logic_vector(3 downto 0) );
end component;

signal temp:std_logic_vector(3 downto 0);
begin
u1: and1 port map(a_in=>a_in, b_in =>b_in, c_out =>temp);
    process (clk)
    begin
        if falling_edge(clk) then
            d_out <=temp;
        end if;
    end process;
end beh;
```

CD178

@E: Can't find formal <c_in>

The formal specified in the component instantiation statement did not match the name of the ports in the entity declaration. In the test case below, `c_in` is specified as a formal in the component instantiation statement but, because there is no formal with the name `c_in` in the component declaration, the compiler errors out.

```
entity and1 is
    port (a_in, b_in: in bit;
          c_out: out bit );
end and1;

architecture test of and1 is
begin
    c_out<=a_in and b_in;
end test;

use work.and1.all;
entity seq is
    port (clk: in bit;
          a_in, b_in: in bit;
          d_out: out bit);
end seq;

architecture beh of seq is
component and1 is
    port (a_in, b_in: in bit;
          c_out: out bit );
end component;

signal temp:bit;
begin
u1:and1 port map(a_in=>a_in, b_in=>b_in, c_in =>temp);
    process (clk)
    begin
        if (clk'event and clk='1') then
            d_out <=temp;
        end if;
    end process;
end beh;
```

Action

Be sure to use the formals specified in the component declaration when instantiating a component as shown in the corrected test case below.

```
entity and1 is
    port (a_in, b_in: in bit;
          c_out: out bit );
end and1;
```

```
architecture test of and1 is
begin
    c_out<=a_in and b_in;
end test;

use work.and1.all;
entity seq is
    port (clk: in bit;
          a_in, b_in: in bit;
          d_out: out bit);
end seq;

architecture beh of seq is
component and1 is
    port (a_in, b_in: in bit;
          c_out: out bit );
end component;

signal temp:bit;
begin
u1:and1 port map(a_in=>a_in, b_in=>b_in, c_out =>temp);
    process (clk)
    begin
        if (clk'event and clk='1') then
            d_out <=temp;
        end if;
    end process;
end beh;
```


CHAPTER 7

CD Messages 179 – 265

CD179

@E: Formal port of this direction may not use conversion function (only directions 'out', 'inout', 'buffer' or 'linkage' allowed) <b_in>

A conversion function was applied on a formal port of direction input. In component instantiation, conversion functions on formals can only be applied on formals of direction out, inout, buffer, and linkage.

In the test case below, the function not is applied on formal port b_in which is of direction input.

```
library ieee;
use ieee.std_logic_1164.all;

entity and1 is
    port (a_in, b_in: in std_ulogic_vector(3 downto 0);
          c_out: out std_ulogic_vector(3 downto 0) );
end and1;

architecture and1 of and1 is
begin
    c_out<=a_in and b_in;
end and1;
```

```
library ieee;
use ieee.std_logic_1164.all;
use work.and1.all;

entity test is
    port (a_in_top, b_in_top: in std_ulogic_vector(3 downto 0);
          d_out_top: out std_ulogic_vector(3 downto 0) );
end test;

architecture beh of test is
component and1 is
    port (a_in, b_in: in std_ulogic_vector(3 downto 0);
          c_out: out std_ulogic_vector(3 downto 0) );
end component;

begin
u1: and1 port map(a_in=>a_in_top,not b_in=> b_in_top,
      c_out=>d_out_top);
end beh;
```

Action

Be sure to use the conversion functions only on formal ports of valid direction; the function can be applied on the corresponding actual as long as the functionality remains the same. This action is illustrated in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity and1 is
    port (a_in, b_in: in std_ulogic_vector(3 downto 0);
          c_out: out std_ulogic_vector(3 downto 0) );
end and1;

architecture and1 of and1 is
begin
    c_out<=a_in and b_in;
end and1;

library ieee;
use ieee.std_logic_1164.all;
use work.and1.all;
```

```

entity test is
    port (a_in_top, b_in_top: in std_ulogic_vector(3 downto 0);
          d_out_top: out std_ulogic_vector(3 downto 0) );
end test;

architecture beh of test is
component and1 is
    port (a_in, b_in: in std_ulogic_vector(3 downto 0);
          c_out: out std_ulogic_vector(3 downto 0) );
end component;

begin
u1: and1 port map(a_in=>a_in_top, b_in=> not b_in_top,
                    c_out=>d_out_top);
end beh;

```

CD181

@E: Actual of this direction may not use conversion function (only directions 'in', 'inout', and 'linkage' allowed)

A conversion function was applied on an actual of direction output or buffer. In component instantiation, conversion functions can be applied only on actuals of direction in, inout, and linkage.

In the test case below, the function not is applied on formal port d_out which is of direction output.

```

library ieee;
use ieee.std_logic_1164.all;

entity and1 is
    port (a_in, b_in: in std_ulogic_vector(3 downto 0);
          c_out: out std_ulogic_vector(3 downto 0) );
end and1;

architecture and1 of and1 is
begin
    c_out<=a_in and b_in;
end and1;

```

```

library ieee;
use ieee.std_logic_1164.all;
use work.and1.all;

entity test is
    port (a_in_top, b_in_top: in std_ulogic_vector(3 downto 0);
          d_out_top: out std_ulogic_vector(3 downto 0) );
end test;

architecture beh of test is
component and1 is
    port (a_in, b_in: in std_ulogic_vector(3 downto 0);
          c_out: out std_ulogic_vector(3 downto 0) );
end component;

begin
u1: and1 port map(a_in=>a_in_top,b_in=>b_in_top,
                     c_out=>not(d_out_top));
end beh;

```

Action

Be sure to use the conversion functions only on actual ports of valid direction; the function can be applied on the corresponding formal as long as the functionality remains the same. This action is illustrated in the corrected test case below.

```

library ieee;
use ieee.std_logic_1164.all;

entity and1 is
    port (a_in, b_in: in std_ulogic_vector(3 downto 0);
          c_out: out std_ulogic_vector(3 downto 0) );
end and1;

architecture and1 of and1 is
begin
    c_out<=a_in and b_in;
end and1;

library ieee;
use ieee.std_logic_1164.all;
use work.and1.all;

```

```

entity test is
    port (a_in_top, b_in_top: in std_ulogic_vector(3 downto 0);
          d_out_top: out std_ulogic_vector(3 downto 0) );
end test;

architecture beh of test is
component and1 is
    port (a_in, b_in: in std_ulogic_vector(3 downto 0);
          c_out: out std_ulogic_vector(3 downto 0) );
end component;

begin
u1: and1 port map(a_in=>a_in_top,b_in=>b_in_top,
                    not(c_out)=>(d_out_top));
end beh;

```

CD184

@E: target <c> of assignment is not writeable

The compiler encountered an assignment being made to an input port in the concurrent portion of an architecture block. The target of an assignment statement can be an output port, a signal, or a variable. In the test case below, input port c is being assigned a value that causes the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (a,b: in std_logic;
          c: in std_logic;
          dout : out std_logic);
end;

architecture rtl of comb is
begin
    dout<=a when c='1' else '0';
    c<=b;
end;

```

Action

Make sure that an output port, signal, or variable is used as the assignment target. The above test case can be corrected by removing the assignment statement as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (a: in std_logic;
          c: in std_logic;
          dout : out std_logic);
end;

architecture rtl of comb is
begin
    dout<=a when c='1' else '0';
end;
```

CD185

@E: Expecting readable signal name, got <1>

Non-readable signals were encountered in the sensitivity list. Some VHDL constructs such as process and wait require readable signals that should either be declared as ports inside the entity or as signals to act on.

In the following test case, the compiler encounters the string 1 in the sensitivity list of the process statement which is not declared as a signal or port which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (a : in std_logic_vector(2 downto 0);
          output : out std_logic_vector(2 downto 0) );
end test;
```

```

architecture beh of test is
begin
    process (1)
    begin
        case a is
            when "000" => output <= "101";
            when "001" => output <= "100";
            when "010" => output <= "000";
            when "011" => output <= "110";
            when "100" => output <= "001";
            when "101" => output <= "010";
            when "110" => output <= "011";
            when "111" => output <= "111";
        end case ;
    end process ;
end beh;

```

Action

Be sure to use a proper readable signal (i.e., an input port or a signal that has been assigned a value outside the process). This corrective action is illustrated in the following test case where the process statement references readable signal *a* in the sensitivity list.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (a : in std_logic_vector(2 downto 0);
          output : out std_logic_vector(2 downto 0) );
end test;

architecture beh of test is
begin
    process (a)
    begin
        case a is
            when "000" => output <= "101";
            when "001" => output <= "100";
            when "010" => output <= "000";
            when "011" => output <= "110";
            when "100" => output <= "001";
            when "101" => output <= "010";

```

```

        when "110" => output <= "011";
        when "111" => output <= "111";
    end case ;
end process ;
end beh;
```

CD187

@E: Expecting end if label <u2>

The label of an end if statement differs from the label of the if statement at the beginning of the loop. In an If loop, the label for the end if statement must match the label for the corresponding if statement at the beginning of the loop.

In the following test case, the end if statement of the nested If block (u2) ends with the label u1 which causes the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (q: out std_logic;
          rst: in std_logic;
          d : in std_logic;
          clk : in std_logic);
end test;

architecture test of test is
begin
    process(clk)
    begin
        u1:if(clk'event and clk='1') then
            u2:if(rst='1')then
                q<='0';
            else
                q<=d;
            end if u1;
        end if u1;
    end process;
end test;
```

Action

Make sure that the label at the start of an If statement matches the label of its corresponding end if statement. This corrective action is illustrated in the following test case where the end if statement label of the nested If block has been changed to match the u2 label of the if statement at the beginning of the block.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (q: out std_logic;
          rst: in std_logic;
          d : in std_logic;
          clk : in std_logic);
end test;

architecture test of test is
begin
    process(clk)
    begin
        u1:if(clk'event and clk='1') then
            u2:if(rst='1')then
                q<='0';
            else
                q<=d;
            end if u2;
        end if u1;
    end process;
end test;

```

CD191

@E: Expecting | to separate choices or => to end choices

The compiler failed to find a | or => between a *choice* and its *sequential_statement*. The format of a case statement is:

```
case expression is
  when choices => sequential_statements      -- form 1
  when choice1|choice2 => sequential_statements  --form 2
  [when others => sequential_statements]
end case;
```

In the test case below, there is no | or => between the choice 000 and the sequential statement output <= "101" which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
  port (addr : in std_logic_vector(2 downto 0);
        output : out std_logic_vector(2 downto 0));
end test;

architecture beh of test is
begin
  process(addr)
  begin
    case addr is
      when "000"  output <= "101";
      when "001" => output <= "101";
      when "010" => output <= "000";
      when "011" => output <= "110";
      when "100" => output <= "001";
      when "101" => output <= "010";
      when "110" => output <= "011";
      when "111" => output <= "111";
    end case ;
  end process ;
end beh;
```

Action

Make sure that a => is specified after a *choice* if it is a single choice or that a | is specified if there are two or more choices. This corrective action is illustrated in the test case below.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity test is
    port (addr : in std_logic_vector(2 downto 0);
          output : out std_logic_vector(2 downto 0));
end test;

architecture beh of test is
begin
    process(addr)
    begin
        case addr is
            when "000" => output <= "101";
            when "001" => output <= "101";
            when "010" => output <= "000";
            when "011" => output <= "110";
            when "100" => output <= "001";
            when "101" => output <= "010";
            when "110" => output <= "011";
            when "111" => output <= "111";
        end case ;
    end process ;
end beh;
```

CD192

@E: target of assignment is not writeable

The compiler encountered an assignment being made to an input port within a process block (sequential). The target of an assignment statement can only be an output port, a signal, or a variable. In the test case below, input port c is being assigned a value which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (a,b: in std_logic;
          c: in std_logic;
          dout : out std_logic);
end;
```

```
architecture rtl of seq is
begin
    process(a,c)
    begin
        if( c='1') then
            dout<= a ;
        else
            dout<='0';
        end if;
c<=b;
    end process;
end;
```

Action

Make sure that an output port, signal, or variable is used as an assignment target. The above test case can be corrected by removing the assignment statement as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (a,b: in std_logic;
          c: in std_logic;
          dout : out std_logic);
end;

architecture rtl of seq is
begin
    process(a,c)
    begin
        if( c='1') then
            dout<= a ;
        else
            dout<='0';
        end if;
    end process;
end;
```

CD193

@E: Expecting aggregate assignment

An aggregate target was not assigned a value. The target of a variable assignment statement or signal assignment statement can be an aggregate. An aggregate target represents a combination of one or more names. The compiler recognizes an aggregate target by the enclosing parentheses.

In the following test case, the compiler identifies the aggregate target (va, vc), but finds no target assignment which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (a: in bit_vector(0 to 1);
          ra, rb: out bit );
end test;

architecture muxes of test is
begin
    process (a)
    variable va,vc: bit;
    begin
        (va,vc);
        ra <= va;
        rb <= vc;
    end process;
end muxes;
```

Action

Make sure to specify a proper aggregate assignment (:= a) as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (a: in bit_vector(0 to 1);
          ra, rb: out bit );
end test;
```

```

architecture muxes of test is
begin
    process (a)
        variable va, vc: bit;
    begin
        (va,vc):= a;
        ra <= va;
        rb <= vc;
    end process;
end muxes;

```

CD194

@E: Expecting loop label <u1>

The beginning and ending loop labels do not match. The syntax of a simple loop statement is

```

[loopLabel:] loop
    sequential statements
end loop [loopLabel];

```

In the test case below, *loopLabel* at the start of the loop block is u1 and *loopLabel* at the end of the loop block is u2 which causes the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (addr : in std_logic_vector(2 downto 0);
          output : out std_logic_vector(2 downto 0));
end test;

architecture beh of test is
begin
    process(addr)
        variable i:integer;
        variable temp: std_logic;
    begin
        i:=2;
        temp:='1';
        output(2)<=addr(0);
        output(0)<=addr(2);

```

```
u1:loop
    temp:=temp and addr(i);
    i:=i-1;
    exit when i=0;
end loop u2;
    output(1)<=temp;
end process;
end beh;
```

Action

Be sure to specify the same *loop_label* at both the start and end of the loop block. This corrective action is illustrated in the test case below where the loop label at the start and end of the loop block is u1.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (addr : in std_logic_vector(2 downto 0);
          output : out std_logic_vector(2 downto 0));
end test;

architecture beh of test is
begin
    process(addr)
        variable i:integer;
        variable temp: std_logic;
    begin
        i:=2;
        temp:='1';
        output(2)<=addr(0);
        output(0)<=addr(2);
        u1:loop
            temp:=temp and addr(i);
            i:=i-1;
            exit when i=0;
        end loop u1;
        output(1)<=temp;
    end process;
end beh;
```

CD195

@E: Expected iteration variable name

A for statement does not include an iteration variable name (*identifier*). The syntax of a simple for loop statement is:

```
[loopLabel]: for identifier in range loop
    sequential statements
end loop [loopLabel];
```

In the test case below, the iteration variable is omitted from the for statement which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (addr : in std_logic_vector(2 downto 0);
          clk:in std_logic;
          output : out std_logic_vector(2 downto 0));
end test;

architecture beh of test is
begin
    process(clk)
    begin
        if(clk'event and clk='1')then
            for in 0 to 2 loop
                output(i)<=addr(i);
            end loop ;
        end if;
    end process;
end beh;
```

Action

This syntax error can be eliminated by specifying the iteration variable in the for statement as shown in the corrected test case below (iteration variable i specified).

```
library ieee;
use ieee.std_logic_1164.all;
```

```

entity test is
    port (addr : in std_logic_vector(2 downto 0);
          clk:in std_logic;
          output : out std_logic_vector(2 downto 0));
end test;

architecture beh of test is
begin
    process(clk)
    begin
        if(clk'event and clk='1')then
            for i in 0 to 2 loop
                output(i)<=addr(i);
            end loop ;
        end if;
    end process;
end beh;

```

CD196

@E: Expecting for loop label <u1>

The loop label at the start of a for loop did not match the loop label in the corresponding end loop statement. The syntax of a simple for loop statement is:

```

[loopLabel:] for identifier in range loop
    sequential statements
end loop [loopLabel];

```

In the test case below, the loop label at the start of the for loop block is u1 and the loop label at the end of the for loop block is u2 which results in the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (addr : in std_logic_vector(2 downto 0);
          clk:in std_logic;
          output : out std_logic_vector(2 downto 0) );
end test;

```

```
architecture beh of test is
begin
    process(clk)
    begin
        if(clk'event and clk='1')then
            u1:for i in 0 to 2 loop
                output(i)<=addr(i);
            end loop u2;
        end if;
    end process;
end beh;
```

Action

Make sure that the same *loop_label* is specified at both the start and end of the for loop block. This corrective action is illustrated in the test case below which shows the *loop_label* at both the start and end of the for block set to u1.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (addr : in std_logic_vector(2 downto 0);
          clk:in std_logic;
          output : out std_logic_vector(2 downto 0));
end test;

architecture beh of test is
begin
    process(clk)
    begin
        if(clk'event and clk='1')then
            u1:for i in 0 to 2 loop
                output(i)<=addr(i);
            end loop u1;
        end if;
    end process;
end beh;
```

CD197

@E: Can't assign to constant

An attempt was made to assign a value to a constant after the initialization of that constant. In VHDL, an object of class constant can hold a single value of a given type. This value is assigned to the constant prior to the start of simulation, and the value cannot be changed during the course of simulation. In the test case below, the attempt to increment the constant is not valid which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (addr : in std_logic_vector(2 downto 0);
          clk:in std_logic;
          output : out std_logic_vector(2 downto 0) );
end test;

architecture beh of test is
constant j: integer:=0;
begin
    process(clk)
    begin
        if(clk'event and clk='1')then
            for i in 0 to 2 loop
                output(i)<=addr(i+j);
                j:=j+1;
            end loop ;
        end if;
    end process;
end beh;
```

Action

Be careful not to assign values to constants elsewhere in the process or to declare identifiers as variables or signals. In the corrected test case below, constant j is not assigned a value after it is initialized.

```
library ieee;
use ieee.std_logic_1164.all;
```

```

entity test is
    port (addr : in std_logic_vector(2 downto 0);
          clk:in std_logic;
          output : out std_logic_vector(2 downto 0) );
end test;

architecture beh of test is
constant j: integer:=0;
begin
    process(clk)
    begin
        if(clk'event and clk='1')then
            for i in 0 to 2 loop
                output(i)<=addr(i+j);
            end loop ;
        end if;
    end process;
end beh;

```

CD199

@E: elseif should probably be elsif

An `elseif` string appeared as `elseif` in an If block. In the test case below, the compiler encounters the string “`elseif`” which causes the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (q: out std_logic_vector(7 downto 0);
          rst: in std_logic;
          d : in std_logic_vector(7 downto 0);
          clk : in std_logic);
end test;

architecture test of test is
begin
    process(clk,rst)
    begin
        if(rst='1')then
            q<="00000000";

```

```
        elsif (clk'event and clk='1') then
            q<=d;
        end if ;
    end process;
end test;
```

Action

Change the `elseif` string to an `elsif` string as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (q: out std_logic_vector(7 downto 0);
          rst: in std_logic;
          d : in std_logic_vector(7 downto 0);
          clk : in std_logic);
end test;

architecture test of test is
begin
    process(clk,rst)
    begin
        if(rst='1')then
            q<="00000000";
        elsif(clk'event and clk='1') then
            q<=d;
        end if ;
    end process;
end test;
```

@E: Misspelled variable, signal or procedure name?

An undeclared signal or variable or an undefined procedure was used within a process block. In the test case below, undeclared variable `p` is used within the process which causes the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (a,b: in std_logic_vector(1 downto 0);
          q:out std_logic_vector(1 downto 0));
end comb;

architecture test of comb is
begin
    process(a,b)
    begin
        for i in 1 downto 0 loop
            p(i):= a(i) and b(i);
        end loop;
        q<=p;
    end process;
end test;

```

Action

Be sure to declare the target variable, signal, or procedure before it is used. In the test case below, declaring variable *p* in the architecture declaration before it is referenced corrects the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (a,b: in std_logic_vector(1 downto 0);
          q:out std_logic_vector(1 downto 0));
end comb;

architecture test of comb is
begin
    process(a,b)
        variable p: std_logic_vector(1 downto 0);
    begin
        for i in 1 downto 0 loop
            p(i):= a(i) and b(i);
        end loop;
        q<=p;
    end process;
end test;

```

CD201

@E: Can't find loop label </2>

A loop with the specified *loopLabel* cannot be found within the `next` or `exit` statement. The `next` and `exit` sequential statements can be used within a loop to control execution flow. The general syntax is:

```
next|exit [loopLabel] [when condition];
```

In the test case below, the `next` statement includes *loopLabel* L2, but because there is no loop with *loop_label* L2, the compiler errors out.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (a:in std_logic_vector( 9 downto 0);
          clk: in std_logic;
          b:in std_logic_vector(7 downto 0);
          op: out std_logic);
end test;

architecture beh of test is
begin
    process(a,b,clk)
        variable temp:std_logic;
    begin
        temp:='1';
        L1:for i in 7 downto 0 loop
            for j in 9 downto 0 loop
                if(clk='1') then
                    temp:= temp and a(j);
                else
next L2;
                end if;
            end loop;
        temp:= temp and b(i);
        end loop;
        op<=temp;
    end process;
end beh;
```

Action

Be sure to specify a valid *loop_label* in the corresponding next or exit statement. This corrective action is illustrated in the test case below which shows the *loop_label* of the for loop enclosing the next statement set to L1.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (a:in std_logic_vector( 9 downto 0 );
          clk: in std_logic;
          b:in std_logic_vector(7 downto 0);
          op: out std_logic);
end test;

architecture beh of test is
begin
    process(a,b,clk)
        variable temp:std_logic;
    begin
        temp:='1';
        L1:for i in 7 downto 0 loop
            for j in 9 downto 0 loop
                if(clk='1') then
                    temp:= temp and a(j);
                else
                    next L1;
                end if;
            end loop;
            temp:= temp and b(i);
        end loop;
        op<=temp;
    end process;
end beh;
```

CD202

@E: No enclosing loop

A next or exit statement was found outside a loop. The next and exit sequential statements can be used only within a loop to control execution flow.

In the test case below, the `exit` statement is used as a sequential statement inside a process, but because it has no enclosing loop, the compiler errors out.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (addr : in std_logic_vector(2 downto 0);
          clk:in std_logic;
          output : out std_logic_vector(2 downto 0));
end test;

architecture beh of test is
begin
    process(clk)
        variable i:integer:=0;
        variable temp:std_logic;
    begin
        L1:for i in 0 to 5 loop
            exit L1 when i > 2;
            if(clk'event and clk='1') then
                output(i)<=addr(i);
            end if;
        end loop L1;
        end process;
    end beh;
```

Action

Make sure that there is no `next` or `exit` statement outside of a loop. In the test case below, the `exit` statement is simply omitted.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (addr : in std_logic_vector(2 downto 0);
          clk:in std_logic;
          output : out std_logic_vector(2 downto 0));
end test;

architecture beh of test is
begin
    process(clk)
        variable i:integer;
        variable temp:std_logic;
```

```

begin
  if(clk'event and clk='1') then
    output(2)<=addr(0);
    output(0)<=addr(2);
    output(1)<=addr(1);
  end if;
end process;
end beh;

```

CD203

@E: return statement must be used inside a procedure or function

A return statement was encountered outside a procedure or function. In VHDL, a return statement is used only inside a procedure or function to return a value to the subprogram-call.

In the test case below, the return statement occurs within a for loop which causes the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
  port (a:in std_logic_vector( 9 downto 0 );
        clk: in std_logic;
        b:in std_logic_vector(7 downto 0 );
        op: out std_logic);
end test;

architecture beh of test is
begin
  process(a,b,clk)
  variable temp:std_logic;
  begin
    temp:='1';
    L1:for i in 7 downto 0 loop
      for j in 9 downto 0 loop
        if(clk='1') then
          temp:= temp and a(j);
        else
          return L1;
        end if;
      end loop;
    end loop;
  end process;
end beh;

```

```
        end loop;
        temp:= temp and b(i);
    end loop;
    op<=temp;
end process;
end beh;
```

Action

Make sure that there is no return statement outside of a procedure or function. In the test case below, the reference to the invalid return statement is replaced with a next statement.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (a:in std_logic_vector( 9 downto 0 );
          clk: in std_logic;
          b:in std_logic_vector(7 downto 0 );
          op: out std_logic);
end test;

architecture beh of test is
begin
    process(a,b,clk)
    variable temp:std_logic;
    begin
        temp:='1';
        L1:for i in 7 downto 0 loop
            for j in 9 downto 0 loop
                if(clk='1') then
                    temp:= temp and a(j);
                else
                    next L1;
                end if;
            end loop;
            temp:= temp and b(i);
        end loop;
        op<=temp;
    end process;
end beh;
```

CD204

@E: Expecting sequential statement

A concurrent statement was found within a process, If or loop block, or other VHDL construct that includes only a sequential statement. In the test case below, the compiler finds a component instantiation statement (which is a concurrent statement) within an If block which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity not1 is
    port (a:in bit;
          b:out bit);
end not1;

architecture tst of not1 is
begin
    b<= not a;
end tst;

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (din : in bit;
          rst: in bit;
          clk: in bit;
          q: out bit);
end test;

architecture beh of test is
signal temp: bit;
begin
process(clk,rst,temp)
begin
    if(rst='1')then
        ul: entity work.not1(tst) port map('1',temp);
        q<=temp;
    else if (clk'event and clk='1') then
        q<=din;
    end if;
    end if;
end process;
end beh;
```

Action

Make sure that there are no concurrent statements appearing within process blocks. In the test case below, the concurrent statement is moved to the concurrent portion of the architecture block.

```
library ieee;
use ieee.std_logic_1164.all;

entity not1 is
    port (a:in bit;
          b:out bit);
end not1;

architecture tst of not1 is
begin
    b<= not a;
end tst;

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (din : in bit;
          rst: in bit;
          clk: in bit;
          q: out bit);
end test;

architecture beh of test is
signal temp: bit;
begin
u1: entity work.not1(tst) port map('1',temp);
process(clk,rst,temp)
begin
    if(rst='1')then
        q<=temp;
    else if (clk'event and clk='1') then
        q<=din;
    end if;
    end if;
end process;
end beh;
```

CD205

@E: Expecting statement label <u1>

The labels at the start and end of a `case` statement did not match. In the test case below, the label at the start of the `case` statement is `u1` while the label at the end of the `case` statement is `u2` which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (addr : in std_logic_vector(2 downto 0);
          output : out std_logic_vector(2 downto 0));
end comb;

architecture beh of comb is
begin
    process(addr)
    begin
        u1: case addr is
            when "000" => output <= "101";
            when "001" => output <= "100";
            when "010" => output <= "000";
            when "011" => output <= "110";
            when "100" => output <= "001";
            when "101" => output <= "010";
            when "110" => output <= "011";
            when "111" => output <= "111";
        end case u2;
    end process;
end beh;
```

Action

Be sure to provide the same label at the start and end of the `case` statement as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (addr : in std_logic_vector(2 downto 0);
          output : out std_logic_vector(2 downto 0));
end comb;
```

```

architecture beh of comb is
begin
    process(addr)
    begin
        u1: case addr is
            when "000" => output <= "101";
            when "001" => output <= "100";
            when "010" => output <= "000";
            when "011" => output <= "110";
            when "100" => output <= "001";
            when "101" => output <= "010";
            when "110" => output <= "011";
            when "111" => output <= "111";
        end case u1;
    end process;
end beh;

```

CD207

@E: Expecting sensitivity list or declaration

A process statement did not have a sensitivity list and no process-item declarations were found. In the test case below, the compiler finds a process statement with no sensitivity list and, in the ensuing line, finds a string (**begin1**) instead of a process-item declaration which causes the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (addr : in std_logic_vector(2 downto 0);
          output : out std_logic_vector(2 downto 0));
end test;

architecture beh of test is
begin
    process
begin1
        case addr is
            when "000" => output <= "101";
            when "001" => output <= "100";
            when "010" => output <= "000";
            when "011" => output <= "110";

```

```
        when "100" => output <= "001";
        when "101" => output <= "010";
        when "110" => output <= "011";
        when "111" => output <= "111";
    end case ;
    wait on (addr);
end process ;
end beh;
```

Action

Be sure to specify a sensitivity list for the process statement or to declare process-item statements following the process statement. In the corrected test case below, the process statement is immediately followed by a begin process-item declaration.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (addr : in std_logic_vector(2 downto 0);
          output : out std_logic_vector(2 downto 0));
end test;

architecture beh of test is
begin
    process
    begin
        case addr is
            when "000" => output <= "101";
            when "001" => output <= "100";
            when "010" => output <= "000";
            when "011" => output <= "110";
            when "100" => output <= "001";
            when "101" => output <= "010";
            when "110" => output <= "011";
            when "111" => output <= "111";
        end case ;
        wait on (addr);
    end process ;
end beh;
```

CD208

@E: Expecting process label <u1>

The process labels at the start and end of a process block differ. The syntax of a process statement is:

```
[processLabel:] process [(sensitivityList)] [is]
    [processItemDeclaration]
begin
    sequential statements;
end process [processLabel];
```

In the following test case, the label at the start of the process block is u1 and the label at the end of the process block is u2 which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (addr : in std_logic_vector(2 downto 0);
          output : out std_logic_vector(2 downto 0));
end test;

architecture beh of test is
begin
    u1:process(addr)
    begin
        case addr is
            when "000" => output <= "101";
            when "001" => output <= "100";
            when "010" => output <= "000";
            when "011" => output <= "110";
            when "100" => output <= "001";
            when "101" => output <= "010";
            when "110" => output <= "011";
            when "111" => output <= "111";
        end case ;
    end process u2;
end beh;
```

Action

Be sure to specify the same label at the start and end of a process block as shown in the corrected test case below where the process label at the end of the process block has been changed from u2 to u1.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (addr : in std_logic_vector(2 downto 0);
          output : out std_logic_vector(2 downto 0));
end test;

architecture beh of test is
begin
    u1:process(addr)
    begin
        case addr is
            when "000" => output <= "101";
            when "001" => output <= "100";
            when "010" => output <= "000";
            when "011" => output <= "110";
            when "100" => output <= "001";
            when "101" => output <= "010";
            when "110" => output <= "011";
            when "111" => output <= "111";
        end case ;
    end process u1;
end beh;
```

CD209

@E: Expecting block label <g2>

The block label at the start and at the end of a block statement are not the same. The syntax for a block statement is:

```

blockLabel : block [(guardExpression)] [is]
    [blockHeader]
    [blockDeclarations]
begin
    concurrent_statements
end block [blockLabel];

```

In the following test case, the block label at the start of the block is g2 and the block label at the end of the block is f2 which causes the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity fa is
    port (a,b,cin: std_logic;
          s,cout:out std_logic);
end fa;

architecture str of fa is
begin
    s<= a xor b xor cin;
    cout<= (a and b) or( a and cin) or (b and cin);
end str;

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (a,b:in std_logic_vector( 3 downto 0 );
          cin: std_logic;
          s: out std_logic_vector(3 downto 0 );
          cout:out std_logic);
end test;

architecture beh of test is
component fa is
    port (a,b,cin: std_logic;
          s,cout:out std_logic);
end component;

signal c_in: std_logic_vector(3 downto 0 );
begin
g2:block
    constant i: integer:=0;
    begin
        fa: fa port map (a(i),b(i),cin,s(i),c_in(i));
    end block f2;

```

```
g1:for k in 1 to 3 generate
    fa port map (a(k),b(k),c_in(k-1),s(k),c_in(k));
end generate;
cout<=c_in(3);
end beh;
```

Action

Be sure to specify the same block label at the start and end of a block as shown in the corrected test case below where the block label at the end of the process block has been changed from f2 to g2.

```
library ieee;
use ieee.std_logic_1164.all;

entity fa is
    port (a,b,cin: std_logic;
          s,cout:out std_logic);
end fa;

architecture str of fa is
begin
    s<= a xor b xor cin;
    cout<= (a and b) or( a and cin) or (b and cin);
end str;

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (a,b:in std_logic_vector( 3 downto 0);
          cin: std_logic;
          s: out std_logic_vector(3 downto 0);
          cout:out std_logic);
end test;

architecture beh of test is
component fa is
    port (a,b,cin: std_logic;
          s,cout:out std_logic);
end component;
```

```

signal c_in: std_logic_vector(3 downto 0);
begin
g2:block
    constant i: integer:=0;
    begin
        fa: fa port map (a(i),b(i),cin,s(i),c_in(i));
    end block g2;

g1:for k in 1 to 3 generate
    fa: fa port map (a(k),b(k),c_in(k-1),s(k),c_in(k));
end generate;
cout<=c_in(3);
end beh;

```

CD210

@E: Expecting generate label <g2>

The generate labels at the start and the end of a generate block are not the same. The format of the generate statement using the for-generate scheme is:

```

generateLabel: for generateIdentifier in discreteRange generate
    [block_declarations]
    [begin]
        concurrent_statements
    end generate [generateLabel];

```

In the following test case, the generate label at the start of the inner generate block is g2 and the generate label at the end of that generate block is g4 which causes the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity fa is
    port (a,b,cin: std_logic;
          s,cout:out std_logic);
end fa;

```

```

architecture str of fa is
begin
    s<= a xor b xor cin;
    cout<= (a and b) or( a and cin) or (b and cin);
end str;

library ieee;
use ieee.std_logic_1164.all;
entity test is
    port (a,b:in std_logic_vector( 3 downto 0);
          cin: std_logic;
          s: out std_logic_vector(3 downto 0);
          cout:out std_logic);
end test;

architecture beh of test is
component fa is
    port (a,b,cin: std_logic;
          s,cout:out std_logic);
end component;

signal c_in: std_logic_vector(3 downto 0);
begin
g1:for k in 0 to 3 generate
    g2:if k=0 generate
        fa: fa port map (a(k),b(k),cin,s(k),c_in(k));
    end generate g4;
    g3:if k>0 generate
        fa: fa port map (a(k),b(k),c_in(k-1),s(k),c_in(k));
    end generate;
end generate;
cout<=c_in(3);

end beh;

```

Action

Make sure that the same generate labels are specified at the start and end of a generate block as shown in the corrected test case below where the generate label at the end of the generate block has been changed from g4 to g2.

```

library ieee;
use ieee.std_logic_1164.all;

```

```
entity fa is
    port (a,b,cin: std_logic;
          s,cout:out std_logic);
end fa;

architecture str of fa is
begin
    s<= a xor b xor cin;
    cout<= (a and b) or( a and cin) or (b and cin);
end str;

library ieee;
use ieee.std_logic_1164.all;
entity test is
    port (a,b:in std_logic_vector( 3 downto 0);
          cin: std_logic;
          s: out std_logic_vector(3 downto 0);
          cout:out std_logic);
end test;

architecture beh of test is
component fa is
    port (a,b,cin: std_logic;
          s,cout:out std_logic);
end component;

signal c_in: std_logic_vector(3 downto 0);
begin
g1:for k in 0 to 3 generate
    g2:if k=0 generate
        fa: fa port map (a(k),b(k),cin,s(k),c_in(k));
    end generate g2;
    g3:if k>0 generate
        fa: fa port map (a(k),b(k),c_in(k-1),s(k),c_in(k));
    end generate;
end generate;
    cout<=c_in(3);

end beh;
```

CD211

@E: component instantiation needs label

A component instantiation statement does not have a component label. The format of a component instantiation statement is:

componentLabel: componentName [port map (associationList)];

In the test case below, the component instantiation statement for component and1 is missing its component label which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity and1 is
    port (a_in, b_in: in std_logic_vector(3 downto 0);
          c_out: out std_logic_vector(3 downto 0) );
end and1;

architecture test of and1 is
begin
    c_out<=a_in and b_in;
end test;

library ieee;
use ieee.std_logic_1164.all;
use work.and1.all;

entity test is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0) );
end test;

architecture beh of test is
component and1 is
    port (a_in, b_in: in std_logic_vector(3 downto 0);
          c_out: out std_logic_vector(3 downto 0) );
end component;

signal temp:std_logic_vector(3 downto 0);
begin
and1 port map(a_in,b_in,temp);
    process (clk)
    begin
```

```
        if falling_edge(clk) then
            d_out <=temp;
        end if;
    end process;
end beh;
```

Action

Be sure to specify a component label at the start of a component instantiation statement. As shown in the test case below, adding component label u1 to the and1 component instantiation statement corrects the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity and1 is
    port (a_in, b_in: in std_logic_vector(3 downto 0);
          c_out: out std_logic_vector(3 downto 0) );
end and1;

architecture test of and1 is
begin
    c_out<=a_in and b_in;
end test;

library ieee;
use ieee.std_logic_1164.all;
use work.and1.all;

entity test is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0) );
end test;

architecture beh of test is
component and1 is
    port (a_in, b_in: in std_logic_vector(3 downto 0);
          c_out: out std_logic_vector(3 downto 0) );
end component;

signal temp:std_logic_vector(3 downto 0);
begin
u1:and1 port map(a_in,b_in,temp);
    process (clk)
    begin
```

```

        if falling_edge(clk) then
            d_out <=temp;
        end if;
    end process;
end beh;
```

CD212

@E: entity instantiation needs label

An entity instantiation statement did not have a component label. The syntax for an entity instantiation statement is:

```

componentLabel: entity entityName [(architectureName)]
    [generic map (genericAssociationList)]
    [port map (associationList)];
```

In the test case below, the entity instantiation statement for entity ha is missing a component label which causes the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity ha is
    port (U,V: in bit;
          X,Y: out bit);
end ha;

architecture GATE of ha is
begin
    x<= u xor v;
    y<= u and v;
end;

entity test is
    port (A, B: in bit;
          SUM, CARRY: out bit);
end test;

architecture STRUCT of test is
begin
    entity work.ha(GATE) port map (A, B, SUM,CARRY);
end STRUCT;
```

Action

Be sure to specify a component label at the start of an entity-instantiation statement. As shown in the test case below, adding component label u1 at the start of the entity-instantiation statement for entity ha corrects the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity ha is
    port (U,V: in bit;
          X,Y: out bit);
end ha;

architecture GATE of ha is
begin
    x<= u xor v;
    y<= u and v;
end;

entity test is
    port (A, B: in bit;
          SUM, CARRY: out bit);
end test;

architecture STRUCT of test is
begin
    u1:entity work.ha(GATE) port map (A, B, SUM,CARRY);
end STRUCT;
```

CD213

@E: Undefined identifier

An identifier that was not declared as a signal or a port has been assigned a value in the concurrent part of an architectural block. In the test case below, target d1 is not declared as a port or as a signal which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity comb is
    port (a1,b1,c1 : in std_logic;
          out1 : out std_logic);
end entity;

architecture rtl of comb is
signal reg: std_logic;
begin
    reg<=a1;
    d1<=b1 and reg and c1;
end;
```

Action

Be sure to declare each identifier as either a port or a signal before using it as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (a1,b1,c1 : in std_logic;
          out1 : out std_logic);
end entity;

architecture rtl of comb is
signal reg: std_logic;
begin
    reg<=a1;
    out1<=b1 and reg and c1;
end;
```

CD214

@E: Not a concurrent statement

While running a syntax check, the compiler found a statement that was not fully defined as a concurrent statement. This type of syntax error can occur in process statements, instantiations, selected assignments, conditional assignments, and in for-generate statements. For example, this error occurs

when the code does not include a process statement before a begin statement. The following code segment generates this error because the component name is not specified:

```
u1 : port map (port-list); -- Component name missing
```

Action

Make sure your code is fully defined. To correct the error in the above code segment, specify the component name:

```
u1: test port map (port-list);
```

CD215

@E: label required for block

A block statement does not have a block label. The syntax for a block statement is:

```
blockLabel : block [(guardExpression)] [is]
  [blockHeader]
  [blockDeclarations]
  begin
    concurrent_statements
  end block [blockLabel];
```

In the test case below, the block statement is missing its block label which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity fa is
  port (a,b,cin: std_logic;
        s,cout:out std_logic);
end fa;
```

```

architecture str of fa is
begin
    s<= a xor b xor cin;
    cout<= (a and b) or( a and cin) or (b and cin);
end str;

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (a,b:in std_logic_vector( 3 downto 0);
          cin: std_logic;
          s: out std_logic_vector(3 downto 0);
          cout:out std_logic);
end test;

architecture beh of test is
component fa is
    port (a,b,cin: std_logic;
          s,cout:out std_logic);
end component;

signal c_in: std_logic_vector(3 downto 0);
begin
    block
        constant i: integer:=0;
        begin
            fal: fa port map (a(i),b(i),cin,s(i),c_in(i));
        end block ;
    g1:for k in 1 to 3 generate
        fa2: fa port map (a(k),b(k),c_in(k-1),s(k),c_in(k));
    end generate;

    cout<=c_in(3);
end beh;

```

Action

Be sure that all blocks include block labels. As shown in the corrected test case below, block label fablk has been added to the block statement.

```

library ieee;
use ieee.std_logic_1164.all;

```

```
entity fa is
    port (a,b,cin: std_logic;
          s,cout:out std_logic);
end fa;

architecture str of fa is
begin
    s<= a xor b xor cin;
    cout<= (a and b) or( a and cin) or (b and cin);
end str;

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (a,b:in std_logic_vector( 3 downto 0);
          cin: std_logic;
          s: out std_logic_vector(3 downto 0);
          cout:out std_logic);
end test;

architecture beh of test is
component fa is
    port (a,b,cin: std_logic;
          s,cout:out std_logic);
end component;

signal c_in: std_logic_vector(3 downto 0);
begin
    fablk:block
        constant i: integer:=0;
        begin
            fa1: fa port map (a(i),b(i),cin,s(i),c_in(i));
        end block ;
    g1:for k in 1 to 3 generate
        fa2: fa port map (a(k),b(k),c_in(k-1),s(k),c_in(k));
    end generate;

    cout<=c_in(3);
end beh;
```

CD216

@E: label required for generate

A generate label was missing at the start of a generate block statement. In the test case below, no label is specified for the generate statement which causes the error.

```
entity xor_n is
    port (a,b: in bit;
          c: out bit);
end;

architecture rtl of xor_n is
begin
    c<= a xor b;
end;

entity comb is
    port (a,b: in bit_vector(13 downto 0);
          c: out bit_vector(13 downto 0));
end;

architecture rtl of comb is
component xor_n is
    port (a,b: in bit;
          c: out bit);
end component;

begin
    for i in 0 to 13 generate
        u1: xor_n port map(a(i),b(i),c(i));
    end generate;
end;
```

Action

Be sure to specify the generate label at the start of the generate block as shown in the corrected test case below.

```
entity xor_n is
    port (a,b: in bit;
          c: out bit);
end;
```

```

architecture rtl of xor_n is
begin
    c<= a xor b;
end;

entity comb is
    port (a,b: in bit_vector(13 downto 0);
          c: out bit_vector(13 downto 0));
end;

architecture rtl of comb is
component xor_n is
    port (a,b: in bit;
          c: out bit);
end component;

begin
    s1:for i in 0 to 13 generate
        u1: xor_n port map(a(i),b(i),c(i));
    end generate;
end;

```

CD217

@W: Built in mapping used - please avoid!!!

A function is mapped to a builtin. Builtins are the predefined functions and types defined by the language. Generally, builtin mapping is used when defining the language packages. In the test case below, the function my_and is mapped to the builtin function and which causes the warning.

Note: This warning can be safely ignored when IEEE packages are redefined.

```

library ieee;
use ieee.std_logic_1164.all;

package my_pack is
    function my_and (a,b: std_logic_vector)
        return std_logic_vector =>"and";
    end my_pack;

```

```

library ieee;
use ieee.std_logic_1164.all;
use work.my_pack.all;

entity seq is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0) );
end seq;

architecture beh of seq is
begin
    process (clk)
    begin
        if falling_edge(clk) then
            d_out <= my_and(a_in, b_in);
        end if;
    end process;
end beh;

```

Action

Make sure that any builtin functions are not mapped and be sure to define the function explicitly as shown in the corrected test case below.

```

library ieee;
use ieee.std_logic_1164.all;

package my_pack is
    function my_and (a,b: std_logic_vector)
        return std_logic_vector;
end my_pack;

package body my_pack is
    function my_and (a,b: std_logic_vector)
        return std_logic_vector is
        variable temp: std_logic_vector( a'range);
    begin
        temp:= a and b;
        return temp;
    end my_and;
end my_pack;

library ieee;
use ieee.std_logic_1164.all;
use work.my_pack.all;

```

```

entity seq is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0) );
end seq;

architecture beh of seq is
begin
    process (clk)
    begin
        if falling_edge(clk) then
            d_out <= my_and(a_in, b_in);
        end if;
    end process;
end beh;

```

CD218

@E: Declaration hides port <qrs>

A signal is declared as a port and is also declared in the architecture of the entity in which the port has been declared. Note that the warning “Port qrs hidden by declaration in architecture” appears in conjunction with this error. By default, the signal declaration overrides the port declaration. In the following test case, the signal qrs overrides the port declaration which causes port qrs to be tied to 0 as it is undriven. qrs in the test case below is regarded as an internal-only signal.

```

library ieee;
use ieee.std_logic_1164.all;

entity dff1 is
    port (data, clk, reset, set: in std_logic;
          qrs: out std_logic_vector (3 downto 0) );
end dff1;

architecture async_set_reset of dff1 is
signal qrs: std_logic_vector (3 downto 0);
begin
    setreset: process (clk, reset, set)
    begin
        if reset = '1' then
            qrs <= "0000";

```

```

        elsif set = '1' then
            qrs <= "1111";
        elsif rising_edge(clk) then
            qrs <= data & data & data & data;
        end if;
    end process setreset;
end async_set_reset;

```

Action

Make sure there are no duplicate declarations for ports that can cause the port to be tied to constant. Ports, by default, are defined by VHDL to be of signal data type. To eliminate the error in the above test case, comment out the line signal qrs: std_logic_vector (3 downto 0); to realize the intended behavior of qrs (set-reset 4-bit flip-flop).

```

library ieee;
use ieee.std_logic_1164.all;

entity dff1 is
    port (data, clk, reset, set: in std_logic;
          qrs: out std_logic_vector (3 downto 0) );
end dff1;

architecture async_set_reset of dff1 is
--signal qrs: std_logic_vector (3 downto 0);
begin
    setreset: process (clk, reset, set)
    begin
        if reset = '1' then
            qrs <= "0000";
        elsif set = '1' then
            qrs <= "1111";
        elsif rising_edge(clk) then
            qrs <= data & data & data & data;
        end if;
    end process setreset;
end async_set_reset;

```

CD219

@W: Port <qrs> hidden by declaration in architecture

A signal is declared as a port and also declared in the architecture of the entity in which this port has been declared. Note that the warning “Declaration hides port qrs” appears in conjunction with this warning. By default, the signal declaration overrides the port declaration. In the following test case, the signal qrs overrides the port declaration which causes port qrs to be tied to 0 because it is undriven. qrs is the test case below is an internal-only signal.

```
library ieee;
use ieee.std_logic_1164.all;

entity dff1 is
    port (data, clk, reset, set: in std_logic;
          qrs: out std_logic_vector (3 downto 0) );
end dff1;

architecture async_set_reset of dff1 is
signal qrs: std_logic_vector (3 downto 0);
begin
    setreset: process (clk, reset, set)
    begin
        if reset = '1' then
            qrs <= "0000";
        elsif set = '1' then
            qrs <= "1111";
        elsif rising_edge(clk) then
            qrs <= data & data & data & data;
        end if;
    end process setreset;
end async_set_reset;
```

Action

Make sure there are no duplicate declarations for ports which can cause the port to be tied to a constant. Ports, by default, are defined by VHDL to be of signal data type. To eliminate the warning in the above test case, comment out the line signal qrs: std_logic_vector (3 downto 0); to realize the intended behavior of qrs (reset-set 4-bit flip-flop).

```
library ieee;
use ieee.std_logic_1164.all;
```

```

entity dff1 is
    port (data, clk, reset, set: in std_logic;
          qrs: out std_logic_vector (3 downto 0) );
end dff1;

architecture async_set_reset of dff1 is
--signal qrs: std_logic_vector (3 downto 0);
begin
    setreset: process (clk, reset, set)
    begin
        if reset = '1' then
            qrs <= "0000";
        elsif set = '1' then
            qrs <= "1111";
        elsif rising_edge(clk) then
            qrs <= data & data & data & data;
        end if;
    end process setreset;
end async_set_reset;

```

CD220

@W: Function already has body!

The VHDL files present in the project file for compilation contain more than one definition for the same function. The function can be overloaded by having different bodies for the functions and the same arguments. When the compiler finds two function definitions in the visible libraries (made visible through library and use clauses), it uses the last function analyzed and issues a warning. In the test case below, the function definition for Pow is analyzed twice and compiled in the work library which is visible by default for all VHDL entities.

```

package specialfunctions is
    function Pow (N,Exp : integer) return integer;
end specialfunctions;

package body specialFunctions is
    function Pow (N,Exp : integer) return integer is
        variable Result1 : integer := 1;
    begin
        for I in 1 to Exp -1 loop

```

```
        Result1 := Result1 * N;
    end loop;
return (Result1);
end Pow;
end specialFunctions;

package body specialFunctions is
function Pow (N,Exp : integer) return integer is
variable Result : integer := 1;
begin
    for I in 1 to Exp loop
        Result := Result * N;
    end loop;
return (Result);
end Pow;
end specialFunctions;

-- 4 bit up-counter with load and asynchronous low reset
library ieee,functions_lib;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;
use work.specialfunctions.all;

entity powerof is
    port (clk: in std_logic;
          input: in unsigned (3 downto 0);
          power: out unsigned (15 downto 0) );
end powerof;

architecture behave of powerof is
signal inputValInt : integer range 0 to 15;
signal powerL : integer range 0 to 65535;
begin
begin
inputValInt<= to_integer (input);
power <= to_unsigned (powerL,16);
process begin
    wait until CLk='1';
        powerL <= Pow (inputValInt, 4);
    end process;
end behave;
```

Action

Make sure that only one function body is specified for each specific function. If the function must be overloaded, make sure that there is no discrepancy in what is picked and what is intended. Make sure you specify the library in which the function is defined as it is given higher priority than the one present in the default work library. To eliminate the warning in the above case, it is more useful to segregate the pow function definition into a different library and make it visible by using a library-use clause.

```

specialfunctions.vhd
package specialfunctions is
function Pow (N,Exp : integer) return integer;
end specialfunctions;

package body specialFunctions is
function Pow (N,Exp : integer) return integer is
variable Result : integer := 1;
begin
    for I in 1 to Exp loop
        Result := Result * N;
    end loop;
    return (Result);
end Pow;
end specialFunctions;

--dff2.vhd
library ieee,functions_lib;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;
use work.specialfunctions.all;

entity powerof is
    port (clk: in std_logic;
          input: in unsigned (3 downto 0);
          power: out unsigned (15 downto 0) );
end powerof;

architecture behave of powerof is
signal inputValInt : integer range 0 to 15;
signal powerL : integer range 0 to 65535;
begin
inputValInt<= to_integer (input);
power <= to_unsigned (powerL,16);

```

```

process begin
    wait until CLk='1';
        powerL <= Pow (inputValInt, 4);
    end process;
end behave;

```

Project file snippet:

```

#add_file options
add_file -vhdl -lib functions_lib "specialfunctions.vhd"
add_file -vhdl -lib work "dff2.vhd"

```

CD222

@W: Label "<A>" is already declared as port name.

A port name is being used as a loop label. In the test case below, port name A is also used as the label for the for loop which causes the warning.

```

library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (A : out std_logic_vector(7 downto 0);
          B : in std_logic_vector(7 downto 0);
          C : in std_logic_vector(7 downto 0));
end seq;

Architecture RTL of seq is
Begin
    process (B,C)
    begin
        A: for i in 7 downto 0 loop
            if (B(i) = '0') then
                A(i) <= C(i);
            else
                A(i) <= 'Z';
            end if;
        end loop A;
    end process;
end rtl;

```

Action

Make sure that the label used in the for loop is not defined as a port. In the corrected test case below, the loop label is given as A1 which is not declared as a port.

```

library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (A : out std_logic_vector(7 downto 0);
          B : in std_logic_vector(7 downto 0);
          C : in std_logic_vector(7 downto 0));
end seq;

Architecture RTL of seq is
Begin
    process (B,C)
    begin
        A1: for i in 7 downto 0 loop
            if (B(i) = '0') then
                A(i) <= C(i);
            else
                A(i) <= 'Z';
            end if;
        end loop A1;
    end process;
end rtl;

```

CD223

@W: Ignoring property syn_preserve on Bi-Directional Port

A `syn_preserve` synthesis directive is being used on an inout port. The `syn_preserve` directive can be used only on output ports, internal signals, and the architecture of an entity. It is ignored if it is used in association with ports that are bidirectional as shown by the port `qrs` in the test case below.

```

library ieee;
use ieee.std_logic_1164.all;

```

```

entity dff1 is
    port (data, clk, reset, set,load: in std_logic;
          qrs1,qrs2: inout std_logic);
attribute syn_preserve : boolean;
attribute syn_preserve of qrs2 : signal is true;
end dff1;

architecture async_set_reset of dff1 is
begin
setreset:
    process (clk, reset, set)
    begin
        if reset = '1' then
            qrs1 <= '0';
            qrs2 <= '0';
        elsif set = '1' then
            qrs1 <= '1';
            qrs2 <= '1';
        elsif rising_edge(clk) then
            if (load ='1') then
                qrs1 <= data;
                qrs2 <= data;
            else
                qrs1 <= 'Z';
                qrs2 <= 'Z';
            end if;
        end if;
    end process setreset;
end async_set_reset;

```

Action

Make sure that the synthesis directive is used correctly. To eliminate the warning in the above test case, change the port direction to output and comment out the else statement.

```

library ieee;
use ieee.std_logic_1164.all;

entity dff1 is
    port (data, clk, reset, set,load: in std_logic;
          qrs1,qrs2: out std_logic);
attribute syn_preserve : boolean;
attribute syn_preserve of qrs2 : signal is true;
end dff1;

```

```

architecture async_set_reset of dff1 is
begin
setreset:
  process (clk, reset, set)
  begin
    if reset = '1' then
      qrs1 <= '0';
      qrs2 <= '0';
    elsif set = '1' then
      qrs1 <= '1';
      qrs2 <= '1';
    elsif rising_edge(clk) then
      if (load ='1') then
        qrs1 <= data;
        qrs2 <= data;
      --else
        --qrs1 <= 'Z';
        --qrs2 <= 'Z';
      end if;
    end if;
  end process setreset;
end async_set_reset;

```

The Z assignment is commented out because the ports behave as output ports.

CD224

@E: Nil range in ports not yet supported: <c> (<-1> downto <-2>)

An incorrect range direction was given for an array object that has negative indices. In the test case below, the range of array object d has negative indices with an incorrect direction (-1 to -2) which causes the compiler to error out.

```

library ieee;
use ieee.std_logic_1164.all;

entity comb is
  port (A: in std_logic;
        B: in std_logic;
        C: out std_logic_vector( -1 to -2 ));
end comb;

```

```
architecture rtl of comb is
begin
    c<= (a & b);
end rtl;
```

Action

Make sure to specify the proper direction when using negative indices for range declarations (the compiler remaps the negative indices to a positive range). Changing the range of array d as shown in the corrected test case below corrects the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic_vector( -1 downto -2 ));
end comb;

architecture rtl of comb is
begin
    c<= (a & b);
end rtl;
```

CD225

@E: Nil range in ports not yet supported: <q> (<-2> to <-1>)

An incorrect range direction was given for an array object that has negative indices. In the test case below, the range of array object d has negative indices with an incorrect direction (-2 downto -1) which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (A: in std_logic;
          B: in std_logic;
          d: out std_logic_vector(-2 downto -1) ;
end comb;
```

```
architecture rtl of comb is
begin
    d <= a & b;
end rtl;
```

Action

Make sure that the proper direction is specified when using negative indices for range declarations (the compiler remaps the negative indices to a positive range). Changing the range of array d as shown in the corrected test case below corrects the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (A: in std_logic;
          B: in std_logic;
          d: out std_logic_vector(-2 to -1) );
end comb;

architecture rtl of comb is
begin
    d <= a & b;
end rtl;
```

CD228

@E: Array has negative length!

The vector array defined in VHDL has a negative length. If the vector array is defined using an ascending range, the index of the left most value must be less than or equal to the index of the right most value. Similarly for an array of descending range, the left most index must be greater than or equal to the index of the right most value. In the following test case, ports a and d_out have been defined incorrectly which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity test is
  port (
    clk: in std_logic;
    a: in std_logic_vector(3 downto 5);
    d_out: out std_logic_vector(3 downto 5)
  );
end test;

architecture beh of test is
begin
  process (clk)
  begin
    if rising_edge(clk) then
      d_out <= a;
    end if;
  end process;end beh;
```

Action

Define the range of the vector array such that it has a positive length. The ports in the corrected test case below are declared as `std_logic_vector` of ascending range.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
  port (
    clk: in std_logic;
    a: in std_logic_vector(3 to 5);
    d_out: out std_logic_vector(3 to 5)
  );
end test;

architecture beh of test is
begin
  process (clk)
  begin
    if rising_edge(clk) then
      d_out <= a;
    end if;
  end process;
end beh;
```

CD231

@N: Using onehot encoding for type <state_values> (<sx>=<"100000">)

An enumerated type is declared in the VHDL code. By default, enumerated types are always encoded to sequential encoding when extracted as state registers. This encoding is not dependent on the number of states. You can globally change the enumerated encoding in the graphical interface (select Implementation Options->VHDL-> Default Enum Encoding and select the encoding style from the drop down menu).

If you require individual state-machine controls, apply the `syn_enum_encoding` attribute to the state-machine definition in the source code. Enumerated encoding, applied with the attribute, will be the final output encoding of the state registers when the FSM compiler is turned off (the default). When the FSM compiler is turned on, enumerated encoding can be overridden by the `syn_encoding` attribute.

```
library ieee;
use ieee.std_logic_1164.all;

entity stmchl is
    port (clk, in1, rst: in std_logic;
          temp: out std_logic );
end stmchl;

architecture behave of stmchl is
type state_values is (sx, s0, s1, s2,s3,s4);
signal state, next_state: state_values;
begin
    process (clk, rst)
    begin
        if rst = '1' then
            state <= s0;
        elsif rising_edge(clk) then
            state <= next_state;
        end if;
    end process;

    process (state, in1)
    begin
-- set defaults for output and state
        temp <= '0';
        next_state <= sx; -- catch missing assignments to next_state
```

```
case state is
    when s0 =>
        if in1 = '0' then
            temp <='1';
            next_state <= s1;
        else
            temp <= '0';
            next_state <= s0;
        end if;
    when s1 =>
        if in1 = '0' then
            temp <='0';
            next_state <= s0;
        else
            temp <= '1';
            next_state <= s2;
        end if;
    when s2 =>
        if in1 = '0' then
            temp <='0';
            next_state <= s3;
        else
            temp <= '1';
            next_state <= s2;
        end if;
    when s3 =>
        if in1 = '0' then
            temp <='0';
            next_state <= s1;
        else
            temp <= '1';
            next_state <= s4;
        end if;
    when s4 =>
        if in1 = '0' then
            temp <='0';
            next_state <= s0;
        else
            temp <= '1';
            next_state <= s2;
        end if;
    when sx =>
        next_state <= sx;
    end case;
end process;
end behave;
```

Action

The compiler writes out a note with the default encoding used. The mapper remaps the encoding of the state machine by using the `syn_encoding` specification when the FSM compiler is on. In the log file, the one-to-one match on the encodings in the compiler and the mapper is reported. The following is an excerpt from the log file after synthesizing a state machine with five states that is reencoded as sequential due to the `syn_encoding` specification

```
Reading constraint file:
D:\amp322\examples\vhd1\statmchs\statmch1.sdc
Adding property syn_encoding, value "sequential", to instance
state[0:4]
Encoding state machine work.stmchl(behave)-state_h.state[0:4]
original code -> new code
 000010 -> 000
 001100 -> 001
 001000 -> 010
 010000 -> 011
 100000 -> 100
```

The above explanation assumes that enumerated encoding is applied to enumerated types that are extracted as state registers. When the enumerated register is not a state register, the enumerated encoding specified through a `syn_enum_encoding` attribute is applied irrespective of the FSM compiler setting.

CD232

@N: Using gray code encoding for type <state_type>

The type of encoding being used. The above note is issued if either:

- the `syn_enum_encoding` directive is not applied and the number of states is greater than 25
- the `syn_enum_encoding` directive is applied with an attribute value of `gray` (irrespective of whether the FSM compiler is enabled or disabled)

In the absence of a `syn_enum_encoding` directive, if the compiler detects an enumerated type declared in the VHDL code, it automatically assigns an encoding style based on the number of states:

- 0-4: sequential
- 5-24: onehot
- >25: gray

In the test case below, the syn_enum_encoding directive is applied to the enumerated type state_type with a value of gray. Even though the number of states is 3, gray encoding is used as indicated by the above note.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (clk, rst: bit;
          O: out std_logic_vector(2 downto 0) );
end seq;

architecture behave of seq is
type state_type is (S0, S1, S2);
attribute syn_enum_encoding: string;
attribute syn_enum_encoding of state_type : type is "gray";

signal machine : state_type;
begin
    process (clk, rst)
    begin
        if rst = '1' then
            machine <= S0;
        elsif clk = '1' and clk'event then
            case machine is
                when S0 => machine <= S1;
                when S1 => machine <= S2;
                when S2 => machine <= S0;
                when others => null;
            end case;
        end if;
    end process;

    with machine select
        O <= "001" when S0,
        "010" when S1,
        "101" when S2;
end behave;
```

Action

The encoding style can be overridden by changing the value of the `syn_enum_encoding` directive when the FSM compiler is disabled. When the FSM compiler is enabled, use the `syn_encoding` attribute instead of the `syn_enum_encoding` directive to specify the encoding styles of extracted state machines.

CD233

@N: Using sequential encoding for type <state_values>

An enumerated type has been declared in the VHDL code; the following encoding is used for the enumerated type in a state machine based on the number of states:

- 0-4: sequential
- 5-24: onehot
- >25: gray

These encodings can be overwritten by using the `syn_encoding` attribute when the FSM compiler is on. You can use `syn_enum_encoding` to define one of the above encoding styles or any other encoding that needs to be implemented. Note that for `syn_enum_encoding` to be honored, the FSM compiler must be turned off.

```
library ieee;
use ieee.std_logic_1164.all;

entity stmchl is
    port (clk, in1, rst: in std_logic;
          out1: out std_logic );
end stmchl;

architecture behave of stmchl is
type state_values is (sx, s0, s1);
signal state, next_state: state_values;
begin
    process (clk, rst)
    begin
        if rst = '1' then
```

```
        state <= s0;
      elsif rising_edge(clk) then
        state <= next_state;
      end if;
    end process;

  process (state, in1)
  begin
    -- set defaults for output and state
    out1 <= '0';
    next_state <= sx; -- catch missing assignments to next_state
    case state is
      when s0 =>
        if in1 = '0' then
          out1 <='1';
          next_state <= s1;
        else
          out1 <= '0';
          next_state <= s0;
        end if;
      when s1 =>
        if in1 = '0' then
          out1 <='0';
          next_state <= s0;
        else
          out1 <= '1';
          next_state <= s1;
        end if;
      when sx =>
        next_state <= sx;
    end case;
  end process;
end behave;
```

Action

The compiler writes out a note with the default encoding used. The mapper remaps the encoding of the state machine by using the `syn_encoding` specification when the FSM compiler is on. In the log file, the one-to-one match on the encodings in the compiler and the mapper is reported. The following is an excerpt from the log file after synthesizing a state machine that has five states which would be implemented as a onehot in the compiler, but is reencoded as sequential due to the `syn_encoding` specification

```
Reading constraint file:  
D:\amp322\examples\vhd1\statmchs\statmch1.sdc  
Adding property syn_encoding, value "sequential", to instance  
state[0:4]  
Encoding state machine work.stmchl(behave)-state_h.state[0:4]  
original code -> new code  
000010 -> 000  
000100 -> 001  
001000 -> 010  
010000 -> 011  
100000 -> 100
```

The above explanation assumes that enumerated encoding is applied to enumerated types that are extracted as state registers. When the enumerated register is not a state register, the enumerated encoding specified through a `syn_enum_encoding` attribute is applied irrespective of the FSM compiler setting.

CD234

@N: Using user defined encoding for type <`state_values`>

An enumerated type has been declared in the VHDL code; the following encoding is used for the enumerated type in a state machine based on the number of states:

- 0-4: sequential
- 5-24: onehot
- >25: gray

These encodings can be overwritten by using the `syn_encoding` attribute when the FSM compiler is on. You can use `syn_enum_encoding` to define one of the above encoding styles, or any other encoding to be implemented. Note that for `syn_enum_encoding` to be honored, the FSM compiler must be turned off. The following test case uses `syn_enum_encoding` to implement the user-defined encoding

```
library ieee;  
use ieee.std_logic_1164.all;
```

```
entity stmchl is
    port (clk, in1, rst: in std_logic;
          out1: out std_logic );
end stmchl;

architecture behave of stmchl is
type state_values is (sx, s0, s1,s2,s3,s4);
attribute syn_enum_encoding : string;
attribute syn_enum_encoding of state_values: type
    is "101 111 100 110 000 010";
signal state, next_state: state_values;
begin
begin
    process (clk, rst)
begin
    if rst = '1' then
        state <= s0;
    elsif rising_edge(clk) then
        state <= next_state;
    end if;
end process;

    process (state, in1)
begin
-- set defaults for output and state
out1 <= '0';
next_state <= sx; -- catch missing assignments to next_state
case state is
    when s0 =>
        if in1 = '0' then
            out1 <='1';
            next_state <= s1;
        else
            out1 <= '0';
            next_state <= s0;
        end if;
    when s1 =>
        if in1 = '0' then
            out1 <='0';
            next_state <= s0;
        else
            out1 <= '1';
            next_state <= s2;
        end if;
    when s2 =>
        if in1 = '0' then
            out1 <='0';
            next_state <= s3;
```

```

        else
            out1 <= '1';
            next_state <= s2;
        end if;
    when s3 =>
        if in1 = '0' then
            out1 <='0';
            next_state <= s1;
        else
            out1 <= '1';
            next_state <= s4;
        end if;
    when s4 =>
        if in1 = '0' then
            out1 <='0';
            next_state <= s0;
        else
            out1 <= '1';
            next_state <= s2;
        end if;
    when others =>
        next_state <= sx;
    end case;
end process;
end behave;

```

Action

The compiler and mapper honor the user-defined enumerated type encoding if the FSM compiler is off. In the above test case, the encodings of the states are as follows

```

State sx => 101
State s0 => 111
State s1 => 100
State s2 => 110
State s3 => 000
State s4 => 010

```

The above explanation assumes that enumerated encoding is applied to enumerated types that are extracted as state registers. When the enumerated register is not a state register, the enumerated encoding specified through a `syn_enum_encoding` attribute is applied irrespective of the FSM compiler setting.

CD235

@E: Expecting entity or configuration name

A binding indication did not have an entity name or a configuration name.

The syntax of a configuration specification is:

for *componentLabelList: componentName* **use** *bindingIndication*

In the above syntax statement, *bindingIndication* can be any of the following three forms:

use entity *entityName* [(*architectureName*)]
[**generic map** (*genericAssociationList*)]
[**port map** (*associationList*)];

use configuration *configurationName*

use open (not supported)

In the test case below, the binding indication ends with the keyword use and does not have either an entity name or a configuration name which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity HA is
    port (U,V: in bit;
          X,Y: out bit);
end HA;

architecture beh of Ha is
begin
    x<= u xor v;
    y<= u and v;
end beh;

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (A, B: in bit;
          SUM, CARRY: out bit);
end test;
```

```
architecture beh of test is
component dummy
    port (A, B:in bit;
          SUM, CARRY: out bit);
end component;

for u1: dummy use ;
begin
    u1: dummy port map (A, B, SUM,CARRY);
end beh;
```

Action

Be sure to specify an entity name or a configuration name in a binding indication. In the test case below, entity HA is added to the configuration specification to correct the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity HA is
    port (U,V: in bit;
          X,Y: out bit);
end HA;

architecture beh of Ha is
begin
    x<= u xor v;
    y<= u and v;
end beh;

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (A, B: in bit;
          SUM, CARRY: out bit);
end test;

architecture beh of test is
component dummy
    port (A, B:in bit;
          SUM, CARRY: out bit);
end component;
```

```

for u1: dummy use entity HA port map ( A, B, SUM,CARRY);
begin
    u1: dummy port map (A, B, SUM,CARRY);
end beh;

```

CD236

@E: Expecting configuration name, got keyword <end>.

The keyword configuration is specified and is not followed by the configuration. The general syntax of a configuration is:

```

configuration configurationName of entityName is
    for architecture
    end for;
end configurationName

```

In the following example, the incomplete configuration specification results in the error when the final keyword end is encountered.

```

library ieee;
use ieee.std_logic_1164.all;

entity fa is
port (a,b,cin : in std_logic;
      sum,carry : out std_logic);
end fa;

architecture ex1 of fa is
begin
    sum <= a xor b xor cin;
    carry <= ((a and b) or ( a and cin) or ( b and cin));
end ex1;

architecture ex2 of fa is
begin
    sum   <= a xor b xor cin;
    carry <= ((a and b) or (cin and (a xor b)));
end ex2;

configuration
end adder;

```

Action

Complete the configuration specification as shown in the following example.

```

library ieee;
use ieee.std_logic_1164.all;

entity fa is
port (a,b,cin : in std_logic;
      sum,carry : out std_logic);
end fa;

architecture ex1 of fa is
begin
    sum <= a xor b xor cin;
    carry <= ((a and b) or ( a and cin) or ( b and cin));
end ex1;

architecture ex2 of fa is
begin
    sum  <= a xor b xor cin;
    carry <= ((a and b) or (cin and (a xor b)));
end ex2;

configuration adder of fa is
    for ex1
        end for;
end adder;
```

CD241

@E: Expecting configuration name <cfg>.

The configuration name at the end of a configuration block did not match the configuration name specified in the first line of the configuration declaration. In the test case below, the configuration name at the start of the configuration block (CFG) is different from the configuration name at the end of the configuration block (CFG1) which causes the error.

```

entity comb is
port (U,V: in bit;
      X,Y: out bit);
end comb;
```

```

architecture GATE of comb is
begin
    x<= u xor v;
    y<= u and v;
end;

entity top is
    port (A, B: in bit;
          SUM, CARRY: out bit);
end top;

architecture STRUCT of top is
component HALFADDER
    port (A, B: in bit;
          SUM, CARRY: out bit);
end component;

begin
    u1: HALFADDER port map (A, B, SUM,CARRY);
end STRUCT;

configuration CFG of top is
    for STRUCT
        for u1: HALFADDER use entity work.comb(GATE)
            port map ( U => A, V => B, X => SUM);
        end for;
    end for;
end CFG1;

```

Action

Make sure that the configuration names specified at the start and end of the configuration block are the same. Changing the CFG1 configuration name at the end of the configuration block to CFG as shown in the following test case corrects the error.

```

entity comb is
    port (U,V: in bit;
          X,Y: out bit);
end comb;

architecture GATE of comb is
begin
    x<= u xor v;
    y<= u and v;
end;

```

```

entity top is
    port (A, B: in bit;
          SUM, CARRY: out bit);
end top;

architecture STRUCT of top is
component HALFADDER
    port (A, B: in bit;
          SUM, CARRY: out bit);
end component;

begin
    u1: HALFADDER port map (A, B, SUM,CARRY);
end STRUCT;

configuration CFG of top is
    for STRUCT
        for u1: HALFADDER use entity work.comb(GATE)
            port map ( U => A, V => B, X => SUM);
        end for;
    end for;
end CFG;

```

CD243

@W: Keyword open is not supported in entity aspect, and is ignored.

An open construct is being used in the *entity_aspect*. The *entity_name* and *configuration_name* can be used in the *entity_aspect* of a *binding_indication*. The syntax of a configuration specification is:

for *list_of_comp_labels* : *componentName binding_indication*;

where the *binding_indication* syntax is:

use *entity_aspect*

In the test case below, the keyword open is used in the *entity-aspect* part of the configuration specification which causes the warning.

```
entity HA is
    port (U,V: in bit;
          X,Y: out bit);
end HA;

architecture str of HA is
begin
    x<= u xor v;
    y<= u and v;
end str;

architecture beh of HA is
begin
    process(u,v)
    begin
        y<= u xor v;
        x<= u and v;
    end process;
end beh;

entity comb is
    port (A, B: in bit;
          SUM, CARRY: out bit);
end comb;

architecture beh of comb is
component HA
    port (U,V: in bit;
          X,Y: out bit);
end component;

for u1: HA use open;
begin
    u1: HA port map (A,B,SUM,CARRY);
end beh;
```

Action

Avoid using the keyword `open` in the *entity-aspect* part of the configuration specification. In the corrected test case below, the *entity-aspect* uses the string “**entity entity_name [(architecture_identifier)]**” to eliminate the warning.

```
entity HA is
    port (U,V: in bit;
          X,Y: out bit);
end HA;
```

```
architecture str of HA is
begin
    x<= u xor v;
    y<= u and v;
end str;

architecture beh of HA is
begin
    process(u,v)
    begin
        y<= u xor v;
        x<= u and v;
    end process;
end beh;

entity comb is
    port (A, B: in bit;
          SUM, CARRY: out bit);
end comb;

architecture beh of comb is
component HA
    port (U,V: in bit;
          X,Y: out bit);
end component;

for ul: HA use entity HA( str);
begin
    ul: HA port map (A,B,SUM,CARRY);
end beh;
```

CD244

@E: No architecture associated with entity <HA>

An instantiated entity had no associated architecture and there was also no architecture name specified in the configuration-specification statement.

The syntax of a configuration specification is:

for componentLabelList: componentName use bindingIndication

In the above syntax statement, *bindingIndication* can be any of the following three forms:

```
use entity entityName [(architectureName)]
[generic map (genericAssociationList)]
[port map (associationList)];  
  
use configuration configurationName  
  
use open (not supported)
```

In the test case below, the configuration specification has no architecture name specified and the instantiated entity HA has no associated architecture which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity HA is
    port (U,V: in bit;
          X,Y: out bit);
end HA;

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (A, B: in bit;
          SUM, CARRY: out bit);
end test;

architecture beh of test is
component dummy
    port (A, B:in bit;
          SUM, CARRY: out bit);
end component;

begin
    ul: dummy port map (A, B, SUM,CARRY);
end beh;

configuration my_config of test is
    for beh
        for ul:dummy use entity HA port map (U => A, V => B, X =>
SUM);
        end for ;
    end for ;
end my_config;
```

Action

Make sure an architecture is specified for the entity being instantiated so that the configuration-specification statement can use that architecture by default. In the test case below, associating architecture str with entity HA eliminates the need to explicitly specify the architecture name in the configuration specification which corrects the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity HA is
    port (U,V: in bit;
          X,Y: out bit);
end HA;

architecture str of Ha is
begin
    x<= u xor v;
    y<= u and v;
end str;

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (A, B: in bit;
          SUM, CARRY: out bit);
end test;

architecture beh of test is
component dummy
    port (A, B:in bit;
          SUM, CARRY: out bit);
end component;

begin
    u1: dummy port map (A, B, SUM,CARRY);
end beh;

configuration my_config of test is
    for beh
        for u1:dummy use entity HA port map (U => A, V => B, X =>
SUM);
        end for ;
    end for ;
end my_config;
```

CD246

@W: Range has negative width!

An incorrect syntax was used for specifying a descending range. The following test case contains an argument (a) using an ascending range syntax to specify a descending range which causes the warning.

```
library ieee;
use ieee.std_logic_1164.all;

entity and2 is
    port (a: in std_logic_vector (0 to 1);
          c: out std_logic_vector (1 downto 0) );
end;

architecture rtl of and2 is
begin
    c (1 downto 0) <= a(1 to 0);
end;
```

Action

Make sure that vectors are defined in the ascending order with the keyword to and in descending order with the keyword downto. To eliminate this warning, change the vector order in the architecture declaration.

```
library ieee;
use ieee.std_logic_1164.all;

entity and2 is
    port (a: in std_logic_vector (0 to 1);
          c: out std_logic_vector (1 downto 0) );
end;

architecture rtl of and2 is
begin
    c (1 downto 0) <= a(0 to 1);
end;
```

CD247

@N: Instance <u1> is bound to entity <HA>, architecture <beh>.

A configuration specification statement is used and indicates the entity-architecture pair binding.

The syntax of a configuration specification is:

```
for list_of_comp_labels : componentName binding_indication;
```

where the *binding_indication* syntax is:

```
use entity_aspect
```

In the test case below, entity HA has two architectures, str and beh. Because the *architecture_identifier* is not present in the configuration specification, the instance is bound by default to the last architecture (beh) as indicated by the above note.

```
entity HA is
    port (U,V: in bit;
          X,Y: out bit);
end HA;

architecture str of Ha is
begin
    x<= u xor v;
    y<= u and v;
end str;

architecture beh of HA is
begin
    process(u,v)
    begin
        y<= u xor v;
        x<= u and v;
    end process;
end beh;

entity test2 is
    port (A, B: in bit;
          SUM, CARRY: out bit);
end test2;
```

```
architecture beh of test2 is
component dummy
    port (A, B:in bit;
          SUM, CARRY: out bit);
end component;
for u1: dummy use entity HA port map (A,B,SUM,CARRY);
begin
    u1: dummy port map (A, B, SUM,CARRY);
end beh;
```

Action

Any other architecture of an entity can be specified for binding an instance. In the modified test case below, the *entity_aspect* has the architecture name str which causes the compiler to use that architecture for binding instance u1.

```
entity HA is
    port (U,V: in bit;
          X,Y: out bit);
end HA;
architecture str of Ha is
begin
    x<= u xor v;
    y<= u and v;
end str;
architecture beh of HA is
begin
    process(u,v)
    begin
        y<= u xor v;
        x<= u and v;
    end process;
end beh;
entity test3 is
    port (A, B: in bit;
          SUM, CARRY: out bit);
end test3;
```

```

architecture beh of test3 is
component dummy
    port (A, B:in bit;
          SUM, CARRY: out bit);
end component;

for u1: dummy use entity HA(str) port map (A,B,SUM,CARRY);
begin
    u1: dummy port map (A, B, SUM,CARRY);
end beh;

```

CD248

@N: Instance <u1> is bound to default architecture

A *binding_indication* is missing in a configuration specification statement. The syntax of a configuration specification is:

for *list_of_comp_labels* : *componentName binding_indication*;

If the component declaration is the same as any of the entities in the working library, the default architecture for that entity is used; if there is no corresponding entity, the compiler still issues the note but considers the component as a black box.

In the test case below, the *binding_indication* is missing in the configuration specification statement but, because the HA component declaration is the same as entity HA in the working library, the default architecture of beh is used.

```

entity HA is
    port (U,V: in bit;
          X,Y: out bit);
end HA;

architecture str of HA is
begin
    x<= u xor v;
    y<= u and v;
end str;

```

```
architecture beh of HA is
begin
    process(u,v)
    begin
        y<= u xor v;
        x<= u and v;
    end process;
end beh;

entity comb is
    port (A, B: in bit;
          SUM, CARRY: out bit);
end comb;

architecture beh of comb is
component HA
    port (U,V: in bit;
          X,Y: out bit);
end component;

for ul: HA ;
begin
    ul: HA port map (A,B,SUM,CARRY);
end beh;
```

Action

If a different architecture is to be used, the *binding_indication* must be specified with the proper entity-architecture pair name as shown in the modified test case below.

```
entity HA is
    port (U,V: in bit;
          X,Y: out bit);
end HA;

architecture str of HA is
begin
    x<= u xor v;
    y<= u and v;
end str;
```

```

architecture beh of HA is
begin
  process(u,v)
  begin
    y<= u xor v;
    x<= u and v;
  end process;
end beh;

entity comb is
  port (A, B: in bit;
        SUM, CARRY: out bit);
end comb;

architecture beh of comb is
component dummy
  port (A, B:in bit;
        SUM, CARRY: out bit);
end component;

for u1: dummy use entity HA(str) port map (A,B,SUM,CARRY);
begin
  u1: dummy port map (A, B, SUM, CARRY);
end beh;

```

CD249

@N: Instance <u1> is bound to configuration <adder_config>, entity <fa>, architecture <adder>.

An instance is bound to a particular configuration (entity-architecture pair) through a use configuration statement.

In the test case below, low-level entity fa has two architectures, adder and sub, which are configured by the configurations adder_config and sub_config, respectively. For top-level entity fulladder, the configuration fulladder_config binds instance u1 to entity fa and architecture adder through configuration adder_config.

```
entity fa is
    port (a,b,cin : in bit;
          sum,carry : out bit);
end fa;

architecture adder of fa is
begin
    sum <= a xor b xor cin;
    carry <= ((a and b) or ( a and cin) or ( b and cin));
end adder;

architecture sub of fa is
begin
    sum <= a xor b xor cin;
    carry <= (not a) and ( b or cin);
end sub;

configuration adder_config of fa is
    for adder
        end for;
    end adder_config;

configuration sub_config of fa is
    for sub
        end for;
    end sub_config;

entity fulladder is
    port (a,b,cin : in bit;
          sum,carry : out bit);
end fulladder;

architecture beh of fulladder is
component fa is
    port (a,b,cin : in bit;
          sum,carry : out bit);
end component;

begin
ul: fa port map (a,b,cin,sum,carry);
end beh;

configuration fulladder_config of fulladder is
    for beh
        for ul: fa use configuration work.adder_config ;
            end for;
    end for;
```

```
end fulladder_config;
```

Action

Note that the particular instance is bound to the entity-architecture pair. The entity-architecture pair can be changed by using the `use configuration` statement as shown in the modified test case below in which the instance is bound to entity `fa` and architecture `sub` through configuration `sub_config`.

```
entity fa is
port (a,b,cin : in bit;
      sum,carry : out bit);
end fa;

architecture adder of fa is
begin
  sum <= a xor b xor cin;
  carry <= ((a and b) or ( a and cin) or ( b and cin));
end adder;

architecture sub of fa is
begin
  sum <= a xor b xor cin;
  carry <= (not a) and ( b or cin);
end sub;

configuration adder_config of fa is
  for adder
    end for;
end adder_config;

configuration sub_config of fa is
  for sub
    end for;
end sub_config;

entity fulladder is
port (a,b,cin : in bit;
      sum,carry : out bit);
end fulladder;

architecture test of fulladder is
component fa is
port (a,b,cin : in bit;
      sum,carry : out bit);
end component;
```

```
begin
u1: fa port map (a,b,cin,sum,carry);
end test;

configuration fulladder_config of fulladder is
  for test
    for u1: fa use configuration work.sub_config ;
      end for;
    end for;
  end fulladder_config;
```

CD251

@E: The comparison operator in VHDL is a single '='

Multiple “=” are being used as the comparison operator. In the test case below, “==” is used as the comparison operator which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity comp is
  port (a,b: in std_logic;
        equal:out std_logic );
end entity;

architecture only of comp is
begin
  equal <= '1' when (a==b) else '0';
end only;
```

Action

Be sure to use a single ‘=’ as the comparison operator as shown in the modified test case below.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
```

```
entity comp is
    port (a,b: in std_logic;
          equal:out std_logic );
end entity;

architecture only of comp is
begin
    equal <= '1' when (a=b) else '0';
end only;
```

CD253

@E: No package <"my_pack"> in library <work>

A reference was encountered to an undeclared package in the accessed library. In the following test case, the compiler identifies the string my_pack as a package because it is preceded by the string work which is a working library. However, because there is no package my_pack declared in working library work, the compiler errors out.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0) );
end test;

architecture beh of test is
begin
    process (clk)
    begin
        if falling_edge(clk) then
            d_out <= work.my_pack.and1(a_in, b_in);
        end if;
    end process;
end beh;
```

Action

Be sure to declare the package in the referenced library when accessing a package with the *library.package* format. In the following test case, package `my_pack` is initially declared which corrects the error.

```
library ieee;
use ieee.std_logic_1164.all;

package my_pack is
    function and1 (a,b: std_logic_vector) return std_logic_vector;
end my_pack;

package body my_pack is
    function and1 (a,b: std_logic_vector) return std_logic_vector
is variable temp: std_logic_vector(a'range);
begin
    temp:= a and b;
    return temp;
end;
end my_pack;

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0) );
end test;

architecture beh of test is
begin
    process (clk)
    begin
        if falling_edge(clk) then
            d_out <= work.my_pack.and1(a_in, b_in);
        end if;
    end process;
end beh;
```

CD255

@E: No identifier "<AB>" in scope

An undefined signal is used in the design. The following test case generates the above error message.

```
library ieee;
use ieee.std_logic_1164.all;

entity error22 is
port (
    A: in std_logic;
    B: in std_logic;
    C: out std_logic_vector(1 downto 0));
end error22;

architecture rtl of error22 is
begin
    C <= AB;
end rtl;
```

Action

Check the signal names in the design to be sure that they are entered correctly. To eliminate the error in the above test case, change the signal name as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity error22 is
port (
    A: in std_logic;
    B: in std_logic;
    C: out std_logic_vector(1 downto 0));
end error22;

architecture rtl of error22 is
begin
    C <= (A, B);
end rtl;
```

CD256

@E: No identifier <a> in package "<my_pack>"

An undefined identifier from an existing package is accessed. In the test case below, the compiler encounters identifier work.my_pack.A which references identifier A declared in package my_pack. Because identifier A is not declared in package my_pack, the compiler errors out.

```
library ieee;
use ieee.std_logic_1164.all;

package my_pack is
end my_pack;

library ieee;
use ieee.std_logic_1164.all;
use work.my_pack.all;

entity comb is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic);
end comb;

architecture rtl of comb is
begin
    process(b,work.my_pack.A)
    begin
        if (work.my_pack.A='1')then
            C <= B;
        else
            C<= '0';
        end if;
    end process;
end rtl;
```

Action

Be sure to declare the identifier in the referenced package and also make sure to link it properly. As shown in the test case below, predefining identifier A as the signal in package my_pack corrects the problem.

```
library ieee;
use ieee.std_logic_1164.all;
```

```

package my_pack is
signal A: std_logic;
end my_pack;

library ieee;
use ieee.std_logic_1164.all;
use work.my_pack.all;

entity comb is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic);
end comb;

architecture rtl of comb is
begin
my_pack.A<=A;
process(b,work.my_pack.A)
begin
    if (work.my_pack.A='1')then
        C <= B;
    else
        C<= '0';
    end if;
end process;
end rtl ;

```

CD259

@E: Qualified expression: requires parenthesized expression

A qualified expression is not enclosed within parenthesis (some constructs such as qualified expressions must be parenthesized). In the test case below, the expression '1' in the qualified expression BIT"1" is not parenthesized which causes the error.

```

entity test2 is
    port (a:in bit;
          b:out bit);
end test2;

```

```
architecture str of test2 is
begin
    process(a)
    begin
        if(a='1')then
            b<= BIT''1';
        else
            b<='0';
        end if;
    end process;
end str;
```

Action

Make sure that all expressions in qualified expressions are enclosed within parenthesis. In the corrected test case below, expression '1' is enclosed in the parenthesis to correct the error.

```
entity test2 is
    port (a:in bit;
          b:out bit);
end test2;

architecture str of test2 is
begin
    process(a)
    begin
        if(a='1')then
            b<= BIT'('1');
        else
            b<='0';
        end if;
    end process;
end str;
```

CD260

@E: Port <z1> not exist in component <mx>

When using a named association for a configuration specification, the actual of the entity is mapped to a non-existent port in the component. In the test case below, actual Y of entity xor1 is mapped to Z1 in the configuration specification. However, because Z1 does not exist in component MX, the compiler errors out.

```
entity xor1 is
    port (X1, X2 : in bit;
          Y : out bit);
    end xor1;

architecture str of xor1 is
begin
    Y <= X1 xor X2;
end str;

entity top is
    port (a, b : in bit;
          cout : out bit);
    end top;

architecture df of top is
component MX
    port (A, B : in bit;
          Z : out bit);
    end component;

begin
    u1 : MX port map (a,b,cout);
end;

configuration c1 of top is
    for df
        for u1 : MX use entity xor1(str) port map
            (X1 => A, X2 => B, Y => Z1);
        end for;
    end for;
end;
```

Action

Be sure to use only predefined ports in component declarations for named associations. In the corrected test case below, all of the actuals of entity xor1 in the configuration specification are mapped to defined ports in component declaration MX.

```
entity xor1 is
    port (X1, X2 : in bit;
          Y : out bit);
end xor1;

architecture str of xor1 is
begin
    Y <= X1 xor X2;
end str;

entity top is
    port (a, b : in bit;
          cout : out bit);
end top;

architecture df of top is
component MX
    port (A, B : in bit;
          Z : out bit);
end component;

begin
    u1 : MX port map (a,b,cout);
end;

configuration c1 of top is
    for df
        for u1 : MX use entity xor1(str) port map
            ( X1 => A, X2 => B, Y => Z);
        end for;
    end for;
end;
```

CD263

@E: indexed name: missing closing)

When accessing a single bit of a vector, the closing bracket following the index is missing. In the test case below, the MSB of vector `sig` is being assigned to output `c`, but because there is no closing bracket after the index 3, the compiler errors out.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity comb is
    port (a,b: in std_logic_vector(3 downto 0);
          dout : out std_logic_vector(2 downto 0);
          c: out std_logic);
end entity;

architecture rtl_dff of comb is
signal sig: std_logic_vector(3 downto 0);
begin
    sig<= a + b;
    dout<= sig(2 downto 0);
    c<= sig(3 ;
end;
```

Action

Be sure the index is enclosed within brackets as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity comb is
    port (a,b: in std_logic_vector(3 downto 0);
          dout : out std_logic_vector(2 downto 0);
          c: out std_logic);
end entity;
```

```

architecture rtl_dff of comb is
signal sig: std_logic_vector(3 downto 0);
begin
    sig<= a + b;
    dout<= sig(2 downto 0);
    c<= sig(3) ;
end;

```

CD264

@E: Expecting assignment target

The target value on the LHS of the expression is not a valid variable or signal. In the following test case, dataz is not a valid signal which results in the error.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity test is
    port (data_out: out std_logic_vector(7 downto 0);
          data_in: in std_logic_vector(63 downto 0);
          selector: in std_logic_vector(255 downto 0) );
end test;

architecture rtl of test is
begin
    with selector(7 downto 0) select dataz <=
        data_in(63 downto 56) when x"80",
        data_in(55 downto 48) when x"40",
        (others => '0') when others;
end rtl;

```

Action

Make sure that all signal names are valid. Changing dataz to data_out as shown in the corrected test case below eliminates the error.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

```

```

entity test is
    port (data_out: out std_logic_vector(7 downto 0);
          data_in: in std_logic_vector(63 downto 0);
          selector: in std_logic_vector(255 downto 0) );
end test;

architecture rtl of test is
begin
    with selector(7 downto 0) select data_out <=
        data_in(63 downto 56) when x"80",
        data_in(55 downto 48) when x"40",
        (others => '0') when others;
end rtl;

```

CD265

@E: Bad suffix

The suffix following a period (.) was not a valid identifier. In VHDL, any identifier following a period is considered a suffix. Generally, periods are used as hierarchy separators. In the test case below, while using record rp, the invalid identifier ‘j’ is specified following the period which causes the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic_vector(1 downto 0));
end comb;

architecture rtl of comb is
TYPE color IS (red, yellow, blue, white);
TYPE rec_ptr IS
RECORD
    d1 : color;
    d2 : std_logic;
END RECORD;

```

```
signal rp : rec_ptr;
begin
    rp <= (d1 => red, d2 => A);
    C <= (B, rp.);  
end rtl;
```

Action

Make sure to use a valid suffix following a period. In the test case below, adding the suffix d2 to record rp corrects the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic_vector(1 downto 0));
end comb;

architecture rtl of comb is
TYPE color IS (red, yellow, blue, white);
TYPE rec_ptr IS
RECORD
    d1 : color;
    d2 : std_logic;
END RECORD;
signal rp : rec_ptr;
begin
    rp <= (d1 => red, d2 => A);
    C <= (B, rp.d2);
end rtl;
```


CHAPTER 8

CD Messages 266 – 387

CD266

@W: <outp1> is not readable. This may cause a simulation mismatch.

An output port is being read within the architecture. The only readable ports are ports that are defined as inputs or inouts. Output ports cannot be used as inputs; values cannot be read from output ports, and values can only be assigned to output ports. Internal signals, signals declared within the architecture, can be read from as well as assigned to within the architecture. In the following test case, outp1 is an output that is incorrectly used to assign outp2.

```
library ieee;
use ieee.std_logic_1164.all;
entity decoder is
    port (inp: in std_logic_vector(2 downto 0);
          outp1: out std_logic_vector(7 downto 0);
          outp2: out std_logic_vector(7 downto 0));
end decoder;

architecture behave of decoder is
begin
    process (inp) begin
        case inp is
            when "000" => outp1 <= "00000001";
            when "001" => outp1 <= "00000010";
```

```

        when "010" => outp1 <= "00000100";
        when "011" => outp1 <= "00001000";
        when "100" => outp1 <= "00010000";
        when "101" => outp1 <= "00100000";
        when "110" => outp1 <= "01000000";
        when "111" => outp1 <= "10000000";
        when others => outp1 <= "XXXXXXXX";
    end case;
    outp2 <= outp1;
end process;
end behave;

```

Action

Make sure that output ports are only assigned to, but not read from. To eliminate the warning in the above test case, assign an internal signal outp to output ports outp1 and outp2 as shown in the corrected test case below.

```

architecture behave of decoder is
signal outp: std_logic_vector (7 downto 0);
begin
    process (inp) begin
        case inp is
            when "000" => outp <= "00000001";
            when "001" => outp <= "00000010";
            when "010" => outp <= "00000100";
            when "011" => outp <= "00001000";
            when "100" => outp <= "00010000";
            when "101" => outp <= "00100000";
            when "110" => outp <= "01000000";
            when "111" => outp <= "10000000";
            when others => outp <= "XXXXXXXX";
        end case;
        outp1 <= outp;
        outp2 <= outp;
    end process;
end behave;

```

CD270

@W: Default relational operators have unexpected behavior for arguments of different widths (<6> vs <8>). See LRM 7.2.2

Relational operators are being used with arrays of different widths which can cause inconsistent results from both the simulator and the synthesis tool as defined in the Language Reference Manual (LRM). The test case below issues the above warning.

```
library ieee;
use ieee.std_logic_1164.all;

entity compare is
    port (a: in bit_vector (5 downto 0);
          b: in bit_vector (7 downto 0);
          equal: out std_logic );
end compare;

architecture behave of compare is
begin
    process (a,b)
    begin
        if (a < b ) then
            equal <= '1';
        else
            equal <= '0';
        end if;
    end process;
end behave;
```

Action

Make sure that all the arguments used with relational operators such as =, /=, ,<, <=, >, >= are of equal length. To eliminate the warning in the above test case, replace the port declaration to match the argument size b and obtain the correct results.

```
library ieee;
use ieee.std_logic_1164.all;
```

```

entity compare is
    port (a: in bit_vector (7 downto 0);
          b: in bit_vector (7 downto 0);
          equal: out std_logic );
end compare;

architecture behave of compare is
begin
    process (a,b)
    begin
        if (a < b ) then
            equal <= '1';
        else
            equal <= '0';
        end if;
    end process;
end behave;

```

CD271

@A: Comparison to '-' in case returns false. Consider use of an if statement with std_match for wild card support

A case statement is using the VHDL wildcard literal ‘-’ in a case choice. The compiler assumes all comparisons with the wildcard literal to be false and does not implement the associated logic. In the following test case, the case branch "00-" is considered false and the associated assignment (outp <= "00000001") is ignored. The RTL view schematic shows that outp(0) is undriven (tied to zero).

```

library ieee;
use ieee.std_logic_1164.all;

entity decoder is
    port (inp: in std_logic_vector(2 downto 0);
          outp: out std_logic_vector(7 downto 0) );
end decoder;

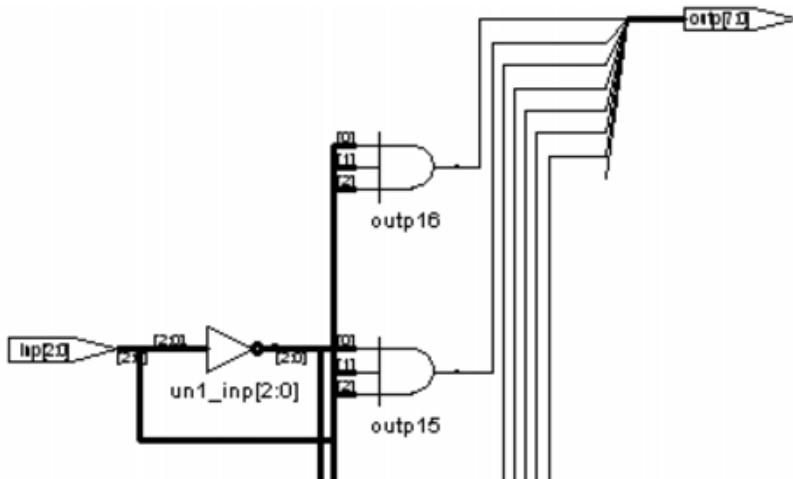
architecture behave of decoder is
begin
    process (inp)
    begin
        case inp is

```

```

when "00-" => outp <= "00000001";
when "001" => outp <= "00000010";
when "010" => outp <= "00000100";
when "011" => outp <= "00001000";
when "100" => outp <= "00010000";
when "101" => outp <= "00100000";
when "110" => outp <= "01000000";
when "111" => outp <= "10000000";
when others => outp <= "XXXXXXXX";
end case;
end process;
end behave;

```



Action

Use the wildcard literal '-' in a case choice when you need a wildcard implementation. If you need to compare a don't care with another don't care, use the std_match function in the numeric_std library.

```

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

entity decoder is
    port (inp: in std_logic_vector(2 downto 0);
          outp: out std_logic_vector(7 downto 0) );
end decoder;

```

```

architecture behave of decoder is
begin
    process (inp)
    begin
        if std_match(inp,"00-") then
            outp <= "00000001";
        else
            case inp is
                when "001" => outp <= "00000010";
                when "010" => outp <= "00000100";
                when "011" => outp <= "00001000";
                when "100" => outp <= "00010000";
                when "101" => outp <= "00100000";
                when "110" => outp <= "01000000";
                when "111" => outp <= "10000000";
                when others => outp <= "XXXXXXXX";
            end case;
        end if;
    end process;
end behave;

```

In the above example, the condition `inp = "00-"` is implemented by a 2-input AND gate with the inverted inputs driving the `outp(0)`th bit as shown in the RTL schematic view.

CD274

@W: Incomplete case statement - add more cases or a when others

The `case` statement used in the HDL code has at least one case choice (branch) that is not defined. This warning can indicate unnecessary logic that is created when an incomplete case statement is specified. In the following test case, `case` mode has only two branches:

```

when "01" =>
when "10" =>

```

Mode is a 2-bit vector, which can have four values “00”, “01”, “10” and “11”. The `case` statement in the following test case is incomplete.

```
library ieee;
use ieee.std_logic_1164.all;

entity shifter is
    port (data: in std_logic_vector (7 downto 0);
          shift_left, shift_right, clk, reset: in std_logic;
          mode: in std_logic_vector (1 downto 0);
          qout: buffer std_logic_vector (7 downto 0) );
end shifter;

architecture behave of shifter is
signal enable: std_logic;
begin
    process
    begin
        wait until (rising_edge(clk));
        if (reset = '1') then
            qout <= "00000000";
        else
            case mode is
                when "01" => qout <= shift_right & qout(7 downto 1);
                -- shift right
                when "10" => qout <= qout(6 downto 0) & shift_left;
                -- shift left
            end case;
        end if;
    end process;
end behave;
```

The tool also generates this message when incomplete case statements are used to define a state machine. The tool automatically optimizes away any duplicate states. The state machine report in the log file includes a list of reachable states, excluding any states that were optimized away.

Action

Make sure that case statements inside the HDL code are complete. If there is a set of case choices that has a don't care condition, use the when others => null; statement. To eliminate the warning in the above test case, add two definitions to the case statement as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity shifter is
    port (data: in std_logic_vector (7 downto 0);
          shift_left, shift_right, clk, reset: in std_logic;
          mode: in std_logic_vector (1 downto 0);
          qout: buffer std_logic_vector (7 downto 0) );
end shifter;

architecture behave of shifter is
signal enable: std_logic;
begin
    process
    begin
        wait until (rising_edge(clk));
        if (reset = '1') then
            qout <= "00000000";
        else
            case mode is
                when "01" => qout <= shift_right & qout(7 downto 1);
                when "10" => qout <= qout(6 downto 0) & shift_left;
                when "11" => qout <= data; -- parallel load
                when others => null; -- null means do nothing
            end case;
        end if;
    end process;
end behave;
```

CD275

@W: Component declarations with different initial values are not supported. Port %n of component %n may have been given a different initial value in two different component declarations

Component declarations for the same entity that have different initial values are not supported. When this message is displayed, check whether initial values given on the ports are the same.

For example, in the two component declarations below for zbuf you can see that one is given an initial value for in1 and the other is not given an initial value:

```
component zbuf
port (in1: in std_logic := 1;
      out1: out std_logic
    );
end component;

component zbuf
port (in1: in std_logic;
      out1: out std_logic
    );
end component;
```

Action

The warning can be eliminated by ensuring both declarations have the same initial value:

```
component zbuf
port (in1: in std_logic := 1;
      out1: out std_logic
    );
end component;

component zbuf
port (in1: in std_logic := 1;
      out1: out std_logic
    );
end component;
```

If the warning continues to be displayed, remove the initialization values during port declaration if it is not required in the design.

If the same component is declared multiple times in the design for making different entities, make a package as shown below and use it in the design:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
package mycomps is
    component zbuf
        port (in1: in std_logic;
              out1: out std_logic
        );
    end component;
end package mycomps;
```

Then, add a use clause where the component is needed:

```
use work.mycomps.all;
```

CD276

@W: Map for port <s> of component <ha> not found

A component declaration does not include a port of the corresponding entity. In the test case below, port s of entity ha is not declared in the component declaration which results in the above warning.

```
entity ha is
    port (a,b: in bit;
          s: out bit;
          c: out bit);
end ha;

architecture beh of ha is
begin
    s<=a xor b;
    c<=a and b;
end beh;

entity test is
    port (a_top,b_top: in bit;
          --s_top: out bit;
          c_top: out bit);
end ;
```

```
architecture str of test is
component ha is
    port (a,b: in bit;
          c: out bit);
end component;

begin
    u1: ha port map (a=>a_top,b=>b_top,c=>c_top);
end str;
```

Action

Make sure that all of the ports of the corresponding entity are declared as shown in the corrected test case below.

```
entity ha is
    port (a,b: in bit;
          s: out bit;
          c: out bit);
end ha;

architecture beh of ha is
begin
    s<=a xor b;
    c<=a and b;
end beh;

entity test is
    port (a_top,b_top: in bit;
          s_top: out bit;
          c_top: out bit);
end ;

architecture str of test is
component ha is
    port (a,b: in bit;
          s: out bit;
          c: out bit);
end component;

begin
    u1: ha port map (a=>a_top,b=>b_top,s=>s_top,c=>c_top);
end str;
```

CD277

@W: Port direction mismatch between component and entity

The directions of ports in the component and the entity to which the ports are tied do not match. To get the intended results, make sure that the number of ports, port direction, port type, and port width are exactly the same for the component as for the corresponding entity. The above warning appears in the following test case because ports a and b in the component declaration are defined as output ports, but declared as inputs in the entity addern.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity addern is
    port (a,b: in std_logic_vector (7 downto 0);
          result: out std_logic_vector(7 downto 0));
end addern;

architecture behave of addern is
begin
    result <= a + b;
end;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity adders is
    port (a,b: in std_logic_vector(7 downto 0);
          result: out std_logic_vector(7 downto 0));
end adders;
architecture behave of adders is
component addern is
    port (a,b: out std_logic_vector (7 downto 0);
          result: out std_logic_vector(7 downto 0));
end component;

begin
I1:addern
    port map (a, b,result);
end behave;
```

Action

Make sure that the number of ports, port direction, port type, and port width are exactly the same for the component and the corresponding entity. To eliminate the warning in the above test case, edit the lines so that the component declaration port interface type and direction are the same as the type and direction of the entity as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity addern is
    port (a,b: in std_logic_vector (7 downto 0);
          result: out std_logic_vector(7 downto 0));
end addern;

architecture behave of addern is
begin
    result <= a + b;
end;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity adders is
    port (a,b: in std_logic_vector(7 downto 0);
          result: out std_logic_vector(7 downto 0));
end adders;
architecture behave of adders is
component addern is
    port (a,b: in std_logic_vector (7 downto 0);
          result: out std_logic_vector(7 downto 0));
end component;

begin
I1:addern
    port map (a, b,result);
end behave;
```

CD278

@W: Port type mismatch between component and entity

The data type of ports in the component as well as the entity to which they are tied do not match. To obtain the intended results, make sure that the number of ports, port direction, port type, and port width are exactly the same as they are for the entity and the architecture. The above warning appears in the following test case because ports `a` and `b` in the component declaration are defined as `bit_vectors`, but they are defined as `std_logic_vectors` in the `addern` entity.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity addern is
    port (a,b: in std_logic_vector (7 downto 0);
          result: out std_logic_vector(7 downto 0));
end addern;

architecture behave of addern is
begin
    result <= a + b;
end;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity adders is
    port (a,b: in std_logic_vector(7 downto 0);
          result: out std_logic_vector(7 downto 0));
end adders;

architecture behave of adders is
component addern is
    port (a,b: in bit_vector (7 downto 0);
          result: out std_logic_vector(7 downto 0));
end component;

```

```
begin
I1: addern
    port map (a, b,result);
end behave;
```

Action

Make sure that the number of ports, port direction, port type, and port width are exactly the same for both the entity and the architecture. To eliminate the warning in the above test case, replace `bit_vector` with `std_logic_vector` in the port definition for the `addern` component to make the component declaration port interface type and direction the same as the type and direction of the entity.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity addern is
    port (a,b: in std_logic_vector (7 downto 0);
          result: out std_logic_vector(7 downto 0));
end addern;

architecture behave of addern is
begin
    result <= a + b;
end;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity adders is
    port (a,b: in std_logic_vector(7 downto 0);
          result: out std_logic_vector(7 downto 0));
end adders;

architecture behave of adders is
component addern is
    port (a,b: in std_logic_vector (7 downto 0);
          result: out std_logic_vector(7 downto 0));
end component;
```

```

begin
I1:addern
    port map (a, b,result);
end behave;

```

CD279

@W: Port <c> of component <addern> not found on corresponding entity

The number of ports defined in the component does not match the number of ports defined in the entity to which it is tied. To obtain the intended results, make sure that the number of ports, port direction, port type, and port width are exactly the same for the component and the associated entity. The above warning appears in the following test case because port c is present in the component declaration, but not present in the entity addern to which it is tied.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity addern is
    port (a,b: in std_logic_vector (7 downto 0);
          result: out std_logic_vector(7 downto 0));
end addern;

architecture behave of addern is
begin
    result <= a + b;
end;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity adders is
    port (a,b: in std_logic_vector(7 downto 0);
          result: out std_logic_vector(7 downto 0));
end adders;

```

```
architecture behave of adders is
component addern is
    port (a,b,c: in std_logic_vector (7 downto 0);
          result: out std_logic_vector(7 downto 0));
end component;

begin
I1:addern
    port map (a,b,result);
end behave;
```

Action

Make sure that the number of ports, port direction, port type, and port width are exactly the same for the component and the corresponding entity. To eliminate the warning in the above test case, edit the lines so that the component declaration port interface type and direction agree as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity addern is
    port (a,b: in std_logic_vector (7 downto 0);
          result: out std_logic_vector(7 downto 0));
end addern;

architecture behave of addern is
begin
    result <= a + b;
end;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity adders is
    port (a,b: in std_logic_vector(7 downto 0);
          result: out std_logic_vector(7 downto 0));
end adders;
```

```

architecture behave of adders is
component addern is
    port (a,b: in std_logic_vector (7 downto 0);
          result: out std_logic_vector(7 downto 0));
end component;

begin
I1:addern
    port map (a,b,result);
end behave;

```

CD280

@W: Unbound component <addern> mapped to black box

A component has been found that is declared and instantiated, but does not map to an existing entity. The above warning appears in the following test case because addern is a component that is declared and instantiated, but is not tied to an entity.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity adders is
    port (a,b: in std_logic_vector(7 downto 0);
          result: out std_logic_vector(7 downto 0));
end adders;

architecture behave of adders is
component addern is
    port (a,b: in std_logic_vector (7 downto 0);
          result: out std_logic_vector(7 downto 0));
end component;

begin
I1:addern
    port map (a,b,result);
end behave;

```

Action

Make sure that all instantiations in the HDL code map to corresponding entities and architecture definitions. To eliminate the warning in the above test case, add a description of the entity to the project file to prevent the compiler from inferring a black box as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity adders is
    port (a,b: in std_logic_vector(7 downto 0);
          result: out std_logic_vector(7 downto 0));
end adders;

architecture behave of adders is
component addern is
    port (a,b: in std_logic_vector (7 downto 0);
          result: out std_logic_vector(7 downto 0));
end component;

begin
I1:addern
    port map (a,b,result);
end behave;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity addern is
    port (a,b: in std_logic_vector (7 downto 0);
          result: out std_logic_vector(7 downto 0));
end addern;
architecture behave of addern is
begin
    result <= a + b;
end;
```

Specify a Component as a Black Box Using syn_black_box

If the component is intended to be a black box, use the synthesis directive `syn_black_box` on the architecture, component, or label of the component. In the above test case, if the component `addern` is intended to be a black box, add this synthesis directive as follows:

```
architecture behave of adders is
component addern is
  port (a,b: in std_logic_vector (7 downto 0);
        result: out std_logic_vector(7 downto 0));
end component;

attribute syn_black_box : boolean;
attribute syn_black_box of addern: component is true;

begin
```

CD284

@W: syn_keep from a buffer port of a component is not supported - ignoring attribute

A `syn_keep` attribute is applied to a wire on a component buffer port (wires on component ports are not supported). In the test case below, component `test` is instantiated in entity `top`, and a `syn_keep` attribute is applied on intermediate wire `c1`. Because the output of component `test` is a buffer which feeds `c1`, the compiler ignores the `syn_keep` attribute and issues the above warning.

```
entity test is
  port (A: in bit_vector(1 downto 0 );
        B: in bit_vector(1 downto 0 );
        C: buffer bit_vector(1 downto 0 ));
end test;

architecture rtl of test is
begin
  c <= a and b ;
end rtl;
```

```

entity top is
    port (A: in bit_vector(1 downto 0 );
          B: in bit_vector(1 downto 0 );
          C: out bit_vector(1 downto 0 ));
end top;

architecture rtl of top is
component test is
    port (A: in bit_vector(1 downto 0 );
          B: in bit_vector(1 downto 0 );
          C: buffer bit_vector(1 downto 0 ));
end component;

signal c1 : bit_vector(1 downto 0 );
attribute syn_keep:Boolean;
attribute syn_keep of c1: signal is TRUE;

begin
u1: test port map (a,b,c1);
    c<=c1;
end rtl;

```

Action

Because a syn_keep attribute on a buffer port of a component is not supported, you cannot apply this attribute even if the mode of the port is changed to inout (the compiler passes the attribute, but the mapper gives a warning that the net with the syn_keep attribute has multiple drivers). You are advised if the mode can be changed to output without affecting the functionality; if not, you must avoid using a syn_keep attribute on the buffer port of a component. In the modified test case below, the mode is changed to output.

```

entity test is
    port (A: in bit_vector(1 downto 0 );
          B: in bit_vector(1 downto 0 );
          C: out bit_vector(1 downto 0 ));
end test;

architecture rtl of test is
begin
    c <= a and b ;
end rtl;

```

```

entity top is
    port (A: in bit_vector(1 downto 0 );
          B: in bit_vector(1 downto 0 );
          C: out bit_vector(1 downto 0 ));
end top;

architecture rtl of top is
component test is
    port (A: in bit_vector(1 downto 0 );
          B: in bit_vector(1 downto 0 );
          C: out bit_vector(1 downto 0 ));
end component;

signal c1 : bit_vector(1 downto 0 );
attribute syn_keep:Boolean;
attribute syn_keep of c1: signal is TRUE;

begin
u1: test port map (a,b,c1);
    c1<=c1;
end rtl;

```

CD285

@W: Port map width mismatch (7=> 8) on port of component <addern>

A mismatch exists between the port width declared in the component and the corresponding entity to which it is tied. To obtain the intended results, make sure that the number of ports, port direction, port type, and port width are exactly the same for the component and the corresponding architecture. The above warning appears in the following test case because port a in the component declaration is 7 bits wide, but in the corresponding entity declaration, it is 8 bits wide.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

```

```
entity addern is
port (a,b: in std_logic_vector (7 downto 0);
      result: out std_logic_vector(7 downto 0));
end addern;

architecture behave of addern is
begin
    result <= a + b;
end;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity adders is
    port (a,b: in std_logic_vector(7 downto 0);
          result: out std_logic_vector(7 downto 0));
end adders;

architecture behave of adders is
component addern is
    port (a: in std_logic_vector (6 downto 0);
          b: in std_logic_vector (7 downto 0);
          result: out std_logic_vector(7 downto 0) );
end component;

begin
I1:addern
    port map (a,b,result);
end behave;
```

Action

Make sure that the number of ports, port direction, port type, and port width are exactly the same for the entity and the architecture. To eliminate the warning in the above test case, change the port width for `a` as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
```

```
entity addern is
  port (a,b: in std_logic_vector (7 downto 0);
        result: out std_logic_vector(7 downto 0));
end addern;

architecture behave of addern is
begin
  result <= a + b;
end;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity adders is
  port (a,b: in std_logic_vector(7 downto 0);
        result: out std_logic_vector(7 downto 0));
end adders;

architecture behave of adders is
component addern is
  port (a: in std_logic_vector (7 downto 0);
        b: in std_logic_vector (7 downto 0);
        result: out std_logic_vector(7 downto 0) );
end component;

begin
I1:addern
  port map (a,b,result);
end behave;
```

CD286

@W: Creating black box for empty architecture <*addern*>

An empty architecture is detected, that is, no logic is present in the architecture definition of the entity. The following test case reports the above warning because the content of the architecture of the adder is empty (the content is commented out).

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity addern is
    port (a,b: in std_logic_vector (7 downto 0);
          result: out std_logic_vector(7 downto 0));
end addern;

architecture behave of addern is
begin
-- result <= a + b;
end;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity adders is
    port (a,b: in std_logic_vector(7 downto 0);
          result: out std_logic_vector(7 downto 0));
end adders;

architecture behave of adders is
component addern is
port (a,b: in std_logic_vector (7 downto 0);
      result: out std_logic_vector(7 downto 0));
end component;
begin
I1:addern
    port map (a => a, b=>b,result=>result);
end behave;

configuration adders_binding of adders is
    for behave
        for I1 : addern use entity work.addern(behave );
            end for;
    end for;
end adders_binding;
```

Action

Make sure that the black box architectures are designated as black boxes by using the `syn_black_box` synthesis directive. To eliminate this warning in the in the above test case, edit the `addern` architecture statement as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity addern is
    port (a,b: in std_logic_vector (7 downto 0);
          result: out std_logic_vector(7 downto 0));
end addern;

architecture behave of addern is
attribute syn_black_box : Boolean;
attribute syn_black_box of behave : architecture is true;
begin
    result <= a + b;
end;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity adders is
    port (a,b: in std_logic_vector(7 downto 0);
          result: out std_logic_vector(7 downto 0));
end adders;

architecture behave of adders is
component addern is
port (a,b: in std_logic_vector (7 downto 0);
      result: out std_logic_vector(7 downto 0));
end component;
begin
I1:addern
    port map (a => a, b=>b,result=>result);
end behave;
```

```
configuration adders_binding of adders is
    for behave
        for I1 : addern use entity work.addern(behave );
            end for;
        end for;
    end adders_binding;
```

CD289

@E: Expecting constant expression

The compiler is expecting a static constant value in places such as the initial value of a for loop and branches of a case statement, but instead encounters a dynamic value that is based on the input. In the following test case, the for-generate statement depends on test (an input) for the starting value of the loop.

```
library ieee;
use ieee.std_logic_1164.all;
entity adder is
    port (a, b, cin: in std_logic;
          sum, cout: out std_logic );
end adder;

architecture behave of adder is
begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or (cin and a) or (cin and b);
end behave;

--Instantiating the 1-bit adder many times with a generate
library ieee;
use ieee.std_logic_1164.all;

entity addern is
    generic (n: integer := 8);
    port (a, b: in std_logic_vector(n downto 1);
          cin: in std_logic;
          test: in integer := 1;
          sum: out std_logic_vector(n downto 1);
          cout: out std_logic );
end addern;
```

```

architecture structural of addern is
-- The component declaration goes here.
-- This allows you to instantiate the adder.
component adder
    port (a, b, cin: in std_logic;
          sum, cout: out std_logic);
end component;

signal carry: std_logic_vector(0 to n);
begin

-- Instantiate a single-bit adder n times
gen: for i in test to n generate
    add_onebit: adder port map
        (a => a(i),
         b => b(i),
         cin => carry(i - 1),
         sum => sum(i),
         cout => carry(i)
        );
end generate;
carry(0) <= cin;
cout <= carry(n);
end structural;
```

Action

This is a synthesis error that occurs because the corresponding hardware with the correct functionality cannot be created. To avoid this error, make sure that all loops used in a design have definite bounds (upper and lower). The corrected test case below shows an initial value for the integer rather than the constant value that causes the test case to error out. To eliminate this error in the above test case, use a constant (1) in the generate statement.

```

library ieee;
use ieee.std_logic_1164.all;
entity adder is
    port (a, b, cin: in std_logic;
          sum, cout: out std_logic );
end adder;

architecture behave of adder is
begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or (cin and a) or (cin and b);
```

```
end behave;
--Instantiating the 1-bit adder many times with a generate
library ieee;
use ieee.std_logic_1164.all;

entity addern is
    generic (n: integer := 8);
    port (a, b: in std_logic_vector(n downto 1);
          cin: in std_logic;
          test: in integer := 1;
          sum: out std_logic_vector(n downto 1);
          cout: out std_logic );
end addern;

architecture structural of addern is
-- The component declaration goes here.
-- This allows you to instantiate the adder.
component adder
    port (a, b, cin: in std_logic;
          sum, cout: out std_logic);
end component;

    signal carry: std_logic_vector(0 to n);
begin

-- Instantiate a single-bit adder n times
gen: for i in 1 to n generate
    add_onebit: adder port map
        (a => a(i),
         b => b(i),
         cin => carry(i - 1),
         sum => sum(i),
         cout => carry(i)
        );
end generate;
carry(0) <= cin;
cout <= carry(n);
end structural;
```

CD293

@E: Width mismatch, expecting width <3>, got <4>

The formal and the actual differ in their widths. In the test case below, formal `a_in(2 downto 0)` in the component instantiation statement has a width of 3 bits and actual `a_in` has a width of 4 bits which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity and1 is
    port (a_in, b_in: in bit_vector(3 downto 0);
          c_out:out bit_vector(3 downto 0) );
end and1;

architecture str of and1 is
begin
    c_out <= a_in and b_in;
end str;

library ieee;
use ieee.std_logic_1164.all;
use work.and1.all;

entity test is
    port (clk: in bit;
          a_in_top,b_in_top: in bit_vector(3 downto 0);
          d_out_top: out bit_vector(3 downto 0) );
end test;

architecture beh of test is
signal temp:bit_vector(3 downto 0);
begin
u1:entity work.and1 port map
    (a_in(3)=>not a_in_top(3), a_in(2 downto 0)=>a_in_top,
     b_in=>b_in_top, c_out=>temp);
process (clk)
begin
    if (clk'event and clk='1') then
        d_out_top <=temp;
    end if;
end process;
end beh;
```

Action

Make sure that the same width is specified for both the formal and the actual. As shown in the corrected test case below, the widths of formal `a_in(2 downto 0)` and actual `a_in(2 downto 0)` are the same.

```
library ieee;
use ieee.std_logic_1164.all;

entity and1 is
    port (a_in, b_in: in bit_vector(3 downto 0);
          c_out:out bit_vector(3 downto 0) );
end and1;

architecture str of and1 is
begin
    c_out <= a_in and b_in;
end str;

library ieee;
use ieee.std_logic_1164.all;
use work.and1.all;

entity test is
    port (clk: in bit;
          a_in_top,b_in_top: in bit_vector(3 downto 0);
          d_out_top: out bit_vector(3 downto 0) );
end test;

architecture beh of test is
signal temp:bit_vector(3 downto 0);
begin
    u1:entity work.and1 port map
        (a_in(3)=>not a_in_top(3),
         a_in(2 downto 0)=>a_in_top(2 downto 0),
         b_in=>b_in_top, c_out=>temp);
    process (clk)
    begin
        if (clk'event and clk='1') then
            d_out_top <=temp;
        end if;
    end process;
end beh;
```

CD297

@E: Width mismatch, location has width <2>, value <4>

There is a mismatch in the width of the arguments on both sides of an expression. In the following test case, q is defined as a 2-bit vector, however, the result of the concatenation of the expression is 4 which results in the message.

```
library ieee;
use ieee.std_logic_1164.all;

entity dff1 is
    port (data1: in std_logic;
          clk: in std_logic;
          data2: in std_logic;
          q: out std_logic_vector(1 downto 0) );
end dff1;

architecture async_set_reset of dff1 is
begin
    setreset: process
    begin
        wait until rising_edge(clk);
        q <= data1 & "11" & data2;
    end process setreset;
end async_set_reset;
```

Action

Make sure that the width of the location on the left side is the same as that expected by the operation on the right side. To eliminate the error in the above test case, change the definition of q as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity dff1 is
    port (data1: in std_logic;
          clk: in std_logic;
          data2: in std_logic;
          q: out std_logic_vector(1 downto 0) );
end dff1;
```

```
architecture async_set_reset of dff1 is
begin
    setreset: process
    begin
        wait until rising_edge(clk);
        q <= data1 & data2;
    end process setreset;
end async_set_reset;
```

Note that here the concatenation of the right side of the expression is two bits wide, the same size as q.

CD298

@E: Attempt to call function <pow> without a defined body

A function call was encountered that was not fully defined. In the following test case, the function Pow has only a function declaration, but no function body, which results in the message.

```
package specialfunctions is
function Pow (N,Exp : integer) return integer;
end specialfunctions;

-- 4 bit up-counter with load and asynchronous low reset

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;
use work.specialfunctions.all;

entity powerof is
    port (clk: in std_logic;
          input: in unsigned (3 downto 0);
          power: out unsigned (15 downto 0) );
end powerof;
```

```

architecture behave of powerof is
signal inputValInt : integer range 0 to 15;
signal powerL : integer range 0 to 65535;
begin
  inputValInt <= to_integer (input);
  power <= to_unsigned (powerL,16);
  process
    begin
      wait until CLk='1';
      powerL <= Pow (inputValInt, 4);
    end process;
  end behave;

```

Action

Make sure that the function body is written up for the function that is called in the RTL code. Functions and the function body can be declared either in the package or in the declarative part of the architecture. The function declaration and the function body must be visible before the function call. If the function and function body are defined in a package, make sure that the library-use clause is used to make them visible for the entity calling the function. For the above test case, add the function body in the package specialfunctions as shown below.

```

package specialfunctions is
  function Pow (N,Exp : integer) return integer;
end specialfunctions;

package body specialFunctions is
  function Pow (N,Exp : integer) return integer is
    variable Result : integer := 1;
  begin
    for I in 1 to Exp loop
      Result := Result * N;
    end loop;
    return (Result);
  end Pow;
end specialFunctions;

```

CD299

@E: Function has branches that can return values of different widths - - all branches must return a value having the same width

A function has two or more branches that return values of different widths. Function constructs can have one or more branches, and a function call can give different outputs according to the arguments it receives. Because the target of a function call always has the same width, all return statements must always return values of the same width.

In the following test case, function my_and has three branches with the first and second branches returning values of the same width. The return value from the third branch (the else portion) is truncated which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (clk: in std_logic;
          a_in,b_in: in std_logic_vector(2 downto 0);
          d_out: out std_logic_vector(2 downto 0) );
end test;

architecture beh of test is
function my_and (a,b: std_logic_vector) return std_logic_vector is
variable temp: std_logic_vector(a'range);
begin
    if(b(0)='1') then
        temp:= not a;
        return temp;
    elsif(b(1)='1')then
        temp:= a;
        return temp;
    elsif(b(2)='1')then
        temp:="000";
        return temp;
    else
        temp:="111";
        return temp(1 downto 0);
    end if;
end my_and;
```

```

begin
process (clk)
begin
  if falling_edge(clk) then
    d_out <= my_and(a_in,b_in);
  end if;
end process;
end beh;
```

Action

Make sure that functions with multiple branches always return values of the same width. In the following test case, function `my_and` has three branches that all return values of same width which corrects the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
  port (clk: in std_logic;
        a_in,b_in: in std_logic_vector(2 downto 0);
        d_out: out std_logic_vector(2 downto 0) );
end test;

architecture beh of test is
function my_and (a,b: std_logic_vector) return std_logic_vector is
  variable temp: std_logic_vector(a'range);
begin
  if(b(0)='1') then
    temp:= not a;
    return temp;
  elsif(b(1)='1')then
    temp:= a;
    return temp;
  elsif(b(2)='1')then
    temp:="000";
    return temp;
  else
    temp:="111";
    return temp;
  end if;
end my_and;
```

```
begin
    process (clk)
    begin
        if falling_edge(clk) then
            d_out <= my_and(a_in,b_in);
        end if;
    end process;
end beh;
```

CD300

@E: Call to procedure without body

Procedures were called in the HDL code without the procedure body being defined. In the following test case, the package common_apps contains only the declaration of procedure sort2, but does not contain the procedure body. This procedure is called in the VHDL code in the following lines:

```
sort2(va, vc); sort2(vb, vd);
sort2(va, vb); sort2(vc, vd);
sort2(vb, vc);
```

Example of the VHDL code:

```
package common_apps is
procedure sort2(x, y : inout bit_vector(0 to 7) );
end common_apps;

library work;
use work.common_apps.all;

entity sort4 is
    port (a, b, c, d: in bit_vector(0 to 7);
          ra, rb, rc, rd: out bit_vector(0 to 7) );
end sort4;

architecture muxes of sort4 is
begin
    process (a, b, c, d)
variable va, vb, vc, vd : bit_vector(0 to 7); begin
    va := a; vb := b; vc := c; vd := d;
    sort2(va, vc); sort2(vb, vd);
```

```

        sort2(va, vb); sort2(vc, vd);
        sort2(vb, vc);
        ra <= va; rb <= vb; rc <= vc; rd <= vd;
    end process;
end muxes;

```

Action

Make sure that the procedure body is defined and is visible before it is called in the HDL code. Procedures can be defined in the declaration portion of the architecture or in a package. Make sure that if the procedure declaration and the body are declared in a package, that the package is visible to the entity by using the library -use clause for the entity calling the package. To eliminate the error in the above test case, change the package common_apps as shown in the corrected test case below.

```

package body common_apps is
    procedure sort2(x, y : inout bit_vector(0 to 7) ) is
        variable tmp : bit_vector(0 to 7);
    begin
        if x > y then
            tmp := x;
            x := y;
            y := tmp;
        end if;
    end sort2;
end common_apps;

library work;
use work.common_apps.all;

entity sort4 is
    port (a, b, c, d: in bit_vector(0 to 7);
          ra, rb, rc, rd: out bit_vector(0 to 7) );
end sort4;

architecture muxes of sort4 is
begin
    process (a, b, c, d)
        variable va, vb, vc, vd : bit_vector(0 to 7); begin
            va := a; vb := b; vc := c; vd := d;
            sort2(va, vc); sort2(vb, vd);

```

```
        sort2(va, vb); sort2(vc, vd);
        sort2(vb, vc);
        ra <= va; rb <= vb; rc <= vc; rd <= vd;
    end process;
end muxes;
```

CD301

@E: This cast hasn't been implemented yet

A type casting was encountered that was not predefined by the VHDL language or by a user-defined function. The VHDL language allows restricted type casting (i.e., explicitly converting values between closely related types). In the test case below, using type cast `rec_ptr`, which has no predefined or user-defined function, causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic_vector(1 downto 0));
end comb;

architecture rtl of comb is
TYPE color IS (red, yellow, blue, white);
TYPE rec_ptr IS
RECORD
    d1 : color;
    d2 : std_logic;
END RECORD;

signal rp :rec_ptr;
signal rp1: rec_ptr;
begin
    rp <= (d1=> red, d2 => A);
    rp1<= rec_ptr (rp);
    C <= (B, rp1.d2);
end rtl;
```

Action

Be sure to use type castings that are predefined or use an appropriate user-defined conversion function. In the test case below, removing type cast `rec_ptr` corrects the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic_vector(1 downto 0));
end comb;

architecture rtl of comb is
TYPE color IS (red, yellow, blue, white);
TYPE rec_ptr IS
RECORD
    d1 : color;
    d2 : std_logic;
END RECORD;

signal rp :rec_ptr;
signal rp1: rec_ptr;
begin
    rp <= (d1=> red, d2 => A);
    rp1<= rp;
    C <= (B, rp1.d2);
end rtl;
```

CD308

@E: Unable to evaluate expression type

The value of a condition for an If block did not evaluate to a Boolean. In the test case below, the condition for the if block compares signal `clk`, which is of type `std_logic`, with the Boolean value 1 which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity seq is
    port (q: out std_logic_vector(7 downto 0);
          d : in std_logic_vector(7 downto 0);
          clk : in std_logic);
end seq;

architecture test of seq is
begin
    process(clk)
    begin
        if(clk'event and clk=1) then
            q<=d;
        end if;
    end process;
end test;
```

Action

Make sure that the expression used for a condition check always evaluates to a Boolean value. In the above test case, comparing signal clk with the std_logic value 1 corrects the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (q: out std_logic_vector(7 downto 0);
          d : in std_logic_vector(7 downto 0);
          clk : in std_logic);
end seq;

architecture test of seq is
begin
    process(clk)
    begin
        if(clk'event and clk= '1') then
            q<=d;
        end if;
    end process;
end test;
```

CD309

@E: Arguments have widths <3> and <2>, they should match

The widths of the arguments in a VHDL operation did not match. In the following test case, the and operation shows argument a with two bits and argument b with three bits which results in the message.

```
library ieee;
use ieee.std_logic_1164.all;

entity and2 is
    port (a: in std_logic_vector (1 downto 0);
          b: in std_logic_vector (2 downto 0);
          c: out std_logic_vector (1 downto 0) );
end;

architecture rtl of and2 is
begin
    c (1 downto 0) <= a and b;
end;
```

Action

Make sure that arguments in any VHDL operation are the same size. The result of such an operation is always deterministic. To eliminate the error in the above test case, make sure that the values for arguments a and b are the same as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity and2 is
    port (a: in std_logic_vector (1 downto 0);
          b: in std_logic_vector (1 downto 0);
          c: out std_logic_vector (1 downto 0) );
end;

architecture rtl of and2 is
begin
    c (1 downto 0) <= a and b;
end;
```

CD312

@E: Slice range direction does not match argument range

A vector was declared in a particular direction, but was accessed in a different direction. In the test case below, port a is declared as a descending vector (1 downto 0), but is used as an ascending vector (0 to 1) which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity and2 is
    port (a: in std_logic_vector (1 downto 0);
          c: out std_logic_vector (1 downto 0));
end;

architecture rtl of and2 is
begin
    c (1 downto 0) <= a(0 to 1);
end;
```

Action

Make sure that vectors and slices of vectors are accessed in the same direction as declared. To eliminate the error in the above test case, either change the declaration of a to match how it is accessed as shown in the first test case or change the way it is accessed as shown in the second test case.

Changing the argument range direction

```
library ieee;
use ieee.std_logic_1164.all;

entity and2 is
    port (a: in std_logic_vector (0 to 1);
          c: out std_logic_vector (1 downto 0));
end;

architecture rtl of and2 is
begin
    c (1 downto 0) <= a(0 to 1);
end;
```

Changing the assignment range direction

```

library ieee;
use ieee.std_logic_1164.all;

entity and2 is
    port (a: in std_logic_vector (1 downto 0);
          c: out std_logic_vector (1 downto 0));
end;

architecture rtl of and2 is
begin
    c (1 downto 0) <= a(1 downto 0);
end;

```

CD314

@E: Condition in if generate must evaluate to a constant value

The condition evaluated in an if-generate statement was not a constant value (either true or false). In the case below, the generate statement depends on the value of test which is defined as an input.

```

library ieee;
use ieee.std_logic_1164.all;

entity counter4 is
    port (count, clock: in std_logic;
          test: in integer := 0;
          q: inout std_logic_vector (0 to 3) );
end counter4;

architecture if_generate of counter4 is
component d_flip_flop
    port (D,CLK: in std_logic;
          que: out std_logic);
end component;

begin
    GK0 : if k = test generate
        DFF : D_FLIP_FLOP port map (count, clock, q(k) );
    end generate GK0;

```

```
GK1_3 : if k > 0 generate
    DFF: D_FLIP_FLOP port map (q(k-1), clock, q(k) );
end generate GK1_3;
end generate GK;
end if_generate;
```

Action

Make sure that the condition in the if-generate statement evaluates to a constant value of being true or false. To eliminate this error in the above test case, change the value in the if-generate statement from test to 0.

```
library ieee;
use ieee.std_logic_1164.all;

entity counter4 is
    port (count, clock: in std_logic;
          test: in integer := 0;
          q: inout std_logic_vector (0 to 3) );
end counter4;

architecture if_generate of counter4 is
component d_flip_flop
    port (D,CLK: in std_logic;
          que: out std_logic);
end component;

begin
    GK0 : if k = 0 generate
        DFF : D_FLIP_FLOP port map (count, clock, q(k) );
    end generate GK0;

    GK1_3 : if k > 0 generate
        DFF: D_FLIP_FLOP port map (q(k-1), clock, q(k) );
    end generate GK1_3;
end generate GK;
end if_generate;
```

In the above corrected test case, test is an input to the design which is assigned an initial value of 0, and not a constant value.

CD316

@E: Case choice is not locally static

A case statement was encountered in which the case branch/choice was not given a definite value at the time of compilation. In the following test case, the case branch depends on the input of the entity and therefore is not static.

```
library ieee;
use ieee.std_logic_1164.all;

entity mux is
    port (output_signal: out std_logic;
          remain: in std_logic;
          in1, in2, in3, in4: in std_logic;
          sel: in std_logic_vector(1 downto 0);
          hold: in std_logic_vector (1 downto 0) );
end mux;

architecture behave of mux is
begin
    process (in1, in2, in3, in4, sel)
    begin
        case sel is
            when "00" =>
                output_signal <= in1;
            when "01" =>
                output_signal <= in2;
            when "10" =>
                output_signal <= in3;
            when "11" =>
                output_signal <= in4;
            when hold =>
                output_signal <= remain;
            when others =>
                output_signal <= 'X';
        end case;
    end process;
end behave;
```

Action

This is a synthesis error that occurs when the appropriate hardware having the described functionality cannot be created. To avoid this error, make sure that the case choices have a static value at the time of synthesis. To eliminate the error in the above test case, comment out the when hold branch (hold is an input) as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity mux is
    port (output_signal: out std_logic;
          remain: in std_logic;
          in1, in2, in3, in4: in std_logic;
          sel: in std_logic_vector(1 downto 0);
          hold: in std_logic_vector (1 downto 0) );
end mux;

architecture behave of mux is
begin
    process (in1, in2, in3, in4, sel)
    begin
        case sel is
            when "00" =>
                output_signal <= in1;
            when "01" =>
                output_signal <= in2;
            when "10" =>
                output_signal <= in3;
            when "11" =>
                output_signal <= in4;
            -- when hold =>
            --     output_signal <= remain;
            when others =>
                output_signal <= 'X';
        end case;
    end process;
end behave;
```

CD320

@E: Port formal <a_in> out of range for bits <0> to <4>

The range of the formal is outside the range limits of the corresponding port when using a named association for mapping. In the test case below, the range of formal `a_in` is specified as 0 to 4 which is outside the 0 to 3 range limit of port `a_in` of entity `and1` which causes the error.

```

entity and1 is
    port (a_in, b_in: in bit_vector(0 to 3);
          c_out:out bit_vector (3 downto 0) );
end and1;

architecture str of and1 is
begin
    c_out <= a_in and b_in;
end str;

use work.and1.all;
entity top is
    port (clk: in bit;
          a_in,b_in: in bit_vector(3 downto 0);
          d_out: out bit_vector(3 downto 0) );
end top;

architecture beh of top is
component and1 is
    port (a_in, b_in: in bit_vector(0 to 3);
          c_out: out bit_vector(3 downto 0) );
end component;

signal temp:bit_vector(3 downto 0);
begin
    u1: and1 port map(a_in(0 to 4)=>a_in,b_in=>b_in,c_out=>temp);
    process (clk)
    begin
        if (clk'event and clk='1') then
            d_out <=temp ;
        end if;
    end process;
end beh;

```

Action

Make sure that the range of the formal lies within the range of the corresponding port and that it also matches the width of the actual. For the above test case, changing the range of formal `a_in` to 0 to 3 to match both the port and corresponding actual corrects the error.

```
entity and1 is
  port (a_in, b_in: in bit_vector(0 to 3);
        c_out:out bit_vector (3 downto 0) );
end and1;

architecture str of and1 is
begin
  c_out <= a_in and b_in;
end str;

use work.and1.all;

entity top is
  port (clk: in bit;
        a_in,b_in: in bit_vector(3 downto 0);
        d_out: out bit_vector(3 downto 0) );
end top;

architecture beh of top is
component and1 is
  port (a_in, b_in: in bit_vector(0 to 3);
        c_out: out bit_vector(3 downto 0) );
end component;

signal temp:bit_vector(3 downto 0);
begin
  u1: and1 port map(a_in(0 to 3)=>a_in,b_in=>b_in,c_out=>temp);
  process (clk)
  begin
    if (clk'event and clk='1') then
      d_out <=temp ;
    end if;
  end process;
end beh;
```

CD321

@E: Port formal <a_in> out of range for bits <4> downto <1>

The range of the formal is outside the range limits of the corresponding port when using a named association for mapping. In the test case below, the range of formal a_in is specified as 4 downto 1 which is outside of range limit 3 downto 0 of the corresponding port a_in of entity and1 which causes the error.

```

entity and1 is
    port (a_in, b_in: in bit_vector(3 downto 0);
          c_out:out bit_vector(3 downto 0) );
end and1;

architecture str of and1 is
begin
    c_out <= a_in and b_in;
end str;

use work.and1.all;

entity test1 is
    port (clk: in bit;
          a_in,b_in: in bit_vector(3 downto 0);
          d_out: out bit_vector(3 downto 0) );

end test1;

architecture beh of test1 is
component and1 is
    port (a_in, b_in: in bit_vector(3 downto 0);
          c_out: out bit_vector(3 downto 0) );
end component;

signal temp:bit_vector(3 downto 0);
begin
    ul: and1 port map(a_in(4 downto 1)=>a_in,b_in=>b_in,c_out=>temp);
    process (clk)
    begin
        if (clk'event and clk='1') then
            d_out <=temp;
        end if;
    end process;
end beh;

```

Action

Make sure that the range of the formal lies within the range of the corresponding port and that it also matches the width of the actual. For the above test case, changing the range of formal `a_in` to 3 downto 0 to match both the port and corresponding actual corrects the error.

```
entity and1 is
    port (a_in, b_in: in bit_vector(3 downto 0);
          c_out:out bit_vector(3 downto 0) );
end and1;

architecture str of and1 is
begin
    c_out <= a_in and b_in;
end str;

use work.and1.all;

entity test1 is
    port (clk: in bit;
          a_in,b_in: in bit_vector(3 downto 0);
          d_out: out bit_vector(3 downto 0) );
end test1;

architecture beh of test1 is
component and1 is
    port (a_in, b_in: in bit_vector(3 downto 0);
          c_out: out bit_vector(3 downto 0) );
end component;

signal temp:bit_vector(3 downto 0);
begin
u1: and1 port map(a_in(3 downto 0)=>a_in,b_in=>b_in,c_out=>temp);
    process (clk)
    begin
        if (clk'event and clk='1') then
            d_out <=temp;
        end if;
    end process;
end beh;
```

CD322

@E: Port formal <a_in> out of range for bit <4>

The range of the formal is outside the range limits of the corresponding port when using a named association for mapping. In the test case below, index 4 of formal a_in(4) is out of range of the corresponding range limits 3 downto 0 of port a_in of entity and1 which causes the error.

```

entity and1 is
  port (a_in, b_in: in bit_vector(3 downto 0);
        c_out:out bit_vector(3 downto 0) );
end and1;

architecture str of and1 is
begin
  c_out <= a_in and b_in;
end str;

entity test2 is
  port (clk: in bit;
        a_in,b_in: in bit_vector(3 downto 0);
        d_out: out bit_vector(3 downto 0) );
end test2;

architecture beh of test2 is
component and1 is
  port (a_in, b_in: in bit_vector(3 downto 0);
        c_out: out bit_vector(3 downto 0) );
end component;

signal temp:bit_vector(3 downto 0);
begin
  u1: and1 port map(a_in(4)=>a_in(3),
                     a_in(2 downto 0)=>a_in(2 downto
                     0),b_in=>b_in,c_out=>temp);
  process (clk)
  begin
    if (clk'event and clk='1') then
      d_out <=temp;
    end if;
  end process;
end beh;

```

Action

Make sure that the indices of the formal lie within the range of the corresponding port. In the above test case, changing the index of formal `a_in` to 3 corrects the error.

```
entity and1 is
    port (a_in, b_in: in bit_vector(3 downto 0);
          c_out:out bit_vector(3 downto 0) );
end and1;

architecture str of and1 is
begin
    c_out <= a_in and b_in;
end str;

entity test2 is
    port (clk: in bit;
          a_in,b_in: in bit_vector(3 downto 0);
          d_out: out bit_vector(3 downto 0) );
end test2;

architecture beh of test2 is
component and1 is
    port (a_in, b_in: in bit_vector(3 downto 0);
          c_out: out bit_vector(3 downto 0) );
end component;

signal temp:bit_vector(3 downto 0);
begin
    u1: and1 port map(a_in(3)=>a_in(3),
                        a_in(2 downto 0)=>a_in(2 downto 0),b_in=>b_in,c_out=>temp);
    process (clk)
    begin
        if (clk'event and clk='1') then
            d_out <=temp;
        end if;
    end process;
end beh;
```

CD323

@E: Association for port <a_in> has already been specified?

A formal port had more than one named association. In the test case below, when instantiating component and1, formal a_in(3) is mapped to both not a_in(3) and a_in(3 downto 0) which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity and1 is
    port (a_in, b_in: in std_logic_vector(3 downto 0);
          c_out:out std_logic_vector(3 downto 0) );
end and1;

architecture str of and1 is
begin
    c_out <= a_in and b_in;
end str;

library ieee;
use ieee.std_logic_1164.all;
use work.and1.all;

entity seq is
    port (clk: in std_logic;
          a_in,b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0) );
end seq;

architecture beh of seq is
component and1 is
    port (a_in, b_in: in std_logic_vector(3 downto 0);
          c_out: out std_logic_vector(3 downto 0) );
end component;

signal temp:std_logic_vector(3 downto 0);
begin
    u1: and1 port map(a_in(3)=>not a_in(3),
                       a_in(3 downto 0)=>a_in(2 downto
0),b_in=>b_in,c_out=>temp);
    process (clk)
    begin
```

```
        if falling_edge(clk) then
            d_out <=temp;
        end if;
    end process;
end beh;
```

Action

Make sure that all formals are mapped only once as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity and1 is
    port (a_in, b_in: in std_logic_vector(3 downto 0);
          c_out:out std_logic_vector(3 downto 0) );
end and1;

architecture str of and1 is
begin
    c_out <= a_in and b_in;
end str;

library ieee;
use ieee.std_logic_1164.all;
use work.and1.all;

entity seq is
    port (clk: in std_logic;
          a_in,b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0) );
end seq;

architecture beh of seq is
component and1 is
    port (a_in, b_in: in std_logic_vector(3 downto 0);
          c_out: out std_logic_vector(3 downto 0) );
end component;

signal temp:std_logic_vector(3 downto 0);
begin
ul: and1 port map(a_in(3)=>not a_in(3),
                   a_in(2 downto 0)=>a_in(2 downto 0), b_in=>b_in,c_out=>temp);
process (clk)
begin
```

```
        if falling_edge(clk) then
            d_out <=temp;
        end if;
    end process;
end beh;
```

CD326

@E: Port <a> of entity <work>. <error1> is unconnected. If a port needs to remain unconnected, use the keyword open.

Note: This error may appear as a warning (@W) or as an error (@E) depending on the context.

A port was not connected during instantiation. The following test case results in the above error message because port A is not connected.

```
library ieee;
use ieee.std_logic_1164.all;

entity error1 is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic );
end error1;

architecture rtl of error1 is
begin
    C <= A and B;
end rtl;

library ieee;
use ieee.std_logic_1164.all;

entity error1top is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic );
end error1top;
```

```
architecture rtl of error1top is
component error1
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic );
end component;

begin
inst1: error1 port map(B => B,C => C);
end rtl;
```

Action

Make sure that all ports in an instantiation are connected. If a port needs to remain unconnected, use the keyword open. To eliminate this error in the above test case, connect port A as shown in one of the corrected test cases below.

Connecting port A

```
library ieee;
use ieee.std_logic_1164.all;

entity error1 is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic );
end error1;

architecture rtl of error1 is
begin
    c <= A and B;
end rtl;

library ieee;
use ieee.std_logic_1164.all;

entity error1top is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic );
end error1top;
```

```
architecture rtl of error1top is
component error1
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic );
end component;

begin
inst1: error1 port map(A => A, B => B,C => C);
end rtl;
```

Making port A unconnected

```
library ieee;
use ieee.std_logic_1164.all;

entity error1 is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic );
end error1;

architecture rtl of error1 is
begin
    C <= A and B;
end rtl;

library ieee;
use ieee.std_logic_1164.all;

entity error1top is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic );
end error1top;

architecture rtl of error1top is
component error1
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic );
end component;

begin
inst1: error1 port map(A => open, B => B,C => C);
end rtl;
```

CD327

@E: Illegal connection of instance input <data> to entity output <q>

The binding of the ports of the instance to the higher level ports in the design did not match. In instantiations, input ports should be connected to inputs up in the hierarchy, and output ports to outputs; inout ports allow either inputs or outputs to be connected to them. In the test case below, the above error occurs due to the incorrect mapping of the input port data in the instance (formal) to the output port q (actual).

```
library ieee;
use ieee.std_logic_1164.all;

entity reg8 is
    port (q: out std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst: in std_logic );
end reg8;

architecture reg8_design of reg8 is -- eight bit register
begin
    process (clk, rst)
    begin
        if rst = '1' then
            q <= X"00";
        elsif rising_edge(clk) then
            q <= data;
        end if;
    end process;
end reg8_design;

library ieee;
use ieee.std_logic_1164.all;

entity top_level is
    port (q: out std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst: in std_logic );
end top_level;

architecture structural of top_level is
```

```

component reg8 -- component declaration for reg8
    port (q: out std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst: in std_logic);
end component;

-- declare the internal signals here
signal mux_out, reg_out: std_logic_vector (7 downto 0);

begin -- structural description begins
inst3: reg8
    port map (clk => clk,data => q,
              q => q, rst=> rst);
end structural;

```

Action

Make sure that ports are connected correctly within the hierarchy. Upper-level inputs are connected to inputs of lower-level entities, upper-level outputs are connected to outputs of lower-level entities, and upper-level inout can be connected to either inputs or outputs of the lower-level entities. To correct this error in the above test case, correct the mapping of **inst3: reg8** as shown in the corrected test case below.

```

library ieee;
use ieee.std_logic_1164.all;

entity reg8 is
    port (q: out std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst: in std_logic );
end reg8;

architecture reg8_design of reg8 is -- eight bit register
begin
    process (clk, rst)
    begin
        if rst = '1' then
            q <= X"00";
        elsif rising_edge(clk) then
            q <= data;
        end if;
    end process;
end reg8_design;

library ieee;
use ieee.std_logic_1164.all;

```

```
entity top_level is
    port (q: out std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst: in std_logic );
end top_level;

architecture structural of top_level is

component reg8 -- component declaration for reg8
    port (q: out std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst: in std_logic);
end component;

-- declare the internal signals here
signal mux_out, reg_out: std_logic_vector (7 downto 0);

begin -- structural description begins
inst3: reg8
    port map (clk => clk,data => data,
              q => q, rst=> rst);
end structural;
```

CD329

@E: Illegal connection of instance output <q> to entity input/constant <data>

The binding of the ports of the instance to higher-level ports in the design did not match. In instantiations, input ports should be connected to inputs up in the hierarchy, and output ports to outputs; inout ports allow either inputs or outputs to be connected to them. In the test case below, the above error occurs due to the incorrect mapping of instance (formal) output port q to the input data (actual). The same error can occur if a constant value is tied to output port q.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity reg8 is
    port (q: out std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst: in std_logic);
end reg8;

architecture reg8_design of reg8 is -- eight bit register
begin
    process (clk, rst)
    begin
        if rst = '1' then
            q <= X"00";
        elsif rising_edge(clk) then
            q <= data;
        end if;
    end process;
end reg8_design;

library ieee;
use ieee.std_logic_1164.all;

entity top_level is
    port (q: out std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst: in std_logic);
end top_level;

architecture structural of top_level is

component reg8 -- component declaration for reg8
    port (q: out std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst: in std_logic);
end component;

-- declare the internal signals here

begin -- structural description begins
inst3: reg8
    port map (clk => clk,data => data,
              q => data, rst=> rst);
end structural;
```

Action

Make sure that ports are connected correctly within the hierarchy. Upper-level inputs are connected to inputs of lower-level entities, and upper-level outputs are connected to outputs of lower-level entities; upper-level inouts can be connected to either the inputs or outputs of the lower-level entities. To eliminate this error in the above test case, correct the mapping of inst3: reg8 as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity reg8 is
    port (q: out std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst: in std_logic);
end reg8;

architecture reg8_design of reg8 is -- eight bit register
begin
    process (clk, rst)
    begin
        if rst = '1' then
            q <= X"00";
        elsif rising_edge(clk) then
            q <= data;
        end if;
    end process;
end reg8_design;

library ieee;
use ieee.std_logic_1164.all;

entity top_level is
    port (q: out std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst: in std_logic);
end top_level;

architecture structural of top_level is

component reg8 -- component declaration for reg8
    port (q: out std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst: in std_logic);
end component;

-- declare the internal signals here
```

```

begin -- structural description begins
inst3: reg8
    port map (clk => clk,data => data,
               q => q, rst=> rst);
end structural;

```

CD330

@E: Port <q> cannot be connected to a constant

An output port was connected to a constant in an instantiation. Only input ports can be connected to constants in an instantiation. If you need output ports connected to a constant, assign the constant in the architecture of the entity.

The following test case causes the error. The test case consists of two files. The first file is the entity and architecture for reg8. The second file is the top-level design which instantiates reg8.

```

library ieee;
use ieee.std_logic_1164.all;

entity reg8 is
    port (q: out std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst: in std_logic);
end reg8;

architecture reg8_design of reg8 is -- eight bit register
begin
    process (clk, rst)
    begin
        if rst = '1' then
            q <= X"00";
        elsif rising_edge(clk) then
            q <= data;
        end if;
    end process;
end reg8_design;

library ieee;
use ieee.std_logic_1164.all;

```

```
entity top_level is
    port (q: out std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst: in std_logic);
end top_level;

architecture structural of top_level is
component reg8 -- component declaration for reg8
    port (q: out std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst: in std_logic);
end component;

begin -- structural description begins
inst3: reg8
    port map (clk => clk,data => data,
              q => "10011111", rst => rst);
end structural;
```

Action

Make sure that constants are assigned only to input ports. To eliminate the error in the above test case, replace the q definition in the ifrst statement and edit the port map definition as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity reg8 is
    port (q: out std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst: in std_logic);
end reg8;

architecture reg8_design of reg8 is -- eight bit register
begin
    process (clk, rst)
    begin
        if rst = '1' then
            q <= "10001111";
        elsif rising_edge(clk) then
            q <= data;
        end if;
    end process;
end architecture;
```

```

        end process;
    end reg8_design;

library ieee;
use ieee.std_logic_1164.all;

entity top_level is
    port (q: out std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst: in std_logic);
end top_level;

architecture structural of top_level is
component reg8 -- component declaration for reg8
    port (q: out std_logic_vector (7 downto 0);
          data: in std_logic_vector (7 downto 0);
          clk, rst: in std_logic);
end component;

begin -- structural description begins
inst3: reg8
    port map (clk => clk,data => data,
              q => q, rst => rst);
end structural;

```

CD332

@E: Maximum design hierarchy component instantiation depth exceeded: <251>

The component instantiation depth exceeded 250. In the test case below, entity a1 continually instantiates itself. When the instantiation depth exceeds 250, the compiler errors out.

```

entity a1 is
    port (a,b: in bit;
          c:out bit);
end a1;

architecture str of a1 is
begin
    u1:entity work.a1(str) port map (a,b,c);
end str;

```

Action

Make sure that the instantiation depth does not exceed the 250 limit.

CD333

@E: No Entities found in input!

There was no entity declaration in any of the VHD files included in the project. The test case represented below is the only input file included in the project. This file has only package without any top-level entity which causes the error .

```
package my_pack is
  subtype my_type is integer range 14 to integer'high;
end my_pack;
```

Action

Make sure that the project file has at least one input (VHD) file with an entity declaration. In the corrected test case below, there entity seq is declared.

```
package my_pack is
  subtype my_type is integer range 14 to integer'high;
end my_pack;

library ieee;
use ieee.std_logic_1164.all;
use work.my_pack.all;

entity seq is
  generic (k: my_type:=2);
    port (q: out std_logic_vector(k-1 downto 0);
          clk : in std_logic;
          d : in std_logic_vector(k-1 downto 0));
end seq;
```

```
architecture test of seq is
begin
  process(clk)
  begin
    if(clk'event and clk='1') then
      q<=d;
    end if;
  end process;
end test;
```

CD334

Note: This error may appear as a warning (@W) or as an error (@E) depending on the context.

@E: No architectures associated with entity <and2>

An architecture was not specified for a design entity. In the following test case, design entity and2 has no associated architecture which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity and2 is
  port ( a: in std_logic;
         b: in std_logic;
         c: out std_logic );
end and2;
```

Action

Specify an associated architecture for every design entity. To eliminate the error in the above test case, add an associated architecture.

```
library ieee;
use ieee.std_logic_1164.all;
```

```

entity and2 is
    port (a: in std_logic;
          b: in std_logic;
          c: out std_logic );
end and2;

architecture rtl of and2 is
begin
    c <= a and b; -- Or other user define logic
end;

```

CD335

@E: Multiple clocks controlling signal - not supported

A signal was controlled by multiple clocks (with the exception of a dual-port RAM, no sequential element can have more than one clock source). In the test case below, signal qrs is driven by two clocks (clk and load) which causes the compiler to error out with the above message.

```

library ieee;
use ieee.std_logic_1164.all;

entity dff1 is
    port (load, clk, d0,d1: in std_logic;
          qrs: out std_logic );
end dff1;

architecture async_set_reset of dff1 is
begin
    setreset: process (clk,load)
    begin
        if rising_edge( clk ) and rising_edge (load ) then
            if load = '1' then
                qrs <= d0;
            else
                qrs <= d1;
            end if;
        end if;
    end process setreset;
end async_set_reset;

```

Action

Avoid using multiple clocks for driving a signal. In the corrected test case below, signal qrs is clocked only by clk.

```

library ieee;
use ieee.std_logic_1164.all;

entity dff1 is
    port (load, clk, d0,d1: in std_logic;
          qrs: out std_logic );
end dff1;

architecture async_set_reset of dff1 is
signal temp_set,temp_rst : std_logic;
begin
temp_set <= load and d0;
temp_rst<= load and (not d0);
setreset: process (clk,temp_rst,temp_set)
begin
    if rising_edge( clk )then
        if (temp_rst = '1') then
            qrs <= '0';
        elsif (temp_set='1') then
            qrs <='1' ;
        else
            qrs <= d1;
        end if;
    end if;
end process setreset;
end async_set_reset;
```

CD336

@E: Could not find return statement

A function failed to return a value after execution. The absence of a return statement in the function body results in the following error. The following test case shows a function where the result is not returned.

```

library ieee;
use ieee.std_logic_1164.all;
```

```

entity test is
  port (
    clk: in std_logic;
    a_in, b_in: in std_logic_vector(3 downto 0);
    d_out: out std_logic_vector(3 downto 0)
  );
end test;

architecture beh of test is
function my_and (a, b: std_logic_vector) return std_logic_vector
  is variable temp : std_logic_vector(a'range);
begin
  temp := a and b;
end function;

begin
  process (clk)
begin
  if rising_edge(clk) then
    d_out <= my_and(a_in, b_in);
  end if;
end process;
end beh;

```

Action

Make sure that all functions in a design return a value after execution. To eliminate the error in the above testcase, add a return statement.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
  port (
    clk: in std_logic;
    a_in, b_in: in std_logic_vector(3 downto 0);
    d_out: out std_logic_vector(3 downto 0)
  );
end test;

architecture beh of test is
function my_and (a, b: std_logic_vector) return std_logic_vector
  is variable temp: std_logic_vector(a'range);
begin
  temp:= a and b;
  return temp;
end function;

```

```

begin
  process (clk)
  begin
    if rising_edge(clk) then
      d_out <= my_and(a_in, b_in);
    end if;
  end process;
end beh;

```

CD340

@E: mod operator only supported when both operands are unsigned

The operands of the mod operator were signed. When a mod operator is used, the operands (and result) must be unsigned. In the test case below, the left and right operands of the mod operator are declared as signed which causes the error.

```

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

entity comb is
  port (A: in signed(1 downto 0);
        B: in signed(1 downto 0);
        C: out signed(1 downto 0));
end comb;

architecture rtl of comb is
begin
  C<= A mod b;
end rtl;

```

Action

Make sure that the operands of a mod operator are declared as unsigned. To eliminate the error in the above test case, declare operands A and B (and result C) as unsigned as shown in the corrected test case below.

```

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

```

```
entity comb is
    port (A: in unsigned(1 downto 0);
          B: in unsigned(1 downto 0);
          C: out unsigned(1 downto 0) );
end comb;

architecture rtl of comb is
begin
    c<= a mod b;
end rtl;
```

CD341

@E: Slice is out of range

The bounds of a slice are outside the range defined for the vector. When a slice of a vector is referenced in VHDL, the range of the slice must be either the same as that defined for the vector or a subset of that range. Shown below is a test case that results in the above error message.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (
        clk: in std_logic;
        d_in: in std_logic_vector(7 downto 0);
        d_out: out std_logic_vector(0 to 3)
    );
end test;

architecture beh of test is
begin
    process (clk)
    begin
        if rising_edge(clk) then
            d_out <= d_in (8 downto 5); --# 8 is out of
                                         range for d_in
        end if;
    end process;
end beh;
```

Action

Make sure that the slice range is within the bounds of the defined vector range. To eliminate this error in the above test case, edit the range for the slice as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
port (
    clk: in std_logic;
    d_in: in std_logic_vector(7 downto 0);
    d_out: out std_logic_vector(0 to 3)
);
end test;

architecture beh of test is
begin
    process (clk)
    begin
        if rising_edge(clk) then
            d_out <= d_in (7 downto 4);
        end if;
    end process;
end beh;
```

CD342

@E: Reversed Range!

Vectors were not referenced using the declared range direction. Vectors can be specified using either ascending or descending range. Defined vectors must be referenced using the same range direction. In the test case below, the range for vector a is reversed which causes the error.

```
module error (
    input clk,
    input [0:3] a, // a defined using ascending range
    input [3:0] b,
    output reg [3:0] q );
```

```
always@(posedge clk)
begin
    q <= a[3:0] & b[3:0]; // a has a descending range
end
endmodule
```

Action

Make sure that the vector range direction is in the same as that specified when the vector was declared as shown in the corrected test case below.

```
module test (
    input clk,
    input [0:3] a,
    input [3:0] b,
    output reg [3:0] q);

    always@(posedge clk)
    begin
        q <= a[0:3] & b[3:0];
    end
endmodule
```

CD343

@E: 'stable is only supported for single bit signals

A ‘stable attribute was applied on a vector array signal. The VHDL attribute ‘stable is used to determine if a signal has changed and can only be applied on single-bit signals when used to describe a register. The following test case uses a 2-bit vector as a clock with the ‘stable attribute being used on the complete vector which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (
```

```

clk: in std_logic_vector(1 downto 0);
a_in: in std_logic_vector(3 downto 0);
d_out: out std_logic_vector(3 downto 0)
);
end test;

architecture beh of test is
begin
process (clk)
begin
if (not clk'stable and clk = "11") then
    d_out <= a_in;
end if;
end process;
end beh;

```

Action

The correct use of the 'stable' attribute is shown below where it is applied on a single-bit clock signal.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
port (
    clk: in std_logic;
    a_in: in std_logic_vector(3 downto 0);
    d_out: out std_logic_vector(3 downto 0)
);
end test;

architecture beh of test is
begin
process (clk)
begin
if (not clk'stable and clk = '1') then
    d_out <= a_in;
end if;
end process;
end beh;

```

You can use the vector as a single-bit signal as shown below.

```
if (not clk(1)'stable and clk(1) = '1') then
```

CD344

@E: 'last_value is only supported for single bit signals

A 'last_value attribute was applied on a vector array signal. The VHDL attribute 'last_value indicates the previous value of a single-bit signal and is used in conjunction with 'event to indicate an edge. The following code uses a multi-bit vector as a clock with the 'last_value attribute being used on the complete vector which results in the above error.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (
        clk: in std_logic_vector(2 downto 0);
        a_in: in std_logic_vector(3 downto 0);
        d_out: out std_logic_vector(3 downto 0)
    );
end test;

architecture beh of test is
signal temp : integer;
begin
    process (clk)
    begin
        if (clk = "11" and clk'last_value = "00") then
            d_out <= a_in;
        end if;
    end process;
end beh;
```

Action

The corrected test case below shows how the 'last_value attribute is used correctly on a single-bit signal to infer a register in the synthesis tool.

```
library ieee;
use ieee.std_logic_1164.all;
```

```

entity test is
port (
    clk: in std_logic;
    a_in: in std_logic_vector(3 downto 0);
    d_out: out std_logic_vector(3 downto 0)
);
end test;

architecture beh of test is
signal temp : integer;
begin
    process (clk)
    begin
        if (clk = '1' and clk'last_value = '0') then
            d_out <= a_in;
        end if;
    end process;
end beh;

```

The user can use the vector as a single bit signal as shown below.

```
if (clk(1)'last_value and clk(1) = '1') then
```

CD345

@E: 'event is only supported for single bit signals

The 'event attribute is commonly used when describing clock edges in VHDL by checking if a signal is at a particular value and if the signal has recently changed. The 'event attribute can only be used on single-bit signals. The use of 'event on a vector array as shown in the following test case causes the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
port (
    clk: in std_logic_vector(1 downto 0);
    a_in, b_in: in std_logic_vector(3 downto 0);
    d_out: out std_logic_vector(3 downto 0)
);
end test;

```

```
architecture beh of test is
signal temp : integer;
begin
    process (clk)
    begin
        if (clk'event and clk = "11") then
            d_out <= a_in;
        end if;
    end process;
end beh;
```

Action

The corrected test case below illustrates the proper use of the 'event expression on a single-bit clock signal to infer a register.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
port (
    clk: in std_logic;
    a_in: in std_logic_vector(3 downto 0);
    d_out: out std_logic_vector(3 downto 0)
);
end test;

architecture beh of test is
signal temp : integer;
begin
    process (clk)
    begin
        if (clk'event and clk = '1') then
            d_out <= a_in;
        end if;
    end process;
end beh;
```

You can use the vector as a single-bit signal as shown below.

```
if (clk(1)'event and clk(1) = '1') then
```

CD351

@E: Can't implement expression type <ufcall> yet

An expression failed to return an expected type. For example, if the expression used to define a vector range contains an operand of type `real`, the compiler would error out with the above message. In this case, the expression must evaluate to an integer to be valid in this context. In the following test case, `num_bits`, which defines the port ranges, is defined as type `real` which results in the error.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity my_adder is
  generic (num_bits : real := 8.0);
  port (a,b: in std_logic_vector (8*num_bits -1 downto 0);
        cin: in std_logic;
        cout: out std_logic;
        result: out std_logic_vector(8*num_bits -1 downto 0) );
end my_adder;

architecture behave of my_adder is
component adder8
  port (a,b: in std_logic_vector (7 downto 0);
        cin: in std_logic;
        cout: out std_logic;
        sum: out std_logic_vector(7 downto 0) );
end component;
signal c: std_logic_vector(num_bits downto 0);
begin
  c(0) <= cin;
  generate_adder:
  for i in 0 to num_bits-1 generate
    U : adder8
    port map (
      a => a(8*i+7 downto 8*i),
      b => b(8*i+7 downto 8*i),
      cin => c(i),
      cout => c(i+1),
    );
  end generate;
end;

```

```

        sum => result(8*i+7 downto 8*i)
    );
end generate generate_adder;
cout <= c(num_bits);
end;

```

Action

To eliminate the error in the above test case, define the generic num_bits as an integer as shown in the corrected test case below.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity my_adder is
generic (num_bits : integer := 8);
port (a,b: in std_logic_vector (8*num_bits -1 downto 0);
      cin: in std_logic;
      cout: out std_logic;
      result: out std_logic_vector(8*num_bits -1 downto 0) );
end my_adder;

architecture behave of my_adder is
component adder8
port (a,b: in std_logic_vector (7 downto 0);
      cin: in std_logic;
      cout: out std_logic;
      sum: out std_logic_vector(7 downto 0) );
end component;

signal c: std_logic_vector(num_bits downto 0);
begin
  c(0) <= cin;
generate_adder:
for i in 0 to num_bits-1 generate
  U : adder8
  port map (
    a => a(8*i+7 downto 8*i),
    b => b(8*i+7 downto 8*i),
    cin => c(i),
    cout => c(i+1),
    sum => result(8*i+7 downto 8*i)
  );
end generate generate_adder;

```

```
cout <= c(num_bits);
end;
```

CD352

@E: generation loop must be over int or enum type

Generate loops in VHDL can be evaluated over an integer range or a range of enumerated types. This error occurs if the type used to specify the loop range is anything other than that specified above. The following test case results in the error due to the use of std_logic_vector type in the loop range specification.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity my_adder is
generic (num_bits : integer := 8;
std_bits: std_logic_vector(3 downto 0) := "0111" );
port (a,b: in std_logic_vector (8*num_bits -1 downto 0);
      cin: in std_logic;
      cout: out std_logic;
      result: out std_logic_vector(8*num_bits -1 downto 0) );
end my_adder;

architecture behave of my_adder is
component adder8
port (a,b: in std_logic_vector (7 downto 0);
      cin: in std_logic;
      cout: out std_logic;
      sum: out std_logic_vector(7 downto 0) );
end component;

signal c: std_logic_vector(num_bits downto 0);
type COLOR is (RED, BLUE, GREEN, YELLOW, BROWN, BLACK);
begin
  c(0) <= cin;
```

```

generate_adder:
  for i in "0000" to std_bits generate
    U : adder8
      port map (
        a => a(8*i+7 downto 8*i),
        b => b(8*i+7 downto 8*i),
        cin => c(i),
        cout => c(i+1),
        sum => result(8*i+7 downto 8*i)
      );
  end generate generate_adder;

  cout <= c(num_bits);
end;

```

Action

Use an integer or enumerated type to specify the range of the generate loop statement. To eliminate the error in the above testcase, replace the std_logic_vector type in the loop range specification with an integer type as shown in the corrected test case below.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity my_adder is
generic (num_bits : integer := 8;
         std_bits: std_logic_vector(3 downto 0) := "0111" );
  port (a,b: in std_logic_vector (8*num_bits -1 downto 0);
        cin: in std_logic;
        cout: out std_logic;
        result: out std_logic_vector(8*num_bits -1 downto 0) );
end my_adder;

architecture behave of my_adder is
component adder8
  port (a,b: in std_logic_vector (7 downto 0);
        cin: in std_logic;
        cout: out std_logic;
        sum: out std_logic_vector(7 downto 0) );
end component;

```

```

signal c: std_logic_vector(num_bits downto 0);
type COLOR is (RED, BLUE, GREEN, YELLOW, BROWN, BLACK);
begin
    c(0) <= cin;

generate_adder:
    for i in 0 to num_bits-1 generate
        U : adder8
            port map (
                a => a(8*i+7 downto 8*i),
                b => b(8*i+7 downto 8*i),
                cin => c(i),
                cout => c(i+1),
                sum => result(8*i+7 downto 8*i)
            );
    end generate generate_adder;
    cout <= c(num_bits);
end;

```

CD353

@E: While loop is not terminating? You can set the maximum number of loop iterations with the syn_looplimit attribute -- attach it to the loop label

The while loop is unconstrained which causes the loop to be executed indefinitely as it does not have a clear exit condition. The following test case illustrates one such scenario where the execution of the while loop is determined by an input to the entity. Since the input value is not defined at compile time, the compiler errors out.

```

library ieee;
use ieee.std_logic_1164.all;

```

```

entity test is
  port (
    clk: in std_logic;
    D: in std_logic_vector(1 downto 0);
    a_in: in std_logic_vector(3 downto 0);
    d_out: out std_logic_vector(3 downto 0)
  );
end test;

architecture beh of test is
begin
  process (clk)
  variable i : integer := 0;
  begin
    if (clk'event and clk = '1') then
      while D = "10" loop
        d_out(i) <= a_in(i);
        i := i + 1;
      end loop;
    end if;
  end process;
end beh;

```

Action

Make sure that the loop limits can be evaluated to a constant at compile time by analyzing the design and recoding the loop statement based on the design requirements.

CD354

@E: for loops with unbound ranges should contain a wait statement

The range of a for loop could not be determined at compile time, for example, when the bounds of a for loop are determined by an input to the design. In this case, loop iteration cannot be determined at compile time which results in the above error.

```

library ieee;
use ieee.std_logic_1164.all;

```

```

entity test is
generic (width: integer := 3);
port (
    clk: in std_logic;
    D: in integer;
    a_in: in std_logic_vector(width downto 0);
    d_out: out std_logic_vector(width downto 0)
);
end test;

architecture beh of test is
signal temp : integer;
begin
process (clk)
variable i : integer := 0;
begin
    if (clk'event and clk = '1') then
        for j in 0 to D loop
            d_out(j) <= a_in(j);
        end loop;
    end if;
end process;
end beh;

```

Action

Specify loop iteration bounds that can be evaluated at compile time. To eliminate the error in the above test case, change the for loop statement as shown in the corrected test case below.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
generic (width : integer := 3);
port (
    clk: in std_logic;
    D: in integer;
    a_in: in std_logic_vector(width downto 0);
    d_out: out std_logic_vector(width downto 0)
);
end test;

```

```
architecture beh of test is
signal temp : integer;
begin
    process (clk)
        variable i : integer := 0;
    begin
        if (clk'event and clk = '1') then
            for j in 0 to d_out'high loop
                d_out(j) <= a_in(j);
            end loop;
        end if;
    end process;
end beh;
```

CD358

@E: No architecture is available for entity <ha>

The entity being instantiated had no architecture. In the test case below, entity HA, which is instantiated in the top-level entity adder, has no architecture declaration which causes the error.

```
entity HA is
    port (U,V: in bit;
          X,Y: out bit);
end HA;

entity adder is
    port (A, B: in bit;
          SUM, CARRY: out bit);
end adder;

architecture beh of adder is
begin
    u1: entity work.HA port map (A, B, SUM,CARRY);
end beh;
```

Action

Make sure that every instantiated entity has at least one architecture block. To eliminate the error in the above test case, define an architecture block for the entity HA as shown in the corrected test case below.

```

entity HA is
    port (U,V: in bit;
          X,Y: out bit);
end HA;

architecture str of ha is
begin
    X<= U xor V;
    Y<= U and V;
end str;

entity adder is
    port (A, B: in bit;
          SUM, CARRY: out bit);
end adder;

architecture beh of adder is
begin
    u1: entity work.HA port map (A, B, SUM,CARRY);
end beh;

```

CD360

@E: Generic <n> of entity <work.ha> is unspecified

A generic defined in the entity being instantiated was not assigned a value. Although you can have undefined generics in entities, a value must later be assigned in either the component instantiation or in the instantiation using a generic map. In the test case below, entity HA is instantiated in the top level and has a generic n which is not assigned a value. This missing value causes the error.

```

entity HA is
generic (n: integer);
    port (U,V: in bit_vector(n-1 downto 0);
          X,Y: out bit_vector(n-1 downto 0));
end HA;

architecture str of ha is
begin
    X<=U xor V;
    Y<=U and V;
end str;

```

```
entity adder is
    port (A, B: in bit_vector(3 downto 0);
          SUM, CARRY: out bit_vector(3 downto 0));
end adder;

architecture beh of adder is
component HA
generic (n: integer);
    port (U,V: in bit_vector(n-1 downto 0);
          X,Y: out bit_vector(n-1 downto 0));
end component;

begin
    u1: HA  port map(A, B, SUM,CARRY);
end beh;
```

Action

Make sure that the generic is declared with a constant value. Assign a default value to the declaration and change the values accordingly by using generic map statements. To eliminate the error in the above test case, either supply a default value or specify the value of the generic using generic map statements while instantiating the component HA as shown in the corrected test case below.

```
entity HA is
generic (n: integer);
    port (U,V: in bit_vector( n-1 downto 0);
          X,Y: out bit_vector( n-1 downto 0));
end HA;

architecture str of ha is
begin
    X<=U xor V;
    Y<=U and V;
end str;

entity adder is
    port (A, B: in bit_vector(3 downto 0);
          SUM, CARRY: out bit_vector(3 downto 0));
end adder;
```

```

architecture beh of adder is
component HA
generic (n: integer);
port (U,V: in bit_vector(n-1 downto 0);
      X,Y: out bit_vector(n-1 downto 0));
end component;

begin
  u1: HA generic map(4) port map(A, B, SUM,CARRY);
end beh;

```

CD364

@N: Removed redundant assignment.

The compiler encountered an assignment that assigns to itself. In the following test case, the assignment `qrs <= qrs` indicates that if the positive clock edge does not occur, the old value of `qrs` is retained. Since a flip-flop is created for `qrs`, this is always true and hence the assignment `qrs <= qrs` is implicit and not required.

```

library ieee;
use ieee.std_logic_1164.all;

entity dff2 is
port (data1: in bit_vector (2 downto 0);
      clk, reset, set: in bit;
      qrs_out: out bit_vector (2 downto 0) );
end dff2;

architecture sync_set_reset of dff2 is
signal qrs : bit_vector (2 downto 0);
begin
setreset:
process (clk)
begin
  if (clk'event and clk='1' ) then
    if reset = '1' then
      qrs <= "000";
    elsif set = '1' then
      qrs <= "111";
    else
      qrs <= data1;
    end if;
  end if;
end process;
end;

```

```
        end if;
else
    qrs <= qrs;
end if;
end process setreset;
qrs_out <= qrs;
end sync_set_reset;
```

Action

Assignments, like the one above, are implicit and do not need to be specified in the HDL code. The compiler writes out a note saying that it is removing a redundant statement as it is unnecessary. To eliminate the note in the above test case, comment out the **qrs** <= **qrs** statement as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity dff2 is
    port (data1: in bit_vector (2 downto 0);
          clk, reset, set: in bit;
          qrs_out: out bit_vector (2 downto 0) );
end dff2;

architecture sync_set_reset of dff2 is
signal qrs : bit_vector (2 downto 0);
begin
setreset:
    process (clk)
    begin
        if (clk'event and clk='1') then
            if reset = '1' then
                qrs <= "000";
            elsif set = '1' then
                qrs <= "111";
            else
                qrs <= data1;
            end if;
        --else
        --    qrs <= qrs;
        end if;
    end process setreset;
    qrs_out <= qrs;
end sync_set_reset;
```

CD367

@N: Instance <u1>, Port <c_out>, Bit <0> connection not specified

During instantiation of a component using named association, a bit of the output formal array port is unconnected. In the test case below, bit 0 of port c_out is unconnected when instantiating component and1 which results in the note.

```
library ieee;
use ieee.std_logic_1164.all;

entity and1 is
    port (a_in, b_in: in std_logic_vector(3 downto 0);
          c_out:out std_logic_vector (3 downto 0) );
end and1;

architecture str of and1 is
begin
    c_out <= a_in and b_in;
end str;

library ieee;
use ieee.std_logic_1164.all;
use work.and1.all;

entity seq is
    port (clk: in std_logic;
          a_in,b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0) );
end seq;

architecture beh of seq is
component and1 is
    port (a_in, b_in: in std_logic_vector(3 downto 0);
          c_out: out std_logic_vector(3 downto 0) );
end component;

signal temp:std_logic_vector(2 downto 0);
begin
u1: and1 port map(a_in=>a_in,b_in=>b_in,c_out(3 downto 1)=>temp);
process (clk)
begin
```

```
        if falling_edge(clk) then
            d_out <=temp & not temp(0);
        end if;
    end process;
end beh;
```

Action

Make sure that the connections for all bits of all ports are specified as shown in the test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity and1 is
    port (a_in, b_in: in std_logic_vector(3 downto 0);
          c_out:out std_logic_vector (3 downto 0) );
end and1;

architecture str of and1 is
begin
    c_out <= a_in and b_in;
end str;

library ieee;
use ieee.std_logic_1164.all;
use work.and1.all;

entity seq is
    port (clk: in std_logic;
          a_in,b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0) );
end seq;

architecture beh of seq is
component and1 is
    port (a_in, b_in: in std_logic_vector(3 downto 0);
          c_out: out std_logic_vector(3 downto 0) );
end component;

signal temp:std_logic_vector(2 downto 0);
signal temp1:std_logic;
begin
u1: and1 port map(a_in=>a_in,b_in=>b_in,c_out
                   (3 downto 1)=>temp,c_out(0)=>temp1);
process (clk)
```

```

begin
    if falling_edge(clk) then
        d_out <=temp & not templ;
    end if;
end process;
end beh;
```

CD370

@W: Named aggregate left holes in array, filling with type's default value

An array is assigned a value using a named aggregate statement, but some bit positions within the array are not given values and are left as holes. With unassigned values, the compiler fills in these holes with the default value of the array type and issues the warning. Because filling is done with the default type, an unexpected behavior may occur if this entity is used in some other block. In the test case below, array C is assigned a concatenation of a and b but, because bit 0 is not assigned a value, it is filled with the default type of array C.

```

library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic_vector(2 downto 0));
end comb;

architecture rtl of comb is
begin
    C<= (2=>a,1=> b);
end rtl;
```

Action

Be sure not to leave any holes in the named aggregate as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic_vector(2 downto 0));
end comb;

architecture rtl of comb is
begin
    C<= (2=>a,1=> b,0=>A);
end rtl;
```

CD371

@E: No matching overload for <*plusone*>

The design uses a function assignment without proper arguments, or uses an operator with an incorrect number or type argument. The following test case generates the above error.

```
library ieee;
use ieee.std_logic_1164.all;

entity error is
    port (
        A: in std_logic;
        B: in std_logic;
        C: out std_logic
    );
end error;

architecture rtl of error is
function plusone(S: std_logic) return std_logic is
begin
    return (s + 1);
end plusone;

begin
    C <= plusone(A,B);
end rtl;
```

Action

Make sure a design uses function assignments and operators with proper arguments. To eliminate the error in the above test case, correct the function arguments as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity error is
port (
    A: in std_logic;
    B: in std_logic;
    C: out std_logic
);
end error;

architecture rtl of error is
function plusone(S: std_logic) return std_logic is
begin
    return (s + 1);
end plusone;

begin
    C <= plusone(A);
end rtl;
```

Overloading can be done on functions as well as operators. Overloading allows different argument types to be associated with an operator or function. The operator or function used depends on the types of inputs, the number of inputs, and the result type. This is transparent to the user as long as a function or operator with those argument types is found. A call to an overloaded operator or function is ambiguous and is considered an error when it is not possible to identify the operator or function using the following information:

- operator or function name
- number of arguments
- type and order of arguments
- result type

Multiple Input Case

Using the addition operator (+) on two inputs of the same type (std_ulogic or std_ulogic_vector) and the IEEE STD_LOGIC_ARITH package can also result in the above error. The IEEE STD_LOGIC_ARITH package function definition for the + operator cannot accept two std_ulogic/std_ulogic_vector inputs.

Some simulation tools have defined overloading functions for the + operator that can take two std_ulogic/std_ulogic_vector inputs as its arguments. To make the synthesis tool compatible, Synopsys provides the cdn_arith.vhd file to redefine the STD_LOGIC_ARITH package to include all of the operator overloads. The cdn_arith.vhd file is included in the *installation/lib/vhd* directory; add this file to your project and use it in place of the IEEE standard STD_LOGIC_ARITH package as shown in the following code segment:

```
library ieee;
use ieee.std_logic_1164.all;
-- use ieee.std_logic_arith.all; -- Replace this library as
-- shown below
use work.std_logic_arith.all;

entity testcase is
    port (clk: in std_ulogic;
          count: out std_ulogic_vector (6 downto 0));
end testcase;

architecture rtl of testcase is
signal SIG: std_ulogic_vector (6 downto 0);
begin
    process (clk)
    begin
        if rising_edge (clk) then
            SIG <= SIG + "1010101";
        end if;
        count <= SIG;
    end process;
end rtl;
```

CD374

@E: choice <3> is out of range

The design assigned an element that is out of the range of the defined vector. The following test case generates the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity error4 is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic_vector(7 downto 0) );
end error4;

architecture rtl of error4 is
begin
    C (7 downto 4) <= (7 => A, 5 => B, 3 => '1');
end rtl;
```

Action

Correct the vector assignment statement. Make sure the assignment is done on the slices within the vector range. To eliminate the error in the above test case, change the vector assignment statement from (7 downto 4) to (7 downto 3).

```
library ieee;
use ieee.std_logic_1164.all;

entity error4 is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic_vector(7 downto 0) );
end error4;

architecture rtl of error4 is
begin
    C (7 downto 3) <= (7 => A, 5 => B, 3 => '1');
end rtl;
```

CD375

@E: Duplicate specification of choice <7>

The design assigned an element of a vector multiple times in the same statement. The following test case generates the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity error5 is
    port (A: in std_logic_vector(3 downto 0);
          B: in std_logic_vector(1 downto 0);
          C: out std_logic_vector(7 downto 0) );
end error5;

architecture rtl of error5 is
begin
    C (7 downto 4) <= (7 => '1', 7 => '0', 5 => '1' );
end rtl;
```

Action

Make sure that a bit is not assigned more than once in a slice. To eliminate the error in the above test case, change the vector assignment statement as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity error5 is
    port (A: in std_logic_vector(3 downto 0);
          B: in std_logic_vector(1 downto 0);
          C: out std_logic_vector(7 downto 0) );
end error5;

architecture rtl of error5 is
begin
    C (7 downto 4) <= (7 => '1', 5 => '1' );
end rtl;
```

CD379

@E: Specified wrong number of values in aggregate, expected <2>

The number of elements after concatenation did not match the width of the assigned vector. The following test case generates the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity error15 is
    port (A: in std_logic_vector(1 downto 0);
          B: in std_logic_vector(1 downto 0);
          C: out std_logic_vector(1 downto 0) );
end error15;

architecture rtl of error15 is
begin
    C <= (A(0), B(0), B(1));
end rtl;
```

Action

Make sure that the result of concatenation matches in width to the result vector width. To eliminate the error in the above test case, correct the number of elements in the aggregate to match the length of the resultant vector (C is a 2-bit vector) as shown below.

```
library ieee;
use ieee.std_logic_1164.all;

entity error15 is
    port (A: in std_logic_vector(1 downto 0);
          B: in std_logic_vector(1 downto 0);
          C: out std_logic_vector(1 downto 0) );
end error15;

architecture rtl of error15 is
begin
    C <= (A(0), B(0));
end rtl;
```

CD385

@E: aggregate with others must be in a constrained context

An others clause was used as one of the arguments in the concatenation operation. The following test case generates the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity error18 is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic_vector(7 downto 0) );
end error18;

architecture rtl of error18 is
begin
    C <= A & B & (others => '0');
end rtl;
```

Action

Use aggregate (slice of a vector) for such assignments instead of a concatenation operator as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity error18 is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic_vector(7 downto 0) );
end error18;

architecture rtl of error18 is
begin
    C <= (7 => A, 6 => B, others => '0');
end rtl;
```

CD387

@E: Positional members must precede named ones

Both implicit and explicit assignments are being used for assigning the record elements in a single statement. The following test case generates the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity error20 is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic_vector(1 downto 0));
end error20;

architecture rtl of error20 is
TYPE color IS (red, yellow, blue, white);
TYPE rec_ptr IS
RECORD
    d1 : color;
    d2 : std_logic;
END RECORD;
signal rp : rec_ptr;
begin
    rp <= (d1 => red, A);
    C <= (rp.d2, B);
end rtl;
```

Action

Use only implicit (positional) association or explicit (named) association. To eliminate the error in the above test case, replace the implicit and explicit assignment with one of the following:

Implicit Association

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity error20 is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic_vector(1 downto 0));
end error20;

architecture rtl of error20 is
TYPE color IS (red, yellow, blue, white);
TYPE rec_ptr IS
RECORD
    d1 : color;
    d2 : std_logic;
END RECORD;
signal rp : rec_ptr;
begin
    rp <= (red, A);
    C <= (rp.d2, B);
end rtl;
```

Explicit Association

```
library ieee;
use ieee.std_logic_1164.all;

entity error20 is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic_vector(1 downto 0));
end error20;

architecture rtl of error20 is
TYPE color IS (red, yellow, blue, white);
TYPE rec_ptr IS
RECORD
    d1 : color;
    d2 : std_logic;
END RECORD;
signal rp : rec_ptr;
begin
    rp <= (d1 => red, d2 => A);
    C <= (rp.d2, B);
end rtl;
```

Record types belong to a composite class consisting of elements of different types. The element declaration for each record declares one or more identifiers and their types. The following code segments illustrate record definition and assignment.

```
--declaration of identifiers and their types
type opcode is (add, sub, mult, div, and, or, nop);
type mode is range 0 to 3;
type address is std_logic_vector (7 downto 0);

--record declaration containing 3 fields of different types
type instruction_format is record
    op:opcode;
    md:mode;
    addr:address;
end record;

--signal of type instruction_format initialized
signal instr(instruction_format) := (nop, 0, "00000000");

--assignment of record fields
instr.op <= add;
instr.md <= 1;
instr.addr <= "00000001";

--complete record assignment using named association
instr <= (op => add, md => 1, addr => "00000001");

--complete record assignment using positional association
instr <= (add, 1, "00000001");
```

CHAPTER 9

CD Messages 388 – 504

CD388

@E: Too many members

A record was assigned with more elements than were defined. The following test case generates the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity error21 is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic_vector(1 downto 0) );
end error21;

architecture rtl of error21 is
TYPE color IS (red, yellow, blue, white);
TYPE rec_ptr IS
RECORD
    d1 : color;
    d2 : std_logic;
END RECORD;
```

```

signal rp : rec_ptr;
begin
    rp <= (red, A, '0');
    C <= (B, rp.d2);
end rtl;

```

Action

Assign the record with the correct number of elements. To eliminate the error in the above test case, change the number of elements in record rp as shown in the corrected test case below.

```

library ieee;
use ieee.std_logic_1164.all;

entity error21 is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic_vector(1 downto 0) );
end error21;

architecture rtl of error21 is
TYPE color IS (red, yellow, blue, white);
TYPE rec_ptr IS
RECORD
    d1 : color;
    d2 : std_logic;
END RECORD;
signal rp : rec_ptr;
begin
    rp <= (red, A);
    C <= (B, rp.d2);
end rtl;

```

Record types belong to a composite class consisting of elements of different types. The element declaration for each record declares one or more identifiers and their types. The following code segments illustrate record definition and assignment.

```

--declaration of identifiers and their types
type opcode is (add, sub, mult, div, and, or, nop);
type mode is range 0 to 3;
type address is std_logic_vector (7 downto 0);

```

```
--record declaration containing 3 fields of different types
type instruction_format is record
    op:opcode;
    md:mode;
    addr:address;
end record;

--signal of type instruction_format initialized
signal instr(instruction_format) := (nop, 0, "00000000");

--assignment of record fields
instr.op <= add;
instr.md <= 1;
instr.addr <= "00000001";

--complete record assignment using named association
instr <= (op => add, md => 1, addr => "00000001");

--complete record assignment using positional association
instr <= (add, 1, "00000001");
```

CD389

@E: No such tag <d3> in record

The design contained an assignment with an undefined record element. The following test case generates the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity error23 is
port (
    A: in std_logic;
    B: in std_logic;
    C: out std_logic_vector(1 downto 0));
end error23;

architecture rtl of error23 is
TYPE color IS (red, yellow, blue, white);
TYPE rec_ptr IS
RECORD
    d1 : color;
    d2 : std_logic;
```

```

END RECORD;
signal rp : rec_ptr;
begin
    rp <= (d3 => red, d2 => A);
    C <= (B, rp.d2);
end rtl;

```

Action

Use only defined record elements. To eliminate the error in the above test case, edit the record element name as shown in the corrected test case below.

```

library ieee;
use ieee.std_logic_1164.all;

entity error23 is
port (
    A: in std_logic;
    B: in std_logic;
    C: out std_logic_vector(1 downto 0));
end error23;

architecture rtl of error23 is
TYPE color IS (red, yellow, blue, white);
TYPE rec_ptr IS
RECORD
    d1 : color;
    d2 : std_logic;
END RECORD;
signal rp : rec_ptr;
begin
    rp <= (d1 => red, d2 => A);
    C <= (B, rp.d2);
end rtl;

```

Record types belong to a composite class consisting of elements of different types. The element declaration for each record declares one or more identifiers and their types. The following code segments illustrate record definition and assignment.

```

--declaration of identifiers and their types
type opcode is (add, sub, mult, div, and, or, nop);
type mode is range 0 to 3;
type address is std_logic_vector (7 downto 0);

```

```
--record declaration containing 3 fields of different types
type instruction_format is record
    op:opcode;
    md:mode;
    addr:address;
end record;

--signal of type instruction_format initialized
signal instr(instruction_format) := (nop, 0, "00000000");

--assignment of record fields
instr.op <= add;
instr.md <= 1;
instr.addr <= "00000001";

--complete record assignment using named association
instr <= (op => add, md => 1, addr => "00000001");

--complete record assignment using positional association
instr <= (add, 1, "00000001");
```

CD390

@E: Duplicate specification of value for field <d1>

A record element was assigned more than once. The following test case generates the above message.

```
library ieee;
use ieee.std_logic_1164.all;

entity error24 is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic_vector(1 downto 0) );
end error24;

architecture rtl of error24 is
TYPE color IS (red, yellow, blue, white);
TYPE rec_ptr IS
RECORD
    d1 : color;
    d2 : std_logic;
END RECORD;
```

```

signal rp : rec_ptr;
begin
    rp <= (d1 => red, d1 => blue, d2 => A);
    C <= (B, rp.d2);
end rtl;

```

Action

Correct the record assignment statement. To eliminate the error in the above test case, remove the duplicate record element assignment.

```

library ieee;
use ieee.std_logic_1164.all;

entity error24 is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic_vector(1 downto 0) );
end error24;

architecture rtl of error24 is
TYPE color IS (red, yellow, blue, white);
TYPE rec_ptr IS
RECORD
    d1 : color;
    d2 : std_logic;
END RECORD;
signal rp : rec_ptr;
begin
    rp <= (d1 => red, d2 => A);
    C <= (B, rp.d2);
end rtl;

```

Record types belong to a composite class consisting of elements of different types. The element declaration for each record declares one or more identifiers and their types. The following code segments illustrate record definition and assignment.

```

--declaration of identifiers and their types
type opcode is (add, sub, mult, div, and, or, nop);
type mode is range 0 to 3;
type address is std_logic_vector (7 downto 0);

```

```
--record declaration containing 3 fields of different types
type instruction_format is record
    op:opcode;
    md:mode;
    addr:address;
end record;

--signal of type instruction_format initialized
signal instr(instruction_format) := (nop, 0, "00000000");

--assignment of record fields
instr.op <= add;
instr.md <= 1;
instr.addr <= "00000001";

--complete record assignment using named association
instr <= (op => add, md => 1, addr => "00000001");

--complete record assignment using positional association
instr <= (add, 1, "00000001");
```

CD391

@E: No value specified for field <d1>

A partially assigned record failed to cover all of its fields. The following test case generates the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity error27 is
    port (
        A: in std_logic;
        B: in std_logic;
        C: out std_logic_vector(1 downto 0));
end error27;

architecture rtl of error27 is
TYPE color IS (red, yellow, blue, white);
TYPE rec_ptr IS
RECORD
    d1 : color;
    d2 : std_logic;
```

```

END RECORD;
signal rp : rec_ptr;
begin
    rp <= (d2 => A);
    C <= (B, rp.d2);
end rtl;

```

Action

Use explicit assignments when assigning a partial record. To eliminate the error in the above test case, correct the record element name as shown in the corrected test case below.

```

library ieee;
use ieee.std_logic_1164.all;

entity error27 is
port (
    A: in std_logic;
    B: in std_logic;
    C: out std_logic_vector(1 downto 0));
end error27;

architecture rtl of error27 is
TYPE color IS (red, yellow, blue, white);
TYPE rec_ptr IS
RECORD
    d1 : color;
    d2 : std_logic;
END RECORD;
signal rp : rec_ptr;
begin
    rp.d2 <= A;
    C <= (B, rp.d2);
end rtl;

```

Record types belong to a composite class consisting of elements of different types. The element declaration for each record declares one or more identifiers and their types. The following code segments illustrate record definition and assignment.

```

--declaration of identifiers and their types
type opcode is (add, sub, mult, div, and, or, nop);
type mode is range 0 to 3;
type address is std_logic_vector (7 downto 0);

```

```
--record declaration containing 3 fields of different types
type instruction_format is record
    op:opcode;
    md:mode;
    addr:address;
end record;

--signal of type instruction_format initialized
signal instr:instruction_format := (nop, 0, "00000000");

--assignment of record fields
instr.op <= add;
instr.md <= 1;
instr.addr <= "00000001";

--complete record assignment using named association
instr <= (op => add, md => 1, addr => "00000001");

--complete record assignment using positional association
instr <= (add, 1, "00000001");
```

CD392

@E: Can't convert expression to type <color>

The compiler was unable to convert the expression to match the type of the resultant assignment. In the following test case, signal cl is of enumerated type (color), but is assigned with two elements of color using an aggregate statement which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity error28 is
    port (A: in std_logic_vector(1 downto 0);
          B: in std_logic_vector(1 downto 0);
          C: out std_logic_vector(1 downto 0) );
end error28;
```

```

architecture rtl of error28 is
TYPE color IS (red, yellow, blue, white);
    signal cl : color;
begin
    cl <= (red, yellow);
    C <= A when (cl = red) else B;
end rtl;

```

Action

Correct the cl assignment statement. To eliminate the error in the above test case, make the assignment to a single element as shown in the corrected test case below.

```

library ieee;
use ieee.std_logic_1164.all;

entity error28 is
    port (A: in std_logic_vector(1 downto 0);
          B: in std_logic_vector(1 downto 0);
          C: out std_logic_vector(1 downto 0));
end error28;

architecture rtl of error28 is
TYPE color IS (red, yellow, blue, white);
    signal cl : color;
begin
    cl <= red;
    C <= A when (cl = red) else B;
end rtl;

```

CD393

@E: String doesn't match type <rec_ptr>

A string was assigned to a type that was not defined as a string. In the following test case, the signal assignment statement for rp contains the string 01 which is not of type rec_ptr.

```

library ieee;
use ieee.std_logic_1164.all;

```

```

entity error29 is
    port (A: in std_logic_vector(1 downto 0);
          B: in std_logic_vector(1 downto 0);
          C: out std_logic_vector(3 downto 0) );
end error29;

architecture rtl of error29 is
TYPE color IS (red, yellow, blue, white);
TYPE rec_ptr IS
RECORD
    d1 : color;
    d2 : std_logic_vector(1 downto 0);
END RECORD;
signal rp : rec_ptr;
begin
    rp <= "01";
    C <= (B & rp.d2);
end rtl;

```

Action

Use aggregate to assign all the elements of the record in a single statement. To correct the error, change the rp statement as shown in the corrected test case below.

```

library ieee;
use ieee.std_logic_1164.all;

entity error29 is
    port (A: in std_logic_vector(1 downto 0);
          B: in std_logic_vector(1 downto 0);
          C: out std_logic_vector(3 downto 0) );
end error29;

architecture rtl of error29 is
TYPE color IS (red, yellow, blue, white);
TYPE rec_ptr IS
RECORD
    d1 : color;
    d2 : std_logic_vector(1 downto 0);
END RECORD;
signal rp : rec_ptr;
begin
    rp <= (red, "10");
    C <= (B & rp.d2);
end rtl;

```

Record types belong to a composite class consisting of elements of different types. The element declaration for each record declares one or more identifiers and their types. The following code segments illustrate record definition and assignment.

```
--declaration of identifiers and their types
type opcode is (add, sub, mult, div, and, or, nop);
type mode is range 0 to 3;
type address is std_logic_vector (7 downto 0);

--record declaration containing 3 fields of different types
type instruction_format is record
    op:opcode;
    md:mode;
    addr:address;
end record;

--signal of type instruction_format initialized
signal instr(instruction_format) := (nop, 0, "00000000");

--assignment of record fields
instr.op <= add;
instr.md <= 1;
instr.addr <= "00000001";

--complete record assignment using named association
instr <= (op => add, md => 1, addr => "00000001");

--complete record assignment using positional association
instr <= (add, 1, "00000001");
```

CD394

@E: Non character array type for string <*memory*>

The assignment of an array type did not match its width. The following test case assigns all elements of the array mem in a single string which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity error30 is
  port (
    A: in std_logic_vector(1 downto 0);
    B: in std_logic_vector(1 downto 0);
    C: out std_logic_vector(7 downto 0) );
end error30;

architecture rtl of error30 is
TYPE memory IS array(0 to 3) of std_logic_vector(1 downto 0);
signal mem : memory;
begin
  mem <= "10101110";
  C <= (A & B & mem(2) & mem(3));
end rtl;
```

Action

Use aggregate assignments to define individual array elements. To eliminate the error in the above test case, edit the array type assignment to match its width as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity error30 is
  port (
    A: in std_logic_vector(1 downto 0);
    B: in std_logic_vector(1 downto 0);
    C: out std_logic_vector(7 downto 0) );
end error30;

architecture rtl of error30 is
TYPE memory IS array(0 to 3) of std_logic_vector(1 downto 0);
signal mem : memory;
begin
  mem <= ("10", "10", "11", "10");
  C <= (A & B & mem(2) & mem(3));
end rtl;
```

CD395

@E: Constant width <4> does not match context width <8>

A mismatch exists between the width of the arguments on both sides of an expression involving a constant value such as “00101111.” As shown in the following test case, a constant value of width 4 (i.e. “0100”) is assigned to the signal `outp` which is defined as an 8-bit vector.

```
library ieee;
use ieee.std_logic_1164.all;

entity decoder is
    port (inp: in std_logic_vector(2 downto 0);
          outp: out std_logic_vector(7 downto 0) );
end decoder;

architecture behave of decoder is
begin
    process (inp) begin
        case inp is
            when "000" => outp <= "00000001";
            when "001" => outp <= "00000010";
            when "010" => outp <= "00000100";
            when "011" => outp <= "00001000";
            when "100" => outp <= "00010000";
            when "101" => outp <= "00100000";
            when "110" => outp <= "0100"; red text
            when "111" => outp <= "10000000";
            when others => outp <= "XXXXXXXX";
        end case;
    end process;
end behave;
```

Action

Make sure that the cumulative width of the right side of an expression is equal to the left side of the expression.

```
library ieee;
use ieee.std_logic_1164.all;
```

```

entity decoder is
    port (inp: in std_logic_vector(2 downto 0);
          outp: out std_logic_vector(7 downto 0) );
end decoder;

architecture behave of decoder is
begin
    process (inp) begin
        case inp is
            when "000" => outp <= "00000001";
            when "001" => outp <= "00000010";
            when "010" => outp <= "00000100";
            when "011" => outp <= "00001000";
            when "100" => outp <= "00010000";
            when "101" => outp <= "00100000";
            when "110" => outp <= "01000000";
            when "111" => outp <= "10000000";
            when others => outp <= "XXXXXXXX";
        end case;
    end process;
end behave;

```

CD396

@E: Type mismatch of arguments in concatenation

A concatenation operator was used and the arguments to the concatenation operation were either not the same or they did not match the resultant type. The first test case causes the error when the data1 and data2 arguments used in the concatenation operator are of different types (i.e., std_logic and bit respectively).

```

library ieee;
use ieee.std_logic_1164.all;

entity dff1 is
    port (data1: in std_logic;
          clk: in std_logic;
          data2: in bit;
          q: out std_logic_vector(3 downto 0) );
end dff1;

```

```

architecture async_set_reset of dff1 is
begin
    setreset: process
    begin
        wait until rising_edge(clk);
        q <= data1 & "11" & data2;
    end process setreset;
end async_set_reset;

```

In the second test case, the resultant type q does not match the type of the data1, data2 arguments used in the concatenation.

```

library ieee;
use ieee.std_logic_1164.all;

entity dff1 is
    port (data1: in std_logic;
          clk: in std_logic;
          data2: in std_logic;
          q: out bit_vector(3 downto 0) );
end dff1;

architecture async_set_reset of dff1 is
begin
    setreset: process
    begin
        wait until rising_edge(clk);
        q <= data1 & "11" & data2;
    end process setreset;
end async_set_reset;

```

Action

Make sure that the concatenation is done with arguments of the same type. In the first test case, because the result of the concatenation (q) is std_logic_vector, make sure that all of the arguments on the right side of the expression are of type std_logic or std_logic_vector. To eliminate the error in the first test case, change the data2 argument as shown in the corrected test case below.

```

--1st example
library ieee;
use ieee.std_logic_1164.all;

```

```
entity dff1 is
    port (data1: in std_logic;
          clk: in std_logic;
          data2 : in std_logic;
          q: out std_logic_vector(3 downto 0) );
end dff1;

architecture async_set_reset of dff1 is
begin
    setreset: process
    begin
        wait until rising_edge(clk);
        q <= data1 & "11" & data2;
    end process setreset;
end async_set_reset;
```

To eliminate this error in the second test case, change the q: argument as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity dff1 is
    port (data1: in std_logic;
          clk: in std_logic;
          data2: in std_logic;
          q: out std_logic_vector(3 downto 0) );
end dff1;

architecture async_set_reset of dff1 is
begin
    setreset: process
    begin
        wait until rising_edge(clk);
        q <= data1 & "11" & data2;
    end process setreset;
end async_set_reset;
```

CD397

@E: Argument type of concatenation does not match with return type

A concatenation statement used characters that did not match the target type. The following test case generates the above message when port C is assigned an integer type (1 and 0) when the expected type is defined as std_logic_vector.

```
library ieee;
use ieee.std_logic_1164.all;

entity error31 is
    port (A: in std_logic_vector(1 downto 0);
          B: in std_logic_vector (1 downto 0);
          C: out std_logic_vector(1 downto 0) );
end error31;

architecture rtl of error31 is
begin
    C <= (0 & 1);
end rtl;
```

Action

In the above test case, 0 and 1 are integers while C is defined as a std_logic_vector. To correct the error in the test case, change the port assignment type as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity error31 is
    port (A: in std_logic_vector(1 downto 0);
          B: in std_logic_vector (1 downto 0);
          C: out std_logic_vector(1 downto 0) );
end error31;

architecture rtl of error31 is
begin
    C <= ('0' & '1');
end rtl;
```

CD398

@E: Can't assign type to null

A null was assigned to a signal. The following test case generates the above message.

```
library ieee;
use ieee.std_logic_1164.all;

entity error33 is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic );
end error33;

architecture rtl of error33 is
begin
    C <= null;
end rtl;
```

Action

Null cannot be assigned to a signal; nulls are primarily used in conjunction with case statements to indicate no action required on that case choice. Assigning null to a signal is a syntax error that results in the error.

CD401

@E: Expecting record for selected name

A signal that is not a record uses a tag. The following test case results in the error.

```
library ieee;
use ieee.std_logic_1164.all;
```

```

entity error36 is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic );
end error36;

architecture rtl of error36 is
TYPE bits IS ('0', '1', '0', '1');
signal bt : bits;
begin
    C <= A when (bt.a = '0') else B;
end rtl;

```

Action

Tags are used only for records. In the above test case, bt is of type bits of width 1. To eliminate this error in the above test case, remove the tag from the signal as shown in the corrected test case below.

```

library ieee;
use ieee.std_logic_1164.all;

entity error36 is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic ) ;
end error36;

architecture rtl of error36 is
TYPE bits IS ('0', '1', '0', '1');
signal bt : bits;
begin
    C <= A when (bt = '0') else B;
end rtl;

```

Record types belong to a composite class consisting of elements of different types. The element declaration for each record declares one or more identifiers and their types. The following code segments illustrate record definition and assignment.

```

--declaration of identifiers and their types
type opcode is (add, sub, mult, div, and, or, nop);
type mode is range 0 to 3;
type address is std_logic_vector (7 downto 0);

```

```
--record declaration containing 3 fields of different types
type instruction_format is record
    op:opcode;
    md:mode;
    addr:address;
end record;

--signal of type instruction_format initialized
signal instr(instruction_format) := (nop, 0, "00000000");

--assignment of record fields
instr.op <= add;
instr.md <= 1;
instr.addr <= "00000001";

--complete record assignment using named association
instr <= (op => add, md => 1, addr => "00000001");

--complete record assignment using positional association
instr <= (add, 1, "00000001");
```

CD403

@E: <d3> is not a member of type <rec_ptr>

A non-member element of a record was referenced. The following test case generates the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity error38 is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic_vector(1 downto 0) );
end error38;

architecture rtl of error38 is
TYPE color IS (red, yellow, blue, white);
TYPE rec_ptr IS
RECORD
    d1: color;
    d2: std_logic;
END RECORD;
```

```

signal rp: rec_ptr;
begin
    rp <= (d1 => red, d2 => A);
    C <= (B, rp.d3);
end rtl;

```

Action

Change the incorrect reference to the non-member element.

```

library ieee;
use ieee.std_logic_1164.all;

entity error38 is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic_vector(1 downto 0) );
end error38;

architecture rtl of error38 is
TYPE color IS (red, yellow, blue, white);
TYPE rec_ptr IS
RECORD
    d1: color;
    d2: std_logic;
END RECORD;
signal rp: rec_ptr;
begin
    rp <= (d1 => red, d2 => A);
    C <= (B, rp.d2);
end rtl;

```

Record types belong to a composite class consisting of elements of different types. Each record element declaration defines one or more identifiers and their types. The following code segments illustrate record definition and assignment.

```

--declaration of identifiers and their types
type opcode is (add, sub, mult, div, and, or, nop);
type mode is range 0 to 3;
type address is std_logic_vector (7 downto 0);

```

```
--record declaration containing 3 fields of different types
type instruction_format is record
    op:opcode;
    md:mode;
    addr:address;
end record;

--signal of type instruction_format initialized
signal instr:instruction_format := (nop, 0, "00000000");

--assignment of record fields
instr.op <= add;
instr.md <= 1;
instr.addr <= "00000001";

--complete record assignment using named association
instr <= (op => add, md => 1, addr => "00000001");

--complete record assignment using positional association
instr <= (add, 1, "00000001");
```

CD404

@E: Indexing operation does not match dimensionality of array

While indexing an array object, the array index included dimensions other than those specified in the array declaration. In the test case below, the single dimension array mem is indexed with two dimensions as mem(2,0) which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity array_test is
    port (A: in std_logic_vector(1 downto 0);
          B: in std_logic_vector(1 downto 0);
          C: out std_logic_vector(7 downto 0));
end array_test;
```

```

architecture rtl of array_test is
TYPE memory IS array(0 to 3) of std_logic_vector(1 downto 0);
signal mem : memory;
signal temp : std_logic_vector(1 downto 0);
begin
mem <= ("10", "10", "11", "10");
temp<=mem(2,0);
C <= (A & B & temp & mem(3));
end rtl;

```

Action

Make sure that while indexing an array object, that the dimensions match the declared dimensions of the array object. To eliminate the error in the above test case, remove the extra dimensions as shown in the corrected test case below.

```

library ieee;
use ieee.std_logic_1164.all;

entity array_test is
port (A: in std_logic_vector(1 downto 0);
      B: in std_logic_vector(1 downto 0);
      C: out std_logic_vector(7 downto 0));
end array_test;

architecture rtl of array_test is
TYPE memory IS array(0 to 3) of std_logic_vector(1 downto 0);
signal mem : memory;
signal temp : std_logic_vector(1 downto 0);
begin
mem <= ("10", "10", "11", "10");
temp<=mem(2);
C <= (A & B & temp & mem(3));
end rtl;

```

CD406

@E: result of "&" should be an array type

The concatenation operator used in VHDL was assigned a scalar quantity.

```
library ieee;
use ieee.std_logic_1164.all;

entity dff1 is
    port (data1,data2, clk: in std_logic;
          q: out std_logic);
end dff1;

architecture async_set_reset of dff1 is
begin
setreset:
    process
    begin
        wait until rising_edge(clk);
        q <= data1 & "11";
    end process setreset;
end async_set_reset;
```

Action

Make sure that objects are accessed as defined. The data1 is scalar (1-bit wide), and two bits “11” are concatenated to it; q should be 3 bits wide. To eliminate this error, change q to a 3-bit wide value (2 downto 0) as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity dff1 is
    port (data1,data2, clk: in std_logic;
          q: out std_logic_vector(2 downto 0));
end dff1;

architecture async_set_reset of dff1 is
begin
setreset:
    process
    begin
        wait until rising_edge(clk);
        q <= data1 & "11";
    end process setreset;
end async_set_reset;
```

CD409

@E: Object of indexed name is not an array

A data type defined as a scalar quantity was used as an array. In the test case below, feedthrough is an indexed object, but is defined as an integer with a scalar quantity which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity decoder is
    port (inp: in unsigned(3 downto 0);
          feedthrough : out integer range 0 to 15);
end decoder;

architecture behave of decoder is
begin
    process (inp)
    begin
        for i in 0 to 3 loop
            feedthrough(i) <= inp(i);
        end loop;
    end process;
end behave;
```

Action

Make sure that objects are accessed as defined. To eliminate the error in the above test case, change the definition of feedthrough to be a vector quantity to match its use as an indexed literal in the design.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity decoder is
    port (inp: in unsigned(3 downto 0);
          feedthrough : out unsigned (3 downto 0));
end decoder;
```

```
architecture behave of decoder is
begin
    process (inp)
    begin
        for i in 0 to 3 loop
            feedthrough(i) <= inp(i);
        end loop;
    end process;
```

CD413

@E: Can't determine range

The range expression of an array object could not be determined. In the test case below, the range expression for array object C is specified as A'range'reverse_range where A'range returns 1 downto 0 resulting in (1 downto 0)'reverse_range. Because this expression cannot be evaluated, the compiler errors out.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (A: in std_logic_vector(1 downto 0);
          B: in std_logic_vector(1 downto 0);
          C: out std_logic_vector(A'range'reverse_range));
end comb;

architecture rtl of comb is
begin
    c<= a and b;
end rtl;
```

Action

Be sure to specify a range expression that evaluates to a discrete range. In the test case below, specifying the range expression for the array object as A'RANGE (which evaluates to 1 downto 0) corrects the error.

```
library ieee;
use ieee.std_logic_1164.all;
```

```

entity comb is
    port (A: in std_logic_vector(1 downto 0);
          B: in std_logic_vector(1 downto 0);
          C: out std_logic_vector(A'range));
end comb;

architecture rtl of comb is
begin
    c<= a and b;
end rtl;

```

CD414

@E: Can't determine type of range

The type of the range could not be determined. When specifying the range for an array, the type of the range can be integer, positive, or natural. In the array declaration of my_array in the test case below, the range is specified as '1' downto '0'. However, because the type of 1 and 0 cannot be determined, the compiler errors out.

```

library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (A,B: in std_logic;
          C: out std_logic);
end comb;

architecture rtl of comb is
type my_type is ('0','1','U','X','Z');
type my_array is array('1' downto '0') of std_logic;
signal temp:my_array;
begin
    temp(0) <=a;
    temp(1)<=b;
    c <=temp(0) and temp(1);
end rtl;

```

Action

Make sure that the type of the range is either an integer or a positive or natural as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (A,B: in std_logic;
          C: out std_logic);
end comb;

architecture rtl of comb is
type my_type is ('0','1','U','X','Z');
type my_array is array(1 downto 0) of  std_logic;
signal temp:my_array;
begin
    temp(0) <=a;
    temp(1)<=b;
    c <=temp(0) and temp(1);
end rtl;
```

CD415

@E: Expecting keyword <is>

The compiler failed to find an expected a keyword. In the following test case, the 'is' keyword is missing in the architecture declaration.

```
library ieee;
use ieee.std_logic_1164.all;

entity adder is
    port (a, b, cin: std_logic;
          sum, cout: out std_logic );
end adder;

architecture behave of adder
begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;
```

Action

The compiler indicates the expected keyword. To eliminate this error in the above test case, add the ‘is’ keyword to the architecture declaration as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity adder is
    port (a, b, cin: std_logic;
          sum, cout: out std_logic );
end adder;

architecture behave of adder is
begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;
```

CD416

@E: Expecting <lib>.<package>.all

The package name was not user-defined. A keyword used in a package name also results in the above error. In the following test case, the package constant is not defined and therefore, does not exist. Also, the VHDL keyword constant is used.

```
library ieee;
use ieee.constant.all;

entity adder is
    port (a, b, cin:in std_logic;
          sum, cout: out std_logic );
end adder is;

architecture behave of adder is
signal temp: std_logic;
begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;
```

Action

Do not use VHDL keywords as literals. Make sure that user-defined names are given to objects that require names such as entities and packages. To eliminate the error in the above test case, edit the package name as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity adder is
    port (a, b, cin:in std_logic;
          sum, cout: out std_logic );
end adder is;

architecture behave of adder is
signal temp: std_logic;
begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;
```

CD417

@E: use: <temp> is not a library

The compiler encountered a use clause containing a library name that had not been declared using a library statement. In the following test case, the library temp is not declared as a library in the design. Although it may or may not be included in the project, the VHDL compiler requires all libraries used in a design to be declared as libraries using a library clause.

```
library ieee;
use temp.component.all;

entity adder is
    port (a, b, cin: in std_logic;
          sum, cout:out std_logic);
end adder;
```

```

architecture behave of adder is
begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;

```

Action

Declare the libraries used in the design. Make sure that the contents of the library are compiled into the required library by setting the VHDL directory. To eliminate this error in the above test case, declare the library name using a library statement.

```

library ieee;
library temp;
use temp.component.all;

entity adder is
    port (a, b, cin: in std_logic;
          sum, cout:out std_logic);
end adder;

architecture behave of adder is
begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;

```

CD418

@E: use: can't find <work>.<comb>.<temp>

The identifier specified in a use statement (*library.entityName.identifier*) does not exist in the corresponding entity. In the test case below, work.comb.temp is specified in the use statement, but identifier temp is not found in entity comb which causes the error.

```

library ieee;
use ieee.std_logic_1164.all;

```

```
entity comb is
    port (a_in, b_in, c_in: in std_logic;
          sum: out std_logic);
end comb;

architecture beh of comb is
use work.comb.temp;
begin
    sum<= a_in xor b_in xor c_in;
end beh;
```

Action

Make sure that the identifier specified in the use statement *library.entityName.identifier* exists in the path *library.entityName*. To eliminate the error in the above test case, change the *identifier* to .all in the use statement to implicitly indicate to use every identifier in the specified path as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (a_in, b_in, c_in: in std_logic;
          sum: out std_logic);
end comb;

architecture beh of comb is
use work.comb.all;
begin
    sum<= a_in xor b_in xor c_in;
end beh;
```

CD420

@E: Expecting <component> name

The component declaration was incomplete. In the following test case, the component declaration for the reg8 component is not fully declared (the name of the component is missing) which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity top_level1 is
port ( q: buffer std_logic_vector (7 downto 0);
       a, b: in std_logic_vector (7 downto 0);
       sel, r_l, clk, rst: in std_logic);
end top_level1;

architecture structural of top_level1 is
component muxhier -- component declaration for mux
port (outvec: out std_logic_vector (7 downto 0);
      a_vec, b_vec: in std_logic_vector (7 downto 0);
      sel: in std_logic);
end component;

component -- component declaration for reg8
port (q: buffer std_logic_vector (7 downto 0);
      data: in std_logic_vector (7 downto 0);
      clk, rst: in std_logic);
end component;

component rotate -- component declaration for rotate
port (q: buffer std_logic_vector (7 downto 0);
      data: in std_logic_vector (7 downto 0);
      clk, rst, r_l: in std_logic);
end component;

-- declare the internal signals here
signal mux_out, reg_out: std_logic_vector (7 downto 0);
begin -- structural description begins

-- instantiate a mux, name it inst1, and wire it up
-- here we connect the mux with positional port mapping (by
-- position)
inst1: muxhier port map (mux_out, a, b, sel);

-- instantiate a rotate, name it inst2, and wire it up
inst2: rotate port map (q, reg_out, clk, rst, r_l);

-- instantiate a reg8, name it inst3, and wire it up
-- reg8 is connected with named port mapping (by name)
-- the port connections can be given in any order
-- Note that the local signal names are on the right of the
-- '>=' mapping operators, and the signal names from the
-- component declaration are on the left.
```

```
inst3: reg8
    port map (clk => clk, data => mux_out,
               q => reg_out, rst => rst);
end structural;
```

Action

Make sure that components are fully declared within the HDL. In the above test case, completing the component declaration as shown in the code segment below eliminates the error.

```
component reg8 -- component declaration for reg8
port (q: buffer std_logic_vector (7 downto 0);
      data: in std_logic_vector (7 downto 0);
      clk, rst: in std_logic);
end component;
```

CD421

@E: Expecting entity name, got keyword <is>

The entity name was not specified in the entity decalration. In the following test case, the above error occurs when the entity name is not specified and the compiler encounters the keyword is instead of the entity name.

```
library ieee;
use ieee.std_logic_1164.all;

entity is
  port (a, b, cin: in std_logic;
        sum, cout:out std_logic );
end adder;

architecture behave of adder is
begin
  sum <= (a xor b) xor cin;
  cout <= (a and b) or (a and cin) or (b and cin);
end behave;
```

Action

Make sure that an entity name is specified. To eliminate the error in the above test case, replace entity is with entity adder is in the entity statement as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity adder is
    port (a, b, cin: in std_logic;
          sum, cout:out std_logic );
end adder;

architecture behave of adder is
begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;
```

CD422

@E: Expecting one of the keywords generic or port

The entity declaration was incomplete. The compiler expected the keyword port or generic after the specified entity name.

```
library ieee;
use ieee.std_logic_1164.all;

entity adder is
    (a, b, cin: in std_logic;
     sum, cout: out std_logic );
end adder;

architecture behave of adder is
begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;
```

Action

Make sure that the entity declaration includes the keyword port following the declaration name as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity adder is
    port (a, b, cin: in std_logic;
          sum, cout: out std_logic );
end adder;

architecture behave of adder is
begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;
```

CD424

@E: Expecting architecture identifier

The user-defined name of an architecture declaration was missing.

```
library ieee;
use ieee.std_logic_1164.all;

entity adder is
    port (a, b, cin: std_logic;
          sum, cout: out std_logic );
end adder;

architecture of adder is
begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;
```

Action

Name the architecture declaration. To eliminate the error in the above test case, create a user-defined name for the architecture declaration.

```

library ieee;
use ieee.std_logic_1164.all;

entity adder is
    port (a, b, cin: std_logic;
          sum, cout: out std_logic );
end adder;

architecture behave of adder is
begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;

```

CD425

@E: No such entity <fa>

There is no entity declaration for the entity name specified in the architecture declaration. In the test case below, fa is incorrectly specified as the entity name in the declaration portion of architecture df. Because there is no entity declared with the name fa, the compiler errors out.

```

library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (a,b : in std_logic;
          c: out std_logic);
end comb;

architecture df of fa is
begin
    c<= a and b;
end df;

```

Action

Make sure that the entity name specified in the architecture declaration has already been declared. To eliminate the error in the above test case, change the entity name to comb which is already declared as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (a,b : in std_logic;
          c: out std_logic);
end comb;

architecture df of comb is
begin
    c<= a and b;
end df;
```

CD427

@E: Can't find package for package body

The package declaration is missing from the design and only the package body declaration is present.

```
package body constants is
    type temp is (red, green, orange);
    constant RESET_VECTOR: std_logic_vector(10 downto 0)
        := "111111111111";
    constant TRIS_DEFAULT: std_logic_vector (7 downto 0)
        := "11111111";
end constants;
```

Action

Define the package declaration before the package body. All packages and corresponding package bodies must be compiled before using them in the design. To eliminate the error in the above test case, modify the code to include the package declaration with the package body as shown in the corrected test case below.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

package constants is
```

```

constant RESET_VECTOR: std_logic_vector(10 downto 0);
constant TRIS_DEFAULT: std_logic_vector (7 downto 0);
constant ONE: std_logic_vector (7 downto 0):= "00000001";

end constants;

package body constants is
    type temp is (red, green, orange);
constant RESET_VECTOR: std_logic_vector(10 downto 0)
    := "111111111111";
constant TRIS_DEFAULT: std_logic_vector (7 downto 0)
    := "11111111";
end constants;

```

CD428

@E: Duplicate definition of package <specialfunctions>

Multiple packages with the same package name were added to the project file. Packages are visible by the use of library and use clauses in the HDL code. In the following test case, there are two definitions of package specialfunctions in the file included in the project which results in the error.

```

--For example : Package file
package specialfunctions is
    function Pow (N,Exp : integer) return integer;
end specialfunctions;

package body specialFunctions is
    function Pow (N,Exp : integer) return integer is
        variable Result1 : integer := 1;
    begin
        for I in 1 to Exp loop
            Result1 := Result1 * N;
        end loop;
        return (Result1);
    end Pow;
end specialFunctions;

package specialfunctions is
    function Pow (N,Exp : integer) return integer;
end specialfunctions;

```

```

package body specialFunctions is
    function Pow (N,Exp : integer) return integer is
        variable Result : integer := 1;
    begin
        for I in 1 to Exp loop
            Result := Result * N;
        end loop;
        return (Result);
    end Pow;
end specialFunctions;

-- 4 bit up-counter with load and asynchronous low reset

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;
use work.specialfunctions.all;

entity powerof is
    port (clk: in std_logic;
          input: in unsigned (3 downto 0);
          power: out unsigned (15 downto 0) );
end powerof;

architecture behave of powerof is
signal inputValInt : integer range 0 to 15;
signal powerL : integer range 0 to 65535;
begin
    inputValInt<= to_integer (input);
    power <= to_unsigned (powerL,16);
    process
    begin
        wait until CLK='1';
        powerL <= Pow (inputValInt, 4);
    end process;
end behave;

```

Action

Make sure that all package names are unique. Also, make sure that packages with the same name are not compiled in the work directory. If packages with the same name must be used, make sure that you use different libraries to compile them by using the Tcl command:

```
add_file -vhdl -lib temp "my_package.vhd"
```

and use library-use clause to specify the correct package as shown below.

```
library temp;
use temp.my_package.all;
```

To eliminate the error in the above test case, comment out one of the package declarations.

```
--For example : Package file
package specialfunctions is
    function Pow (N,Exp : integer) return integer;
end specialfunctions;

package body specialFunctions is
    function Pow (N,Exp : integer) return integer is
        variable Result1 : integer := 1;
        begin
            for I in 1 to Exp loop
                Result1 := Result1 * N;
            end loop;
            return (Result1);
        end Pow;
    end specialFunctions;

--package specialfunctions is
--function Pow (N,Exp : integer) return integer;
--end specialfunctions;

--package body specialFunctions is
--function Pow (N,Exp : integer) return integer is
--variable Result : integer := 1;
--begin
--    for I in 1 to Exp loop
--        Result := Result * N;
--    end loop;
--    return (Result);
--end Pow;
--end specialFunctions;

-- 4 bit up-counter with load and asynchronous low reset

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;
use work.specialfunctions.all;
```

```
entity powerof is
    port (clk: in std_logic;
          input: in unsigned (3 downto 0);
          power: out unsigned (15 downto 0) );
end powerof;

architecture behave of powerof is
signal inputValInt : integer range 0 to 15;
signal powerL : integer range 0 to 65535;
begin
    inputValInt<= to_integer (input);
    power <= to_unsigned (powerL,16);
    process
    begin
        wait until CLK='1';
        powerL <= Pow (inputValInt, 4);
    end process;
end behave;
```

CD429

@E: package: end name does not agree with <constants>

The package name specified at the beginning of the package declaration did not match the package name at the end of package. The package must have the same name at the beginning and end of the package declaration. In the following test case, the beginning package identifier is constants, but the ending package identifier is temp.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

package constants is
    type ALU_SEL_TYPE is (ALU_FOUT, ALU_W, ALU_K, ALU_CNST);

    attribute syn_enum_encoding : string;
    attribute syn_enum_encoding of ALU_SEL_TYPE : type is
    "sequential";
```

```
constant RESET_VECTOR: std_logic_vector(10 downto 0)
    := "111111111111";
constant TRIS_DEFAULT: std_logic_vector (7 downto 0)
    := "11111111";
constant ONE: std_logic_vector (7 downto 0) := "00000001";
end temp;
```

Action

Make sure that the package identifier at the beginning and end of the package declaration match. The package name is optional in the end statement. To eliminate the error in the above test case, either remove the package name from the end statement or change the end statement as shown in the corrected test case below.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

package constants is
    type ALU_SEL_TYPE is (ALU_FOUT, ALU_W, ALU_K, ALU_CNST);

    attribute syn_enum_encoding : string;
    attribute syn_enum_encoding of ALU_SEL_TYPE : type is
        "sequential";

    constant RESET_VECTOR: std_logic_vector(10 downto 0)
        := "111111111111";
    constant TRIS_DEFAULT: std_logic_vector (7 downto 0)
        := "11111111";
    constant ONE: std_logic_vector (7 downto 0) := "00000001";
end constants;
```

CD430

@E: Expecting library unit

A use keyword was missing in a library declaration. In the following test case, the use keyword is missing in the statement specifying the library and packages in the design.

```
library ieee;
ieee.std_logic_1164.all;
```

```
entity adder is
    port (a, b, cin: std_logic;
          sum, cout: out std_logic );
end adder;

architecture behave of adder is
begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;
```

Action

Be sure to add the use keyword to the library declaration as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity adder is
    port (a, b, cin: std_logic;
          sum, cout: out std_logic );
end adder;

architecture behave of adder is
begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;
```

CD432

@E: Can't find library <temp>

A library was declared using the library statement in VHDL, but was not visible in the project. The VHDL compiler requires all libraries to be compiled prior to being used in the design files. In the following test case, library temp is declared, but is not visible which causes the compiler to error out.

```
library ieee;
library temp;
use ieee.std_logic_1164.all;
use temp.types.all;

entity adder is
    port (a: in user_defined_type;
          out1: out user_defined_type );
end adder;

architecture behave of adder is
begin
    process (a)
    begin
        out1 <= a;
    end process;
end behave;
```

Action

To eliminate the error in the above test case, compile the package my_package into the library temp by changing the following Tcl command in the project file from

```
add_file -vhdl -lib work "my_package.vhd"
```

to

```
add_file -vhdl -lib temp "my_package.vhd"
```

where my_package.vhd contains the declared package my_package. Compile the package below before you compile the above test case.

```
package types is
    type user_defined_type is ('x','1','0','z');
end package;

library ieee;
library temp;
use ieee.std_logic_1164.all;
use temp.types.all;

entity adder is
    port (a: in user_defined_type;
          out1: out user_defined_type );
end adder;
```

```
architecture behave of adder is
begin
    process (a)
    begin
        out1 <= a;
    end process;
end behave;
```

To change the library setting from the user interface, select Project->Select VHDL Library and change the library name from “work” to “temp.”

CD433

@W: No design units in file

The VHD files added to the project do not have an entity and architecture definition. This warning occurs along with the error “No entities found in input!”

Action

Make sure that the correct files are added to the project. Make sure that both an entity and architecture definition are present in the files in the project file for synthesis.

CD434

@W: Signal <sigName> in the sensitivity list is not used in the process. Make sure all variables in the sensitivity list are referenced in the process.

The sensitivity list in the process includes a signal that is not read in the process. The compiler assumes completeness; it infers the logic as if all the referenced signals (signals used in the process) are in the sensitivity list. The test case below gives the above warning because signal a has been added to the sensitivity list, but is not used within the process.

```

library ieee;
use ieee.std_logic_1164.all;

entity decoder is
    port (inp: in std_logic_vector(2 downto 0);
          a, b: in std_logic_vector (7 downto 0);
          outp: out std_logic_vector(7 downto 0));
end decoder;

architecture behave of decoder is
begin
    process (inp,a) begin
        case inp is
            when "000" => outp <= "00000001";
            when "001" => outp <= "00000010";
            when "010" => outp <= "00000100";
            when "011" => outp <= "00001000";
            when "100" => outp <= "00010000";
            when "101" => outp <= "00100000";
            when "110" => outp <= "01000000";
            when "111" => outp <= "10000000";
            when others => outp <= "XXXXXXXX";
        end case;
    end process;
end behave;

```

Action

Make sure that only the signals referenced in the process are listed in the sensitivity list to remove any possible RTL/simulation mismatches. To eliminate the warning in the above test case, edit out the unused signal in the sensitivity list as shown in the corrected test case below.

```

library ieee;
use ieee.std_logic_1164.all;

entity decoder is
    port (inp: in std_logic_vector(2 downto 0);
          a, b: in std_logic_vector (7 downto 0);
          outp: out std_logic_vector(7 downto 0));
end decoder;

architecture behave of decoder is
begin
    process (inp)
        begin
            case inp is

```

```
when "000" => outp <= "00000001";
when "001" => outp <= "00000010";
when "010" => outp <= "00000100";
when "011" => outp <= "00001000";
when "100" => outp <= "00010000";
when "101" => outp <= "00100000";
when "110" => outp <= "01000000";
when "111" => outp <= "10000000";
when others => outp <= "XXXXXXXX";
end case;
end process;
end behave;
```

CD436

@E: All waits in a process must be identical

Multiple wait statements are used in a process with different trigger conditions. The following test case results in the above message.

```
library ieee;
use ieee.std_logic_1164.all;

entity dff1 is
    port (data1, data2, clk: in std_logic;
          qrs: out std_logic );
end dff1;

architecture async_set_reset of dff1 is
begin
    setreset: process
    begin
        wait until rising_edge(clk);
        qrs <= data1;
        wait until falling_edge(clk);
        qrs <= data2;
    end process setreset;
end async_set_reset;
```

Action

The above code cannot be synthesized because the process contains wait statements that are not triggered on identical conditions. To eliminate the error in the above test case, use the same trigger conditions for the wait statements as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity dff1 is
    port (data1, data2, clk: in std_logic;
          qrs: out std_logic );
end dff1;

architecture async_set_reset of dff1 is
begin
    setreset: process
    begin
        wait until rising_edge(clk);
        qrs <= data1;
        wait until rising_edge(clk);
        qrs <= data2;
    end process setreset;
end async_set_reset;
```

In a process statement, either a sensitivity list or a wait statement must be used to trigger the process. Wait for statements (e.g., wait for 10 ns;) and wait statements (e.g., wait;) are ignored by the compiler and can cause RTL/post-synthesis simulation mismatches. The wait statements that are honored by the compiler are wait on (e.g., wait on a,b,c;) as well as wait until used with a clock edge (e.g., wait until rising_edge(clk);). Also make sure that all wait statements in the process are triggered on identical conditions.

CD441

@E: process must contain at least one wait

A sensitivity list/wait for statement is missing from within a process. A process statement in VHDL for synthesis must specify the condition that initiates the execution of the process. Initiation is done by two methods, a “sensitivity list”

or a wait statement at the end of the process that triggers the process to be executed. In the following test case, a sensitivity list/wait for statement is missing which results in the error when the process is initiated.

```
library ieee;
use ieee.std_logic_1164.all;

entity pro2 is
    port (a: in std_logic;
          b: in std_logic;
          c: out std_logic);
end;

architecture rtl of pro2 is
begin
    process
    begin
        c <= a and b; --and x(1);
    --wait;
    end process;
end;
```

Action

Specify a “Wait for/sensitivity list” inside the process statement before exiting the process itself. To eliminate the error in the above test case, enter the name of the sensitivity list or wait statement.

```
library ieee;
use ieee.std_logic_1164.all;

entity pro2 is
    port (a: in std_logic;
          b: in std_logic;
          c: out std_logic);
end;

architecture rtl of pro2 is
begin
    process (a,b)
    begin
        c <= a and b; --and x(1);
    --wait;
    end process;
end;
```

In a process statement, either a sensitivity list or a wait statement must be used to trigger the process. Wait for statements (e.g., wait for 10 ns;) and wait statements (e.g., wait;) are ignored by the compiler and can cause RTL/post synthesis simulation mismatches. The wait statements that are honored by the compiler are wait on (e.g., wait on a,b,c;) as well as wait until used with a clock edge (e.g., wait until rising_edge(clk);). Also make sure that all waits in the process are triggered on the identical condition.

CD442

@E: Can't derive sensitivity list from condition

The condition specified in a wait until statement is improper in such a way that the sensitivity list cannot be derived from the condition. In the test case below, condition clk=1 in the wait until statement is improper (1 is not of type std_logic) which, when compared with the clk input, results in the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (a,b : in std_logic;
          clk: in std_logic;
          c: out std_logic);
end seq;

architecture df of seq is
begin
    process
    begin
        c<= a and b;
        wait until clk=1 ;
    end process;
end df;
```

Action

Make sure that a valid condition is specified such as clk='1' which would allow the sensitivity list to be derived from the condition as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (a,b : in std_logic;
          clk: in std_logic;
          c: out std_logic);
end seq;

architecture df of seq is
begin
    process
    begin
        c<= a and b;
        wait until clk='1' ;
    end process;
end df;
```

CD443

@E: Wait with no wakeup condition is not supported except at end of process

A wait with no condition (wait for or wait until) is specified at the beginning of the process instead of at the end of the process. Process statements in VHDL are executed sequentially. When writing a VHDL process, specify the condition that initiates the execution of that process either with a sensitivity list or with a wait statement inside that process to trigger the process itself. In the following test case, the wait without a condition at the beginning of the process causes the error when it cannot determine when the process is initiated and is ended.

```
library ieee;
use ieee.std_logic_1164.all;

entity and2 is
    port (a: in std_logic;
          b: in std_logic;
          c: out std_logic );
end;
```

```
architecture rtl of and2 is
begin
    process
    begin
        wait;
        c <= a and b; --and x(1);
    end process;
end;
```

Action

Specify a wait for at the end of the process statement before exiting the process itself. You can also add a sensitivity list to perform the same function. To eliminate the error in the above test case, add a condition to wait and move it to the end of the process statement.

```
library ieee;
use ieee.std_logic_1164.all;

entity Pro2 is
    port (a: in std_logic;
          b: in std_logic;
          c: out std_logic );
end;

architecture rtl of and2 is
begin
    process
    begin
        c <= a and b; --and x(1);
        wait for 10ns;
    end process;
end;
```

In a process statement, either a sensitivity list or a wait statement must be used to trigger the process. Wait for statements (e.g., wait for 10 ns;) and wait statements (e.g., wait;) are ignored by the compiler and can cause RTL /post synthesis simulation mismatches. The wait statements that are honored by the compiler are wait on (e.g., wait on a,b,c;) as well as wait until used with a clock edge (e.g., wait until rising_edge(clk;)). Also make sure that all wait statements in a process are triggered on the identical condition.

CD449

@E: Divide by 0.0

An expression that required division by zero was encountered during evaluation (division by zero is not supported). The following test cases illustrates a divide by 0 error.

Verilog Example

```
module error (
    input clk,
    input [3:0] a,
    output reg [3:0] q );
    integer i;

    always@(posedge clk)
    begin
        for (i = 0; i < 4; i = i + 1)
            begin
                q[i] <= a[3/i]; // index cannot be evaluated for i = 0
            end
        end
    endmodule
```

Action

Analyze the design and recode the expression to remove the division by zero as shown in the corrected Verilog test case below.

```
module test (
    input clk,
    input [3:0] a,
    output reg [3:0] q);
    integer i;

    always@(posedge clk)
    begin
        for (i = 0; i < 4; i = i + 1)
            begin
                q[i] <= a[3/(i+1)];
            end
        end
    end
```

```
endmodule
```

VHDL Example

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (
        clk: in std_logic;
        a_in: in std_logic_vector(3 downto 0);
        d_out: out std_logic_vector(3 downto 0)
    );
end test;

architecture beh of test is
begin
    process (clk)
    begin
        if rising_edge(clk) then
            for i in 0 to 3 loop
                d_out(i) <= a_in(3/i);
            end loop;
        end if;
    end process;
end beh;
```

Action

Eliminate any expressions requiring division by zero. To eliminate the error in the above VHDL test case, change the divisor to a non-zero positive constant.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (
        clk: in std_logic;
        a_in: in std_logic_vector(3 downto 0);
        d_out: out std_logic_vector(3 downto 0)
    );
end test;
```

```

architecture beh of test is
begin
    process (clk)
    begin
        if rising_edge(clk) then
            for i in 0 to 3 loop
                d_out(i) <= a_in(3/(i+1));
            end loop;
        end if;
    end process;
end beh;

```

Division is supported only for unsigned values. When both the divisor and dividend are constants, the division is calculated by the compiler. For fractional results, the integral part is taken as the result of the division, for example, $3/3 = 1$, $5/4 = 1$, $3/4 = 0$. When the divisor and dividend are not constants, the divisor must be a non-zero positive constant power of 2.

Make sure that all division in the RTL code is done so that the divisor is a positive constant power of 2. In such cases, the logic created is essentially a right shift operation of the dividend. The number of times the shift operation occurs depends on the value of the divisor. For example, if the divisor is 8 ($2^{**}3$), the shift to the right is done three times. In the above test case, change divisor b to a positive constant power of 2, such as 1, 2, 4, 8, and so on.

CD450

@E: * with negative exponent <-15>

The exponentiation operator of an expression contained a negative exponent. In the following test case, the negative exponent in the for loop causes the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (a_in: in std_logic_vector(15 downto 0);
          dout: out integer );
end test;

```

```

architecture beh of test is
FUNCTION my_fun (bin: std_logic_vector) RETURN INTEGER
    IS VARIABLE result: INTEGER;
BEGIN
result := 0;
FOR i IN bin'RANGE LOOP
    IF bin(i) = '1' THEN
        result := result + 2**(-i);
    END IF;
END LOOP;
RETURN result;
END my_fun;

begin
dout <= my_fun(a_in);
end beh;

```

Action

Make sure that exponents are always positive numbers. To eliminate the error in the above test case, replace the negative number as shown in the corrected test case below.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (a_in: in std_logic_vector(15 downto 0);
          dout:out integer );
end test;

architecture beh of test is
FUNCTION my_fun (bin: std_logic_vector) RETURN INTEGER
    IS VARIABLE result: INTEGER;
BEGIN
result := 0;
FOR i IN bin'RANGE LOOP
    IF bin(i) = '1' THEN
        result := result + 2**(i);
    END IF;
END LOOP;
RETURN result;
END my_fun;

begin
dout <= my_fun(a_in);
end beh;

```

Exponentiation is only supported for base 2. If the two arguments are positive constant integers, the exponentiation is calculated by the tool. If one of arguments is a variable, it can only be the right argument of the exponentiation function as in the function $2^{**}b$.

CD451

@E: * large exponent caused overflow: <63>

An expression containing exponentiation overflowed during evaluation. The exponent in the following test case is in the range of 0 to 63 which causes the evaluated expression to overflow.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (a_in: in std_logic_vector(63 downto 0);
          dout: out integer );
end test;

architecture beh of test is
FUNCTION my_fun (bin: std_logic_vector) RETURN INTEGER
    IS VARIABLE result: INTEGER;
BEGIN
result := 0;
    FOR i IN bin'RANGE LOOP
        IF bin(i) = '1' THEN
            result := result + 2** (i);
        END IF;
    END LOOP;
RETURN result;
END my_fun;

begin
    dout <= my_fun(a_in);
end beh;
```

Action

Make sure that the range for which the exponent is evaluated is within reasonable limits. To eliminate the error in the above test case, lower the range of `a_in` as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (a_in: in std_logic_vector(30 downto 0);
          dout: out integer );
end test;

architecture beh of test is
FUNCTION my_fun (bin: std_logic_vector) RETURN INTEGER
    IS VARIABLE result: INTEGER;
BEGIN
    result := 0;
    FOR i IN bin'RANGE LOOP
        IF bin(i) = '1' THEN
            result := result + 2**i;
        END IF;
    END LOOP;
    RETURN result;
END my_fun;

begin
    dout <= my_fun(a_in);
end beh;
```

CD452

@E: overflow during evaluation of **

The evaluation of exponentiation resulted in an internal overflow. The compiler expects the exponent to be within limits to be able to evaluate the expression without errors. The exponent in the following test case ranges to 32 which results in an overflow during evaluation.

```
library ieee;
use ieee.std_logic_1164.all;
```

```

entity test is
    port (a_in: in std_logic_vector(32 downto 0);
          dout: out integer
         );
end test;

architecture beh of test is
FUNCTION my_fun (bin: std_logic_vector) RETURN INTEGER
    IS VARIABLE result: INTEGER;
BEGIN
result := 0;
    FOR i IN bin'RANGE LOOP
        IF bin(i) = '1' THEN
            result := result + 2**(i);
        END IF;
    END LOOP;
RETURN result;
END my_fun;

begin
    dout <= my_fun(a_in);
end beh;

```

Action

Constrain the value of the base and exponent such that they do not result in an overflow. To eliminate this error in the above testcase, change the value of the base and exponent to 30 as shown in the corrected test case below.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (a_in: in std_logic_vector(30 downto 0);
          dout: out integer
         );
end test;

architecture beh of test is
FUNCTION my_fun (bin: std_logic_vector) RETURN INTEGER
    IS VARIABLE result: INTEGER;
BEGIN
result := 0;
    FOR i IN bin'RANGE LOOP
        IF bin(i) = '1' THEN

```

```

        result := result + 2**(i);
    END IF;
END LOOP;
RETURN result;
END my_fun;

begin
    dout <= my_fun(a_in);
end beh;
```

CD453

@W: Index <8> may be out of range

The compiler encountered a signal that is accessed outside the range specified for the signal. In the following test case, `input_bus` is accessed with an index ranging from 0 to 8. Because the `input_bus` is defined as an input vector, `std_logic_vector(7 downto 0)`, the `input_bus(8)` is out of range.

```

library ieee;
use ieee.std_logic_1164.all;

entity parity is
generic (bus_size : integer := 8 );
port (input_bus: in std_logic_vector (7 downto 0);
      even_numbits, odd_numbits: out std_logic );
end parity;

architecture behave of parity is
begin
process (input_bus)
variable temp: std_logic;
begin
    temp := '0';
    for i in 0 to 8 loop
        temp := temp xor input_bus(i);
    end loop;
    odd_numbits <= temp;
    even_numbits <= not temp;
end process;
end behave;
```

Action

Make sure that the signals are indexed correctly as specified in the signal declaration. To eliminate the warning in the above test case, edit the for statement as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity parity is
generic (bus_size : integer := 8 );
port (input_bus: in std_logic_vector (7 downto 0);
      even_numbits, odd_numbits: out std_logic );
end parity;

architecture behave of parity is
begin
  process (input_bus)
  variable temp: std_logic;
  begin
    temp := '0';
    for i in 0 to 7 loop
      temp := temp xor input_bus(i);
    end loop;
    odd_numbits <= temp;
    even_numbits <= not temp;
  end process;
end behave;
```

A better method of ensuring that `input_bus(i)` remains within the specified vector range is to use the '`low`' and '`high`' VHDL attributes as shown below.

```
for i in input_bus'low to input_bus'high loop
```

CD455

@E: Slice's bounds are outside the variable's bounds

The bounds of a slice are outside the range defined for the vector variable object. When a slice of a variable vector is referenced in VHDL, the range of the slice must be either the same as that defined for the variable vector or a subset of that range. In the following test case, variable t is defined with range 0 to 1, but is accessed as 0 to 2 which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity cd455 is
    port (a: in std_logic_vector (1 downto 0);
          c: out std_logic_vector (0 to 1) );
end;

architecture rtl of cd455 is
begin
    process (a)
        variable t: std_logic_vector(0 to 1);
    begin
        t (0 to 2) := a(1 downto 0);
        c<=t;
    end process;
end;
```

Action

Make sure that the slice range of the variable is within the bounds of the defined vector range. To eliminate the error in the above test case, edit the range of the variable for the slice as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (a: in std_logic_vector (1 downto 0);
          c: out std_logic_vector (0 to 1) );
end;
```

```
architecture rtl of comb is
begin
    process (a)
        variable t: std_logic_vector(0 to 1);
    begin
        t (0 to 1) := a(1 downto 0);
        c<=t;
    end process;
end;
```

CD464

@E: Index <4> is out of range

A vector was indexed with a value that exceeded its declared range. For example, this error occurs if the loop index does not exceed the assigned port width in the loop. In the following test case, the generic width is 8 bits wide and the vector myout is 4 bits wide.

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

entity test_array is
    generic (width : integer := 8);
    port (a,b: in std_logic_vector (1 downto 0);
          myout: out std_logic_vector (3 downto 0));
end test_array;

architecture rtl of test_array is
    signal countl: signed (9 downto 0 );
begin
process (b)
begin
    for i in 0 to width -1 loop
        myout(i) <= '0';
    end loop;
end process;
end rtl;
```

Action

Correct the vector assignment statement to make the index value the same as its declared range. To eliminate this error in the above test case, either change the generic width to 4 (first test case) or change myout to 8 bits (second test case).

Changing the width of the generic

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

entity test_array is
generic (width : integer := 4);
    port (a,b: in std_logic_vector (1 downto 0);
          myout: out std_logic_vector (3 downto 0) );
end test_array;

architecture rtl of test_array is
signal countl: signed (9 downto 0 );
begin
process (b)
begin
    for i in 0 to width -1 loop
        myout(i) <= '0';
    end loop;
end process;
end rtl;
```

Changing the port definition

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

entity test_array is
generic (width : integer := 8);
    port (a,b: in std_logic_vector (1 downto 0);
          myout : out std_logic_vector (7 downto 0) );
end test_array;

architecture rtl of test_array is
signal countl: signed (9 downto 0 );
begin
process (b)
begin
```

```

        for i in 0 to width -1 loop
            myout(i) <= '0';
        end loop;
    end process;
end rtl;

```

CD466

@E: Assignment to signal under different clocks not supported

A signal was assigned values under two different clocks. In the test case below, signal mem is assigned two different values under two different clocks (clk1 and clk0) which causes the error.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity ramtest is
    port (q0,q1: out std_logic_vector (7 downto 0);
          data0,data1: in std_logic_vector (7 downto 0);
          waddr0,waddr1: in std_logic_vector (3 downto 0);
          we1,we0: in std_logic;
          clk0,clk1: in std_logic);
end ramtest ;

architecture rtl of ramtest is
type mem_type is array (15 downto 0) of
    std_logic_vector (7 downto 0);
signal reg_addr1,reg_addr0: std_logic_vector (3 downto 0);
signal mem : mem_type;
begin
q0 <= mem(conv_integer(reg_addr0));
q1 <= mem(conv_integer(reg_addr1));
process (clk0, we0, waddr0)
begin
    if rising_edge (clk0) then
        if we0 = '1' then
            mem (conv_integer (waddr0)) <=data0;
        end if;
        reg_addr0<= waddr0;
    end if;
end process;

```

```

process (clk1, we1, waddr1)
begin
    if rising_edge (clk1) then
        if we1 = '1' then
            mem (conv_integer (waddr1)) <= data1;
        end if;
        reg_addr1 <= waddr1;
    end if;
end process;
end rtl;

```

Action

Be sure not to assign values to a signal under different clocks or, when a dual-port block RAM is to be inferred, use a `syn_ramstyle` attribute with a value `no_rw_check` as shown in the corrected test case below.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity ramtest is
    port (q0,q1: out std_logic_vector (7 downto 0);
          data0,data1: in std_logic_vector (7 downto 0);
          waddr0,waddr1: in std_logic_vector (3 downto 0);
          we1,we0: in std_logic;
          clk0,clk1: in std_logic);
end ramtest ;

architecture rtl of ramtest is
type mem_type is array (15 downto 0) of
    std_logic_vector (7 downto 0);
signal reg_addr1,reg_addr0: std_logic_vector (3 downto 0);
signal mem : mem_type;
attribute syn_ramstyle: string;
attribute syn_ramstyle of mem : signal is "no_rw_check";
begin
q0 <= mem(conv_integer(reg_addr0));
q1 <= mem(conv_integer(reg_addr1));
process (clk0, we0, waddr0)
begin
    if rising_edge (clk0) then
        if we0 = '1' then

```

```

        mem (conv_integer (waddr0)) <=data0;
    end if;
    reg_addr0<= waddr0;
    end if;
end process;

process (clk1, wel, waddr1)
begin
    if rising_edge (clk1) then
        if wel = '1' then
            mem (conv_integer (waddr1)) <= data1;
        end if;
        reg_addr1 <= waddr1;
    end if;
end process;
end rtl;

```

CD472

@E: Slice's range direction does not match variable's

The range of a variable in an assignment was not the same direction (ascending/descending) as the range in its declaration. In the following test case, variable t is defined as an ascending vector (0 downto 1), but is assigned as a descending vector (1 downto 0) which causes the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity and2 is
    port (a: in std_logic_vector (1 downto 0);
          c: out std_logic_vector (0 downto 1) );
end;

architecture rtl of and2 is
begin
    process (a)
        variable t: std_logic_vector(0 to 1);
        begin
            t (1 downto 0) := a(1 downto 0); --Change of
            -- direction in t
        end process;
    end;

```

Action

Make sure that the direction of the variable is the same in both its declaration and assignment. To eliminate the error in the above test case, change the assignment of variable t as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity and2 is
    port (a: in std_logic_vector(1 downto 0);
          c: out std_logic_vector (0 downto 1) );
end;

architecture rtl of and2 is
begin
    process (a)
        variable t: std_logic_vector(0 to 1);
    begin
        t(0 to 1) := a(1 downto 0);
    end process;
end;
```

CD473

@E: integer constant overflowed limit of <2147483647>

The value of an integer declared in a design exceeded the limit of 2147483647. In the following test case, the output is being assigned a constant value of 234E10 which, when evaluated, exceeds the 2147483647 limit.

```
library ieee;
use ieee.std_logic_1164.all;

entity arith_test is
    port (a: in integer;
          b: out integer );
end entity;
```

```
architecture only of arith_test is
signal temp:integer;
begin
    temp <= 234E10;
    b <= temp;
end only;
```

Action

Do not use constants that exceed the limit of 2147483647. To eliminate the error in the above test case, change the output vector to a constant value below the 2147483647 limit.

```
library ieee;
use ieee.std_logic_1164.all;

entity arith_test is
    port (a: in integer;
          b: out integer );
end entity;

architecture only of arith_test is
signal temp:integer;
begin
    temp <= 234E6;
    b <= temp;
end only;
```

CD474

@E: doubled '_' is not allowed in numbers

Two consecutive underscores were encountered in a numeric value. A basic identifier in VHDL is composed of a sequence of one or more characters. A legal character is an upper-case letter (A...Z), a lower case letter (a...z), a digit (0...9), or the underscore (_) character. The first character in a basic identifier must be a letter and the last character cannot be an underscore. Also, two underscores cannot appear consecutively. Single underscores between numbers can be used to enhance readability. Two underscores used consecu-

tively is illegal in VHDL. In the following test case, output is assigned a constant value of 123_456 using two underscore characters which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity arith_test is
    port (a: in integer;
          b: out integer );
end entity;

architecture only of arith_test is
signal temp:integer;
begin
    temp <= 123_456;
    b <= temp;
end only;
```

Action

Use a single underscore character between numbers. To eliminate this error in the above test case, delete the extra underscore.

```
library ieee;
use ieee.std_logic_1164.all;

entity arith_test is
    port (a: in integer;
          b: out integer );
end entity;

architecture only of arith_test is
signal temp:integer;
begin
    temp <= 123_456;
    b <= temp;
end only;
```

CD475

@E: Misuse of _ in number

An underscore character was found at the end of an integer value with no further integer literals following the underscore. The underscore (_) character can be used while writing integer literals and has no impact on the value of the literal (i.e., 98_71_28 is the same as 987128). Single underscores between numbers can be used to improve readability. In the test case below, signal temp is assigned the constant value 100_; the terminating underscore character causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (a: in integer;
          b: out integer );
end entity;

architecture only of comb is
signal temp: integer;
begin
    temp <=100_ ;
    b<= a + temp;
end only;
```

Action

Make sure that any integer literals do not end with an underscore character. To eliminate the error in the above test case, remove the underscore at the end and use it between numbers as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (a: in integer;
          b: out integer );
end entity;
```

```
architecture only of comb is
signal temp: integer;
begin
    temp <=10_0 ;
    b<= a + temp;
end only;
```

CD476

@E: Exponent out of range

The exponent used is greater than 256. Exponent values are limited of 256 by the VHDL compiler. In the following test case, output is being assigned a constant value of E300 which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;
entity arith_test is
    port (a: in integer;
          b: out integer );
end entity;

architecture only of arith_test is
signal temp:integer;
begin
    temp <= 2E300;
    b <= temp;
end only;
```

Action

Make sure that any exponent values are less than 256 as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;
entity arith_test is
    port (a: in integer;
          b: out integer );
end entity;
```

```
architecture only of arith_test is
signal temp:integer;
begin
    temp <= 2E255;
    b <= temp;
end only;
```

CD477

@E: Expecting binary digit

The base of a bit string literal was specified, but the bit string was missing. If the base is binary, the compiler expects a binary digit. In the following test case, the bit string base is specified as type binary, but no binary digits are specified which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity arith_test is
    port (a: in integer;
          b: out bit_vector(3 downto 0) );
end arith_test;

architecture only of arith_test is
signal temp:bit_vector(3 downto 0);
begin
    temp <= to_stdlogicvector (bit_vector'(B"'));
    b <= temp;
end only;
```

Action

Make sure the bit string declaration is complete by specifying the binary digits after the base as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
```

```

entity arith_test is
    port (a: in integer;
          b: out bit_vector(3 downto 0) );
end arith_test;

architecture only of arith_test is
signal temp:bit_vector(3 downto 0);
begin
    temp <= to_stdlogicvector (bit_vector'(B"0001"));
    b <= temp;
end only;

```

CD479

@E: Expecting octal digit

The base of a bit string literal was specified, but the bit string was missing. If the base is octal, the compiler expects an octal digit. In the following test case, the bit string has the base of octal, but no octal digits are specified which results in the error.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity arith_test is
    port (a: in integer;
          b: out bit_vector(3 downto 0) );
end arith_test;

architecture only of arith_test is
signal temp:bit_vector(3 downto 0);
begin
    temp <= to_stdlogicvector (bit_vector'(0));
    b <= temp;
end only;

```

Action

Make sure that the bit string declaration is complete by adding the octal digits after the base as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity arith_test is
    port (a: in integer;
          b: out bit_vector(3 downto 0) );
end arith_test;

architecture only of arith_test is
signal temp:bit_vector(3 downto 0);
begin
    temp <= to_stdlogicvector (bit_vector'(0"123"));
    b <= temp;
end only;
```

CD480

@E: Expecting hex digit

The base of a bit string literal was specified, but the bit string was missing. If the base is hexadecimal, the compiler expects a hexadecimal digit. In the following test case, the bit string has a hex base, but no hex digits are specified which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity arith_test is
    port (a: in integer;
          b: out bit_vector(3 downto 0) );
end arith_test;

architecture only of arith_test is
signal temp:bit_vector(3 downto 0);
begin
    temp <= to_stdlogicvector (bit_vector'(X" ));
    b <= temp;
end only;
```

Action

Make sure that the bit string declaration is complete by adding the hex digits after the base. To eliminate this error in the above test case, change the bit string definition as shown in the corrected test case below:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity arith_test is
    port (a: in integer;
          b: out bit_vector(3 downto 0));
end arith_test;

architecture only of arith_test is
    signal temp:bit_vector(3 downto 0);
begin
    temp <= to_stdlogicvector (bit_vector'(X"A"));
    b <= temp;
end only;
```

CD483

@E: Base is out of range 2-16

The specified base is out of the allowed range (the VHDL compiler accepts bases ranging from 2 to 16). In the following test case, the base specified (20) is beyond the allowed range which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity arith_test is
    port (a: in integer;
          b: out integer );
end entity;
```

```
architecture only of arith_test is
signal temp: integer;
begin
    temp <= 20#102456#;
    b<= temp;
end only;
```

Action

Be sure to use bases within the range of 2 to 16. To eliminate the error in the above test case, change the HDL code as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity arith_test is
    port (a: in integer;
          b: out integer );
end entity;

architecture only of arith_test is
signal temp: integer;
begin
    temp <= 10#102456#;
    b<= temp;
end only;
```

CD484

@E: Expecting positive exponent

The expected exponent was negative (an exponent must be a positive constant). In the following test case, a negative exponent is being used which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity arith_test is
    port (a: in integer;
          b: out integer );
end entity;

architecture only of arith_test is
signal temp: integer;
begin
    temp <=2E-3;
    b<= temp;
end only;
```

Action

Make sure all exponents are positive as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity arith_test is
    port (a: in integer;
          b: out integer );
end entity;

architecture only of arith_test is
signal temp: integer;
begin
    temp <=2E3;
    b<= temp;
end only;
```

CD486

@E: Expecting digit after '.'

While using real type literals, a digit must always follow the decimal point ('.'). In the following test case, there is no digit after the decimal point which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity arith_test is
    port (a: in integer;
          b: out real );
end arith_test;

architecture only of arith_test is
signal temp: real;
begin
    temp <=2.;
    b <= temp;
end only;
```

Action

Make sure a digit always follows a decimal point as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity arith_test is
    port (a: in integer;
          b: out real );
end arith_test;

architecture only of arith_test is
signal temp: real;
begin
    temp <=2.0;
    b <= temp;
end only;
```

CD487

@E: Expecting trailing # for based number

A based literal is represented in the format *base#literal#* or *base#literal#Eexponent*. If the trailing '#' is missing following *literal*, the VHDL compiler errors out. In the following test case, the trailing '#' is missing while declaring the based literal.

```
library ieee;
use ieee.std_logic_1164.all;

entity arith_test is
    port (a: in integer;
          b: out integer );
end entity;

architecture only of arith_test is
signal temp: integer;
begin
    temp <=2#10101;
    b<= temp;
end only;
```

Action

Use the trailing '#' to declare based literals. To eliminate this error in the above test case, add a '#' as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity arith_test is
    port (a: in integer;
          b: out integer );
end entity;

architecture only of arith_test is
signal temp: integer;
begin
    temp <=2#10101#;
    b<= temp;
end only;
```

CD488

@E: EOF in string literal

A VHDL file ended abruptly with an incomplete string. In the following test case, the file ends with an incomplete string “101 which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity arith_test is
    port (a: in std_logic_vector(7 downto 0);
          b: out std_logic_vector(7 downto 0) );
end entity;

architecture only of arith_test is
signal temp:std_logic_vector(7 downto 0);
begin
    temp <= "101
```

Action

Make sure that the VHDL file is complete. To eliminate this error in the above testcase, complete the HDL code as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity arith_test is
    port (a: in std_logic_vector(7 downto 0);
          b: out std_logic_vector(7 downto 0) );
end entity;

architecture only of arith_test is
signal temp:std_logic_vector(7 downto 0);
begin
    temp <= "10101011";
    b <= temp;
end only;
```

CD489

@E: Newline in quoted string

A string assignment extended to a newline. Verilog language semantics require that strings be enclosed within quotation marks (""). If the closing double quote does not appear on the same line as shown in the test case below, the compiler errors out.

```
module display1;
reg [7:0] str;
initial
str = "\n\n\n\n\n\n
\n\na"
endmodule
```

Action

Closing the string on the same line as shown in the corrected test case below eliminates the error.

```
module display1;
reg [7:0] str;
initial
str = "\n\n\n\n\n\n\n\na"
endmodule
```

CD490

@E: EOF in extended identifier

An extended identifier was used without the final enclosing backslash. In VHDL, an extended identifier is a sequence of characters enclosed between two backslashes (\). In the test case below, the output is assigned the value \a@in with only a single starting backslash. The compiler interprets the single backslash as an EOF and errors out with the above message.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity comb is
    port (clk: in std_logic;
          \a@in\: in std_logic;
          d_out: out std_logic);
end comb;

architecture beh of comb is
begin
    process (clk)
    begin
        if falling_edge(clk) then
            d_out <= \a@in;
        end if;
    end process;
end beh;
```

Action

Make sure that all extended identifiers are enclosed between backslashes. To eliminate the error in the above test case, add a backslash at the end of the input value as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (clk: in std_logic;
          \a@in\: in std_logic;
          d_out: out std_logic);
end comb;

architecture beh of comb is
begin
    process (clk)
    begin
        if falling_edge(clk) then
            d_out <= \a@in\;
        end if;
    end process;
end beh;
```

CD492

@E: character '_' is not allowed as the first character

The first character of a basic identifier was an underscore. A basic identifier in VHDL is composed of a sequence of one or more characters. A legal character is an upper-case letter (A...Z), a lower case letter (a...z), a digit (0...9), or the underscore (_) character. The first character in a basic identifier must be a letter and the last character cannot be an underscore. Also, two underscores cannot appear consecutively. In the following test case, '_' is the first character of one of the input names which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity adder is
    port (a, _b, cin:std_logic;
          sum, cout:out std_logic);
end adder;

architecture behave of adder is
begin
    sum <= (a xor _b) xor cin;
    cout <= (a and _b) or (a and cin) or (_b and cin);
end behave;
```

Action

Use only an alphabetical character as the first character when specifying an identifier. To eliminate the error in the above test case, use a legal name for the identifier as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity adder is
    port (a, b, cin:std_logic;
          sum, cout:out std_logic);
end adder;

architecture behave of adder is
begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;
```

CD493

@E: character '<!' is not allowed

A basic identifier included an illegal character. A basic identifier in VHDL is composed of a sequence of one or more characters. A legal character is an upper-case letter (A...Z), a lower case letter (a...z), a digit (0...9), or the underscore (_) character. The first character in a basic identifier must be a letter and the last character cannot be an underscore. Also, two underscores cannot appear consecutively.

An extended identifier is a sequence of characters written between two backlashes. Any of the allowable characters can be used, including characters such as .,!,@,\$. In the following test case, ‘!’ is embedded in one of the input names as a basic identifier which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity adder is
    port (a, b!, cin:std_logic;
          sum, cout:out std_logic);
end adder;

architecture behave of adder is
begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;
```

Action

Use only alpha-numeric and underscore characters when specifying basic identifiers. If special characters must be used, use extended identifier format. To eliminate the error in the above test case, place the ‘!’ between two backslashes.

```
library ieee;
use ieee.std_logic_1164.all;

entity adder is
    port (a, b\!\, cin:std_logic;
          sum, cout:out std_logic);
end adder;
```

```

architecture behave of adder is
begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;

```

CD495

@E: Expecting range direction: to or downto

Range direction must use either the keyword to or downto. No other user-defined words are allowed in VHDL syntax. In the following test case, the range direction for `a`, `b`, and `cin` is incomplete which results in the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity adder is
    port (a, b, cin:std_logic_vector (1 0);
          sum, cout:out std_logic_vector (1 downto 0) );
end adder;

architecture behave of adder is
begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;

```

Action

Use to or downto to specify range direction. To eliminate the error in the above test case, add the required range specification.

```

library ieee;
use ieee.std_logic_1164.all;

entity adder is
    port (a, b, cin:std_logic_vector (1 downto 0);
          sum, cout:out std_logic_vector (1 downto 0) );
end adder;

```

```

architecture behave of adder is
begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;
```

CD497

@E: Expecting discrete type for indexing constraint

The indexing constraint type is not a discrete type. Enumerated and integer types are referred to as *discrete* types because they have an associated discrete value that can only be used for an indexing constraint. In the test case below, the indexing constraint of output C is specified as Boolean which results in the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic_vector(Boolean));
end comb;

architecture rtl of comb is
begin
    C<= (1=>a,0=> b) ;
end rtl ;
```

Action

Make sure to use discrete types in all indexing constraints. The above test case can be corrected by using the enumerated BOOLEAN type for the index constraint as shown in the corrected test case below.

```

library ieee;
use ieee.std_logic_1164.all;
```

```

entity comb is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic_vector(true downto false));
end comb;

architecture rtl of comb is
begin
    C<= (true=>a, false=> b) ;
end rtl ;

```

CD498

@E: Expecting type name

A signals or ports defined in VHDL was not associated with a type declaration. In the following test case, the type declaration on inputs a, b, and cin is missing.

```

library ieee;
use ieee.std_logic_1164.all;

entity adder is
    port (a, b, cin:
          sum, cout: out std_logic );
end adder;

architecture behave of adder is
begin
    sum <= (a xor b xor cin);
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;

```

Action

Make sure that the data type is included when a signal or port is declared in the HDL code. To eliminate this error in the above test case, add the type declarations for inputs a, b, and cin.

```

library ieee;
use ieee.std_logic_1164.all;

```

```
entity adder is
    port (a, b, cin: in std_logic ;
          sum, cout: out std_logic );
end adder;

architecture behave of adder is
begin
    sum <= (a xor b xor cin);
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;
```

CD501

@E: Expecting field name

A field name was omitted from a record definition. The missing second record definition in the following test case causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity error37 is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic_vector(1 downto 0) );
end error37;

architecture rtl of error37 is
TYPE color IS (red, yellow, blue, white);
TYPE rec_ptr IS
RECORD
    d1 : color;
    : std_logic;
END RECORD;
signal rp : rec_ptr;
begin
    rp <= (d1 => red, d2 => A);
    C <= (B, rp.d2);
end rtl;
```

Action

Use field names in the record definition. To eliminate the error in the above test case, add the second field name (d2) to the record definition as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity error37 is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic_vector(1 downto 0) );
end error37;

architecture rtl of error37 is
TYPE color IS (red, yellow, blue, white);
TYPE rec_ptr IS
RECORD
    d1 : color;
    d2 : std_logic;
END RECORD;
signal rp : rec_ptr;
begin
    rp <= (d1 => red, d2 => A);
    C <= (B, rp.d2);
end rtl;
```

Record types belong to a composite class consisting of elements of different types. The element declaration for each record declares one or more identifiers and their types. The following code segments illustrate record definition and assignment.

```
--declaration of identifiers and their types
type opcode is (add, sub, mult, div, and, or, nop);
type mode is range 0 to 3;
type address is std_logic_vector (7 downto 0);

--record declaration containing 3 fields of different types
type instruction_format is record
    op:opcode;
    md:mode;
    addr:address;
end record;

--signal of type instruction_format initialized
signal instr:instruction_format := (nop, 0, "00000000");
```

```
--assignment of record fields
instr.op <= add;
instr.md <= 1;
instr.addr <= "00000001";

--complete record assignment using named association
instr <= (op => add, md => 1, addr => "00000001");

--complete record assignment using positional association
instr <= (add, 1, "00000001");
```

CD502

@E: Fields must have a constrained types

A record included an unconstrained field. In the test case below, field d2 of the record rec_ptr is unconstrained which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (A: in std_logic_vector(1 downto 0);
          B: in std_logic;
          C: out std_logic_vector(2 downto 0) );
end comb;

architecture rtl of comb is
TYPE color IS (red, yellow, blue, white);
TYPE rec_ptr IS
RECORD
    d1 : color;
    d2 : std_logic_vector;
END RECORD;

signal rp : rec_ptr;
begin
    rp <= (d1 => red, d2 => A);
    C <= (B & rp.d2);
end rtl;
```

Action

Make sure that all fields of a record are constrained. The above test case can be corrected by constraining field d2 of record `rec_ptr` as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
  port (A: in std_logic_vector(1 downto 0);
        B: in std_logic;
        C: out std_logic_vector(2 downto 0) );
end comb;

architecture rtl of comb is
TYPE color IS (red, yellow, blue, white);
TYPE rec_ptr IS
RECORD
  d1 : color;
  d2 : std_logic_vector(1 downto 0);
END RECORD;

signal rp : rec_ptr;
begin
  rp <= (d1 => red, d2 => A);
  C <= (B & rp.d2);
end rtl;
```

CD503

@E: Duplicate field name <d1>

The same name is used for more than one field of a record. In the test case below, the two fields of record `rec_ptr` have the same name (`d1`) which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity comb is
    port (A: in std_logic_vector(1 downto 0);
          B: in std_logic;
          C: out std_logic_vector(2 downto 0) );
end comb;

architecture rtl of comb is
TYPE color IS (red, yellow, blue, white);
TYPE rec_ptr IS
RECORD
    d1 : color;
    d1 : std_logic_vector(1 downto 0);
END RECORD;

signal rp : rec_ptr;
begin
    rp <= (d1 => red, d2 => A);
    C <= (B & rp.d2);
end rtl;
```

Action

Make sure that all fields in a record have unique names. In the above test case, changing one of the field names to d2 as shown in the test case below corrects the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (A: in std_logic_vector(1 downto 0);
          B: in std_logic;
          C: out std_logic_vector(2 downto 0) );
end comb;

architecture rtl of comb is
TYPE color IS (red, yellow, blue, white);
TYPE rec_ptr IS
RECORD
    d1 : color;
    d2 : std_logic_vector(1 downto 0);
END RECORD;
```

```
signal rp : rec_ptr;
begin
    rp <= (d1 => red, d2 => A);
    C <= (B & rp.d2);
end rtl;
```

CD504

@E: type: Expecting type identifier

When a type is declared, it must be declared with an identifier. If the type identifier is missing, the VHDL compiler errors out with the above message. In the following test case, the type identifier is missing in the package.

```
package testarray is
    type data_bus is array (0 to 31) of bit;
    type is array (0 to 7) of bit;
    type test_enum is (green, red, blue);
    type test_enum_array is array (0 to 2) of test_enum;
end testarray;

use work.testarray.all;

entity extract is
    port (data: in data_bus;
          start: in integer;
          data_out: out small_bus);
end extract;

architecture test of extract is
begin
    process (data, start)
    begin
        for i in 0 to 7 loop
            data_out(i) <= data (i + start);
        end loop;
    end process;
end test;
```

Action

Make sure that the defined types have associated type identifiers. To eliminate the error in the above test case, add the type identifier.

```
package testarray is
    type data_bus is array (0 to 31) of bit;
    type small_bus is array (0 to 7) of bit;
    type test_enum is (green, red, blue);
    type test_enum_array is array (0 to 2) of test_enum;
end testarray;

use work.testarray.all;

entity extract is
    port (data: in data_bus;
          start: in integer;
          data_out: out small_bus);
end extract;

architecture test of extract is
begin
    process (data, start)
    begin
        for i in 0 to 7 loop
            data_out(i) <= data (i + start);
        end loop;
    end process;
end test;
```


CHAPTER 10

CD Messages 505 – 609

CD505

@E: type: Expecting enumeration literal

An enumerated type declaration identifier was enclosed within single quotes. An enumeration type declaration defines a type that has a set of user-defined values consisting of identifiers and character literals. When an enumerated type is declared using identifiers, the syntax required is:

```
type test is (red, green, yellow);
```

If an enumeration type is declared using character literals, the literals must be enclosed in single quotes:

```
type MVL is ('U', '0', '1', 'Z');
```

In the following test case, the enumerated type declaration uses identifiers within single quotes which causes the error.

```
package testarray is
    type data_bus is array (0 to 31) of bit;
    type small_bus is array (0 to 7) of bit;
    type test_enum is ('green', 'red', 'blue');
    type test_enum_array is array (0 to 2) of test_enum;
end testarray;
```

```
use work.testarray.all;

entity extract is
    port (data: in data_bus;
          start: in integer;
          data_out: out small_bus);
end extract;

architecture test of extract is
begin
    process (data, start)
    begin
        for i in 0 to 7 loop
            data_out(i) <= data (i + start);
        end loop;
    end process;
end test;
```

Action

Make sure that the correct syntax is used to declare enumerated types. To eliminate the error in the above test case, change the line declaring the enumerated type declaration as shown in the corrected test case below.

```
package testarray is
    type data_bus is array (0 to 31) of bit;
    type small_bus is array (0 to 7) of bit;
    type test_enum is (green, red, blue);
    type test_enum_array is array (0 to 2) of test_enum;
end testarray;

use work.testarray.all;

entity extract is
    port (data: in data_bus;
          start: in integer;
          data_out: out small_bus);
end extract;
```

```
architecture test of extract is
begin
    process (data, start)
    begin
        for i in 0 to 7 loop
            data_out(i) <= data (i + start);
        end loop;
    end process;
end test;
```

CD506

@E: Expecting , or ;

A use statement did not end with a semicolon or comma. In the test case below, the use statement is not terminated which results in the error.

```
library ieee;
use ieee.std_logic_1164.all

entity comb is
    port (a: in std_logic;
          d: out std_logic);
end comb;

architecture beh of comb is
begin
    d<= a;
end beh;
```

Action

Be sure to include a semicolon at the end of a use statement or to use a comma to separate two use clauses as shown in the corrected test case given below.

```
library ieee;
--use ieee.std_logic_1164.all;
use ieee.std_logic_1164.all, work.all;
```

```
entity comb is
    port (a: in std_logic;
          d: out std_logic);
end comb;

architecture beh of comb is
begin
    d<= a;
end beh;
```

CD512

@E: Expecting base unit

A base unit declaration was omitted from a physical type declaration. A physical type contains values that represent measurements of some physical quantity such as time or voltage. Values of this type are expressed as integer multiples of a base unit. In the test case below, no base unit is specified for physical type my_type which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (A: in std_logic;
          B: in std_logic;
          k: in integer range 0 to 1E9;
          C: out std_logic);
end comb;

architecture rtl of comb is
type my_type is range 0 to 1E9
units
end units;

signal temp: my_type ;
begin
    temp <= k ;
    c <= a and b after temp;
end rtl;
```

Action

Be sure to specify a base unit when declaring a physical type. In the test case below, specifying the base unit as ns as shown corrects the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (A: in std_logic;
          B: in std_logic;
          k: in integer range 0 to 1E9;
          C: out std_logic);
end comb;

architecture rtl of comb is

type my_type is range 0 to 1E9
units
    ns;
    ks =12 ns;
end units;

signal temp: my_type ;
begin
    temp <= k ;
    c <= a and b after temp;
end rtl;
```

CD513

@E: Duplicate physical unit in scope

Two units have the same name in a physical type declaration. A physical type contains values that represent measurements of some physical quantity such as time or voltage. Values of this type are expressed as integer multiples of a base unit. In the test case below, physical unit name ks appears twice with different multiples of the base unit (ns) which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity comb is
    port (A: in std_logic;
          B: in std_logic;
          k: in integer range 0 to 1E9;
          C: out std_logic);
end comb;

architecture rtl of comb is
type my_type is range 0 to 1E9
units
    ns;
    ks =12 ns;
    ks = 14 ns;
end units;

signal temp: my_type ;
begin
    temp <= k ;
    c <= a and b after temp;
end rtl;
```

Action

Avoid duplication of physical unit names. Renaming one of the secondary units to ms as shown in the corrected test case below corrects the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (A: in std_logic;
          B: in std_logic;
          k: in integer range 0 to 1E9;
          C: out std_logic);
end comb;

architecture rtl of comb is
type my_type is range 0 to 1E9
units
    ns;
    ks =12 ns;
    ms = 14 ns;
end units;
```

```

signal temp: my_type ;
begin
    temp <= k ;
    c <= a and b after temp;
end rtl;

```

CD514

@E: Expecting physical unit name

When defining a physical type, there was no secondary-unit name mentioned for an integer base-unit or for a secondary-unit. In the test case below, the secondary-unit name is not specified for 10nA which results in the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic);
end comb;

architecture rtl of comb is
type my_type is range 0 to 1E9
units
    nA;
    10 nA;
end units;
begin
    c <= a and b;
end rtl;

```

Action

Be sure to specify a new *secondary-unit* name for every multiple of a *basic-unit* or multiple of a previously defined *secondary-unit*. In the corrected test case below, *secondary-unit* JA is assigned a value which is a multiple of *base-unit* nA.

```

library ieee;
use ieee.std_logic_1164.all;

```

```

entity comb is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic);
end comb;

architecture rtl of comb is
type my_type is range 0 to 1E9
units
    nA;
    jA= 10 nA;
end units;
begin
    c <= a and b;
end rtl;

```

CD515

@E: Expecting multiplier for physical unit

When defining a physical type, the secondary-unit was assigned to the base-unit or to an already defined secondary-unit without a multiplier. In the test case below, secondary-unit uA is declared as equal to base-unit nA which results in the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic);
end comb;

architecture rtl of comb is
type my_type is range 0 to 1E9
units
    nA;
    uA= nA;
end units;
begin
    c <= a and b;
end rtl;

```

Action

Be sure to specify a *secondary-unit* name as a multiple of the *basic-unit* or as a multiple of a previously defined *secondary-unit*. In the corrected test case below, *secondary-unit* uA is defined as 1000 times *base-unit* nA.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic);
end comb;

architecture rtl of comb is
type my_type is range 0 to 1E9
units
    nA;
    uA= 1000 nA;
end units;
begin
    c <= a and b;
end rtl;
```

CD516

@E: Expecting reference physical unit

When defining a physical type, a *secondary-unit* declaration was not declared as a multiple of the *base-unit* or a previously defined *secondary-unit*. A physical type contains values that represent a measurement of some physical quantity such as time or voltage. Values of this type are expressed as integer multiples of a base unit. In the test case below, *secondary-unit* uA is assigned the value 1000 without any *base-unit* or previously defined *secondary-unit* which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity comb is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic);
end comb;

architecture rtl of comb is
type my_type is range 0 to 1E9
units
    nA;
    uA= 1000;
end units;
begin
    c <= a and b;
end rtl;
```

Action

Make sure any *secondary-unit* is defined as a multiple of a *base-unit* or a previously defined *secondary-unit*. In the corrected test case below, *secondary-unit* uA is defined as 1000 times *base-unit* nA.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic);
end comb;

architecture rtl of comb is
type my_type is range 0 to 1E9
units
    nA;
    uA= 1000 nA;
end units;
begin
    c <= a and b;
end rtl;
```

CD517

@E: Unknown reference unit

A *secondary-unit* is defined as a multiple of an undefined *base-unit* or *secondary-unit*. A physical type contains values that represent a measurement of some physical quantity such as time or voltage. Values of this type are expressed as integer multiples of a base unit. In the test case below, *secondary-unit* uA is defined as 1000 times kA. Because kA is neither defined as a *base-unit* nor as a *secondary-unit*, the compiler errors out.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic);
end comb;

architecture rtl of comb is
type my_type is range 0 to 1E9
units
    nA;
    uA= 1000 kA;
end units;
begin
    c <= a and b;
end rtl;
```

Action

Be sure to define any *secondary-unit* as a multiple of the predefined *base-unit* or *secondary-unit*. In the corrected test case below, *secondary-unit* uA is defined as 1000 times predefined *base-unit* nA.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic);
end comb;
```

```
architecture rtl of comb is
type my_type is range 0 to 1E9
units
  nA;
  uA= 1000 nA;
end units;
begin
  c <= a and b;
end rtl;
```

CD521

@E: Index constraint is illegal here

An illegal index constraint was encountered (the size constraint specified when declaring an array cannot be an index constraint). In the following test case, the size constraint for array my_array is defined in terms of an index to another type which causes error.

```
library ieee;
use ieee.std_logic_1164.all;

entity cd521 is
  port (A: in std_logic;
        B: in std_logic;
        C: out std_logic_vector(2 downto 0));
end cd521;

architecture rtl of cd521 is
type my_type is ('0','1');
type my_array is array(my_type(0) to my_type(1) )
  of std_logic_vector(2 downto 0);
signal temp:my_array;
begin
  temp(0)<= (2=>a,1=> b,0=>a);
  temp(1)<= (2=> not a, 1=> a and b ,0=>a);
  c <= temp(0) and temp(1);
end rtl;
```

Action

Avoid the illegal use of index constraints. In the corrected test case below, specifying the range of array my_array as an integer type as shown eliminates the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity cd521 is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic_vector(2 downto 0));
end cd521;

architecture rtl of cd521 is
type my_type is ('0','1');
type my_array is array(0 to 1) of std_logic_vector(2 downto 0);
signal temp:my_array;
begin
    temp(0)<= (2=>a,1=> b,0=>a);
    temp(1)<= (2=> not a, 1=> a and b ,0=>a);
    c <= temp(0) and temp(1);
end rtl;
```

CD522

@E: Illegal unconstrained range

The range constraint for an unconstrained array was illegal. The typical syntax for unconstrained array type declaration is:

type arrayName is array (*discrete_type range <>*) of *base_type*

In the test case below, the *discrete_type range* variable is missing in the unconstrained array definition for array my_array which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;
```

```

entity comb is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic_vector(2 downto 0 ));
end comb;

architecture rtl of comb is
type my_array is array( <> ) of std_logic_vector(2 downto 0 );
signal temp:my_array(0 to 1);
begin
    temp(0)<= (2=>a,1=> b,0=>a);
    temp(1)<= (2=> not a, 1=> a and b ,0=>a);
    c <= temp(0) and temp(1);
end rtl;

```

Action

Be sure to properly define the range syntax for an unconstrained array type. In the corrected test case below, the syntax for unconstrained array `my_array` is defined properly as `Integer range <>`.

```

library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic_vector(2 downto 0 ));
end comb;

architecture rtl of comb is
type my_array is array(Integer range <>)
    of std_logic_vector(2 downto 0 );
signal temp:my_array(0 to 1);
begin
    temp(0)<= (2=>a,1=> b,0=>a);
    temp(1)<= (2=> not a, 1=> a and b ,0=>a);
    c <= temp(0) and temp(1);
end rtl;

```

CD523

@E: Illegal mix of constrained and unconstrained indices

A mixture of constrained and unconstrained array indices is being incorrectly used within a multidemensional array definition. In the test case below, 2-dimensional array mytype is defined with one dimension constrained and the other dimension unconstrained which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (q: out std_logic_vector(2 downto 0);
          clk : in std_logic;
          d : in std_logic_vector(2 downto 0));
end seq;

architecture test of seq is
type mytype is array (1 downto 0, integer range <>) of
std_logic;
begin
    process(clk)
begin
    if(clk'event and clk='1') then
        q<=d;
    end if ;
    end process;
end test;
```

Action

Make sure that unconstrained and constrained indices are not mixed. In the corrected test case below, both range specifications for array mytype are constrained.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (q: out std_logic_vector(2 downto 0);
          clk : in std_logic;
          d : in std_logic_vector(2 downto 0));
end seq;
```

```

architecture test of seq is
type mytype is array (1 downto 0, 1 downto 0) of std_logic;
begin
    process(clk)
    begin
        if(clk'event and clk='1') then
            q<=d;
        end if ;
    end process;
end test;

```

CD527

@E: Unexpected type name, perhaps this statement should be a subtype declaration

The subtype declaration incorrectly starts with the keyword type instead of subtype. A subtype is a type with a range constraint; the general syntax is:

subtype subtype_name is base_type range bit_value to|downto bit_value

The compiler identifies a type declaration as a subtype by the presence of a *base-type* followed by a range constraint. In the test case below, the subtype declaration for my_datatype starts with the keyword type which results in the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (din: in std_logic_vector(1 downto 0);
          clk: in std_logic;
          q: out std_logic_vector(1 downto 0));
end seq;

architecture beh of seq is
type my_datatype is integer range 0 to 1;
begin
    process(clk)
        variable temp: my_datatype;
        if(clk'event and clk='1') then

```

```

        q(0)<=din(1);
        q(1)<=din(temp);
    end if;
end process;
end beh;
```

Action

Be sure to properly define the subtype. In the corrected test case below, the subtype declaration for my_datatype starts with the proper keyword subtype.

```

library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (din: in std_logic_vector(1 downto 0);
          clk: in std_logic;
          q: out std_logic_vector(1 downto 0));
end seq;

architecture beh of seq is
subtype my_datatype is integer range 0 to 1;
begin
    process(clk)
    variable temp: my_datatype;
    begin
        if(clk'event and clk='1') then
            q(0)<=din(1);
            q(1)<=din(temp);
        end if;
    end process;
end beh;
```

CD528

@E: Unknown type definition

The type declaration syntax is incorrect. The typical syntax for a type declaration is:

type type_name is range bit_value to|downto bit_value

In the test case below, the keyword range is missing from the type declaration which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (din: in std_logic_vector(1 downto 0);
          clk: in std_logic;
          q: out std_logic_vector(1 downto 0));
end seq;

architecture beh of seq is
type my_datatype is 0 to 1;
begin
    process(clk)
        variable temp: my_datatype;
    begin
        if(clk'event and clk='1') then
            q(0)<=din(1);
            q(1)<=din(temp);
        end if;
    end process;
end beh;
```

Action

Be sure to define a type with the proper syntax. In the corrected test case below, the type is defined correctly.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (din: in std_logic_vector(1 downto 0);
          clk: in std_logic;
          q: out std_logic_vector(1 downto 0));
end seq;

architecture beh of seq is
type my_datatype is range 0 to 1;
begin
    process(clk)
        variable temp: my_datatype;
    begin
        if(clk'event and clk='1') then
```

```

        q(0)<=din(1);
        q(1)<=din(temp);
    end if;
end process;
end beh;
```

CD529

@E: Expecting subtype name

A *subtype_name* was not specified in a subtype declaration. A subtype is a type with a range constraint; the general syntax is:

subtype subtype_name is base_type range bit_value to|downto bit_value

In the test case below, the subtype declaration statement does not include a subtype name following the keyword subtype which results in the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (din: in std_logic_vector(1 downto 0);
          clk: in std_logic;
          q: out std_logic_vector(1 downto 0));
end seq;

architecture str of seq is
subtype is integer range 0 to 1;
begin
    process(clk)
        variable temp: my_datatype;
    begin
        if(clk'event and clk='1') then
            q(0)<=din(1);
            q(1)<=din(temp);
        end if;
    end process;
end str;
```

Action

Make sure that all subtype declaration statements have a *subtype_name*. In the corrected test case below, the subtype name `my_datatype` has been added to the subtype declaration statement.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (din: in std_logic_vector(1 downto 0);
          clk: in std_logic;
          q: out std_logic_vector(1 downto 0));
end seq;

architecture str of seq is
    subtype my_datatype is integer range 0 to 1;
begin
    process(clk)
        variable temp: my_datatype;
    begin
        if(clk'event and clk='1') then
            q(0)<=din(1);
            q(1)<=din(temp);
        end if;
    end process;
end str;
```

CD532

@E: Bad Range Constraint

A range constraint was specified incorrectly. In the test case below, the range constraint for array `B` is specified as `integer range <>`. This form of range declaration is invalid which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity seq is
    port (A: in std_logic_vector(1 downto 0 );
          B: in std_logic_vector(integer range <> );
          C: out std_logic_vector(1 downto 0 ));
end seq;

architecture rtl of seq is
begin
    c <= a and b;
end rtl;
```

Action

Make sure to specify a valid range constraint. In the corrected test case below, the range constraint for array B is properly specified as integer range 1 downto 0.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (A: in std_logic_vector(1 downto 0 );
          B: in std_logic_vector(integer range 1 downto 0 );
          C: out std_logic_vector(1 downto 0 ));
end seq;

architecture rtl of seq is
begin
    c <= a and b;
end rtl;
```

CD533

@E: Range direction of std_logic incorrect

The range direction of a type was incorrect. In the test case below, the range of std_logic is incorrectly specified as '1' downto '0' in the subtype definition for my_type which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity seq is
    port (q: out std_logic_vector (2 downto 0);
          clk : in std_logic;
          d : in std_logic_vector(2 downto 0));
end seq;

architecture test of seq is
subtype my_type is std_logic range '1' downto '0';
begin
    process(clk)
    begin
        if(clk'event and clk='1') then
            q<=d;
        end if;
    end process;
end test;
```

Action

Be sure to specify the correct range direction for a type. In the corrected test case below, the range for std_logic is specified as '1' to '0'.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (q: out std_logic_vector (2 downto 0);
          clk : in std_logic;
          d : in std_logic_vector(2 downto 0));
end seq;

architecture test of seq is
subtype my_type is std_logic range '1' to '0';
begin
    process(clk)
    begin
        if(clk'event and clk='1') then
            q<=d;
        end if ;
    end process;
end test;
```

CD534

@E: Range constraint is not applicable to std_logic_vector

A range constraint was incorrectly applied to an array type. In the test case below, range constraint range 3 downto 0 is applied to array type std_logic_vector which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (a_in, b_in: in std_logic_vector(3 downto 0);
          c_out:out std_logic_vector range 3 downto 0 );
end seq;

architecture str of seq is
begin
    c_out <= a_in and b_in;
end str;
```

Action

Make sure that range constraints are not applied to array types. In the corrected test case below, the range constraint is removed from array type std_logic_vector.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (a_in, b_in: in std_logic_vector(3 downto 0);
          c_out:out std_logic_vector (3 downto 0 ) );
end seq;

architecture str of seq is
begin
    c_out <= a_in and b_in;
end str;
```

CD535

@E: Array type is already constrained!

A constrained array was constrained again. Generally, an unconstrained array is first declared and constrained later either while defining it in a subtype or while declaring a signal of that type. In the test case below, type memory is constrained in the type declaration statement itself and it is constrained again when defining signal mem.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (A: in std_logic_vector(1 downto 0);
          B: in std_logic_vector(1 downto 0);
          C: out std_logic_vector(7 downto 0) );
end seq;

architecture rtl of seq is
type memory is array(0 to 3 ) of std_logic_vector(1 downto 0);
signal mem : memory( 0 to 3 );
begin
    mem <= ("10", "10", "11", "10");
    C <= (A & B & mem(2) & mem(3));
end rtl;
```

Action

Make sure that constrained arrays are not constrained again. In the corrected test case below, array type memory is initially declared as an unconstrained array and is later constrained in the signal declaration for signal mem.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (A: in std_logic_vector(1 downto 0);
          B: in std_logic_vector(1 downto 0);
          C: out std_logic_vector(7 downto 0) );
end seq;
```

```
architecture rtl of seq is
type memory is array(integer range <> )
  of std_logic_vector(1 downto 0);
signal mem : memory( 0 to 3);
begin
  mem <= ("10", "10", "11", "10");
  C <= (A & B & mem(2) & mem(3));
end rtl;
```

CD536

@E: Type must be an unconstrained array

The compiler encountered a range constraint on a type that is not an unconstrained array. In the test case below, the range constraint is incorrectly declared type `std_logic` which is not an unconstrained array and which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
  port (q: out std_logic(2 downto 0);
        clk : in std_logic;
        d : in std_logic_vector(2 downto 0));
end seq;

architecture test of seq is
begin
  process(clk)
  begin
    if(clk'event and clk='1') then
      q<=d;
    end if;
  end process;
end test;
```

Action

Be sure to specify range constraints only on unconstrained arrays. In the corrected test case below, the range constraint is applied to unconstrained array `std_logic_vector`.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (q: out std_logic_vector(2 downto 0);
          clk : in std_logic;
          d : in std_logic_vector(2 downto 0));
end seq;

architecture test of seq is
begin
    process(clk)
    begin
        if(clk'event and clk='1') then
            q<=d;
        end if;
    end process;
end test;
```

CD538

@E: Expecting closing) for interface list

A closing parenthesis was omitted in an interface list for a function or procedure. In the test case below, the closing parenthesis is missing in the interface list of function my_and which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0) );
end seq;
```

```
architecture beh of seq is
function my_and (a, b: std_logic_vector return
std_logic_vector is
variable temp: std_logic_vector(a'range);
begin
    temp:= a and b;
    return temp;
end my_and;

begin
    process (clk)
    begin
        if falling_edge(clk) then
            d_out <= my_and(a_in, b_in);
        end if;
    end process;
end beh;
```

Action

Make sure that closing parentheses are included in all interface lists. In the corrected test case below, a closing parenthesis is added to terminate the interface list for function my_and.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0));
end seq;

architecture beh of seq is
function my_and (a, b: std_logic_vector) return std_logic_vector
is
variable temp: std_logic_vector(a'range);
begin
    temp:= a and b;
    return temp;
end my_and;
```

```
begin
  process (clk)
  begin
    if falling_edge(clk) then
      d_out <= my_and(a_in, b_in);
    end if;
  end process;
end beh;
```

CD539

@E: Expecting identifier: Unbalanced parenthesis around generic list?

A generic declaration was missing an enclosing parenthesis for the generic list. In the test case below, the generic declaration for bus_size has a missing terminating parenthesis which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
  generic (bus_size : integer:=8;
  port (input_bus : in std_logic_vector (bus_size-1 downto 0);
        even_numbits, odd_numbits : out std_logic );
end seq;

architecture behave of seq is
begin
  process (input_bus)
  variable temp: std_logic;
  begin
    temp := '0';
    for i in input_bus'low to input_bus'high loop
      temp := temp xor input_bus(i);
    end loop;
    odd_numbits <= temp;
    even_numbits <= not temp;
  end process;
end behave;
```

Action:

Make sure to always enclose generic lists in parentheses. In the corrected test case below, a closing parenthesis is added to terminate the generic bus_size.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
generic (bus_size : integer:=8);
    port (input_bus : in std_logic_vector (bus_size-1 downto 0);
          even_numbits, odd_numbits : out std_logic );
end seq;

architecture behave of seq is
begin
    process (input_bus)
    variable temp: std_logic;
    begin
        temp := '0';
        for i in input_bus'low to input_bus'high loop
            temp := temp xor input_bus(i);
        end loop;
        odd_numbits <= temp;
        even_numbits <= not temp;
    end process;
end behave;
```

CD540

@E: Expecting identifier: Unbalanced parenthesis around port list?

A port declaration is missing a parenthesis enclosing the port list. In the test case below, the port declaration has a missing terminating parenthesis which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (a, b : in std_logic_vector(0 to 2);
          q: out std_logic_vector(0 to 2) ;
    end comb;
```

```
architecture test of comb is
begin
    q<= a and b;
end test;
```

Action

Make sure that the parentheses enclosing a port declaration are always balanced. In the corrected test case below, a terminating parenthesis is added to the port declaration.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (a, b : in std_logic_vector(0 to 2);
          q: out  std_logic_vector(0 to 2) );
end comb;

architecture test of comb is
begin
    q<= a and b;
end test;
```

CD541

@E: Expecting ;

A semicolon delimiter is missing in the HDL code. In the following test case, the semicolon is missing in the assignment to sum.

```
library ieee;
use ieee.std_logic_1164.all;

entity adder is
    port (a, b, cin: in std_logic;
          sum, cout:out std_logic );
end adder;
```

```
architecture behave of adder is
begin
    sum <= (a xor b) xor cin
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;
```

Action

Place delimiters as defined by the language. To eliminate the error in the above test case, add a semicolon at the end of the sum statement.

```
library ieee;
use ieee.std_logic_1164.all;

entity adder is
    port (a, b, cin: in std_logic;
          sum, cout:out std_logic );
end adder;

architecture behave of adder is
begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;
```

Crossprobe the error in the log file to point to the line that requires the delimiter.

CD543

@E: Too many names

Multiple names of objects were referenced within an attribute. In the following test case, syn_preserve is applied on architectures bhv and bv. Because a single architecture (bhv) is expected, the compiler errors out.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity for_while is
port (clk : in std_logic ;
      d : in std_logic_vector(7 downto 0);
      q : out std_logic_vector(7 downto 0));
end entity ; -----for_while;

architecture bhv of for_while is
attribute syn_preserve : boolean ;
attribute syn_preserve of bhv, bv : architecture is true ;
begin
  process
  begin
    wait until clk'event and clk = '1' ;
    for i in 0 to 7 loop
      q(i) <= d(i) ;
    end loop;
  end process ;
end architecture bhv;
```

Action

Make sure multiple names are not used. As shown in the corrected test case below, specifying a single architecture for the `syn_preserve` attribute eliminates the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity for_while is
port (clk : in std_logic ;
      d : in std_logic_vector(7 downto 0);
      q : out std_logic_vector(7 downto 0));
end entity ; -----for_while;

architecture bhv of for_while is
attribute syn_preserve : boolean ;
attribute syn_preserve of bhv : architecture is true ;
begin
  process
  begin
    wait until clk'event and clk = '1' ;
    for i in 0 to 7 loop
      q(i) <= d(i) ;
    end loop;
  end process ;
end architecture bhv;
```

CD544

@E: Expecting <entity> name <and_bb>

An attribute was declared on an entity in an entity declaration block and the name used in the attribute declaration did not match the name of the entity. In the following test case, the name of the entity is `and_bb` and the name appearing in the attribute declaration is `and_bb1` which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity and_bb is
    port (a,b : in std_logic;
          c: out std_logic );
attribute syn_black_box: boolean ;
attribute syn_black_box of and_bb1: entity  is true;
end and_bb;

architecture dummy of and_bb is
begin
end dummy;

library ieee;
use ieee.std_logic_1164.all;

entity top is
    port (a,b : in std_logic;
          c: out std_logic );
end entity;

architecture test of top is
signal temp : std_logic;
component and_bb is
    port (a,b : in std_logic;
          c: out std_logic );
end component;

begin
    u1: and_bb port map (a,b,temp);
    c <= not temp;
end test;
```

Action

Make sure that you use the same entity name as that of the enclosing entity when declaring an attribute inside the entity declaration as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity and_bb is
    port (a,b : in std_logic;
          c: out std_logic );
attribute syn_black_box: boolean ;
attribute syn_black_box of and_bb: entity  is true;
end and_bb;

architecture dummy of and_bb is
begin
end dummy;

library ieee;
use ieee.std_logic_1164.all;

entity top is
    port (a,b : in std_logic;
          c: out std_logic );
end entity;

architecture test of top is
signal temp : std_logic;
component and_bb is
    port (a,b : in std_logic;
          c: out std_logic );
end component;

begin
    u1: and_bb port map (a,b,temp);
    c <= not temp;
end test;
```

CD547

@E: Expecting architecture name <behave>

The declaration name used in the architecture end statement did not match the declaration name at the beginning. In the following test case, the architecture declaration name is behave, but the end statement uses temp as the declaration name which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity adder is
    port (a, b, cin: std_logic;
          sum, cout: out std_logic );
end adder;

architecture behave of adder is
begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or (a and cin) or (b and cin);
end temp;
```

Action

Be sure to use the same declaration name at the beginning and end of the architecture statement. To eliminate this error in the above test case, replace temp with behave in the end statement.

```
library ieee;
use ieee.std_logic_1164.all;

entity adder is
    port (a, b, cin: std_logic;
          sum, cout: out std_logic );
end adder;

architecture behave of adder is
begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;
```

The architecture name in the end statement is optional.

CD548

@E: No such labeled stmt <u>

An incorrect component label was encountered in an attribute declaration. In the following test case, syn_macro is applied on label u instead of u1 (the correct label) which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity fa4 is
port (a, b : in std_logic_vector(3 downto 0);
      cin : in std_logic;
      sum : out std_logic_vector(3 downto 0);
      cout : out std_logic);
end fa4;

architecture bhv of fa4 is
attribute syn_macro : boolean;

component fa
port (a,b,cin : in std_logic;
      sum, cout : out std_logic);
end component;

signal temp : std_logic_vector(4 downto 0);
attribute syn_macro of u : label is true ;
begin
  temp(0) <= cin;
  GEN : for i in 0 to 3 generate
    u1 : fa port map( a => a(i) , b => b(i), cin => temp(i), sum
=>
      sum(i), cout => temp(i+1));
  end generate;
  cout <= temp(4);
end bhv;
```

Action

Correcting the label within the attribute as shown in the test case below eliminates the above error.

```
library ieee;
use ieee.std_logic_1164.all;
```

```

entity fa4 is
port (a, b : in std_logic_vector(3 downto 0);
      cin : in std_logic;
      sum : out std_logic_vector(3 downto 0);
      cout : out std_logic);
end fa4;

architecture bhv of fa4 is
attribute syn_macro : boolean;

component fa
port (a,b,cin : in std_logic;
      sum, cout : out std_logic);
end component;

signal temp : std_logic_vector(4 downto 0);
attribute syn_macro of u1 : label is true ;
begin
  temp(0) <= cin;
  GEN : for i in 0 to 3 generate
    u1 : fa port map( a => a(i) , b => b(i), cin => temp(i), sum
=>
      sum(i), cout => temp(i+1));
    end generate;
  cout <= temp(4);
end bhv;

```

CD551

@E: Alias must have a constrained array type

An alias was declared on an unconstrained type array. In the following test case, alias foo uses bits 31 and 30 of a vector whose dimensions are not defined (an unconstrained array) which causes the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
port (a : in std_logic_vector;
      b : in std_logic;
      c: out std_logic );
end test ;

```

```
architecture test_arc of test is
alias foo is a(31 downto 30);
begin
-- architecture test_arc
  c <= (a(0) and b) or (foo(31) and foo(30));
end test_arc;
```

Action

Make sure that the array is constrained when declaring an alias. To eliminate the syntax error in the above test case, constrain the input a port definition as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
port (a : in std_logic_vector(31 downto 0);
      b : in std_logic;
      c: out std_logic );
end test ;

architecture test_arc of test is
alias foo is a(31 downto 30);
begin
-- architecture test_arc
  c <= (a(0) and b) or (foo(31) and foo(30));
end test_arc;
```

CD554

@E: Can't associate actual. Not enough formal parameters.

The number of actuals exceeds the number of formals when using positional association for component instantiation. In the test case below, there is only one generic declared in the component declaration but, when instantiating the component, the number of actual generics is two which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity and_gate is
generic (p:natural);
  port (a,b : in std_logic_vector(p-1 downto 0);
        c: out std_logic_vector(p-1 downto 0));
end and_gate;

architecture df of and_gate is
begin
  c<= a and b;
end df;

library ieee;
use ieee.std_logic_1164.all;

entity comb is
  port (a,b : in std_logic_vector( 1 downto 0);
        c: out std_logic_vector(1 downto 0));
end comb;

architecture beh of comb is
component and_gate
generic(p:natural);
  port (a,b : in std_logic_vector(p-1 downto 0);
        c: out std_logic_vector(p-1 downto 0));
end component;

begin
  a: and_gate generic map (2,4) port map (a,b,c);
end beh;
```

Action

Make sure that the same number of actuals and formals are used as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity and_gate is
generic (p:natural);
  port (a,b : in std_logic_vector(p-1 downto 0);
        c: out std_logic_vector(p-1 downto 0));
end and_gate;
```

```
architecture df of and_gate is
begin
    c<= a and b;
end df;

library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (a,b : in std_logic_vector( 1 downto 0);
          c: out std_logic_vector(1 downto 0));
end comb;

architecture beh of comb is
component and_gate
generic(p:natural);
    port (a,b : in std_logic_vector(p-1 downto 0);
          c: out std_logic_vector(p-1 downto 0));
end component;

begin
    a: and_gate generic map (2) port map (a,b,c);
end beh;
```

CD555

@E: Unknown identifier ident: <rst1>

An unknown identifier is used as an actual while instantiating a component. In the test case below, actual `rst1` in the component instantiation statement is not defined as a signal or port which causes the error.

```
entity test1 is
    port (clk,rst: in bit;
          din: in bit_vector(3 downto 0);
          dout : out bit_vector(3 downto 0));
end;
```

```
architecture rtl of test1 is
begin
    process(clk,rst)
    begin
        if rst='1' then
            dout<="0000";
        elsif (clk'event and clk='1') then
            dout<=din;
        end if;
    end process;
end;

entity seq is
    port (clk,rst: in bit;
          din: in bit_vector(3 downto 0);
          dout : out bit_vector(3 downto 0));
end;

architecture test_rtl of seq is
begin
    u:entity work.test1 port map(clk,rst1,din,dout);
end;
```

Action

Be sure to use previously defined identifiers as actuals as shown in the corrected test case below.

```
entity test1 is
    port (clk,rst: in bit;
          din: in bit_vector(3 downto 0);
          dout : out bit_vector(3 downto 0));
end;

architecture rtl of test1 is
begin
    process(clk,rst)
    begin
        if rst='1' then
            dout<="0000";
        elsif (clk'event and clk='1') then
            dout<=din;
        end if;
    end process;
end;
```

```
entity seq is
    port (clk,rst: in bit;
          din: in bit_vector(3 downto 0);
          dout : out bit_vector(3 downto 0));
end;

architecture test_rtl of seq is
begin
    u:entity work.test1 port map(clk,rst,din,dout);
end;
```

CD565

@E: Duplicate case label detected

The VHDL language semantics require that a `case` statement not contain duplicate case labels. In the code segment below, case label 1001 appears twice which causes the error.

```
case shift is
    when "1000" => temp <= temp;
    when "1001" => temp(15 downto 0) <= temp(14 downto 0) & '0';
    when "1001" => temp(15 downto 0) <= temp(13 downto 0)
        & "00";
    when "1111" => temp(15 downto 0) <= temp(8 downto 0) &
"00000000";
    when others => temp <= temp;
end case;
```

Action

Make sure that a case label is used only once within a `case` statement. In the above segment, removing or modifying the statement corresponding to case label 1001 would eliminate the error.

CD567

@E: Wait statements not allowed in process with sensitivity list

A wait statement was included in a process with a sensitivity list. The VHDL language syntax requires that a wait statement only be used in a process without a sensitivity list. In the test case below, process has a sensitivity list with clk as a parameter which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (clk, d : in std_logic ;
          q : out std_logic);
end test;

architecture bhv of test is
begin
    process(clk)
    begin
        wait until clk'event and clk = '1' ;
        q <= d;
    end process;
end bhv;
```

Action

Remove the process sensitivity list if a wait statement is used in a process. For instance, modifying the example as follows removes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (clk, d : in std_logic ;
          q : out std_logic);
end test;
```

```
architecture bhv of test is
begin
    process
    begin
        wait until clk'event and clk = '1' ;
        q <= d;
    end process;
end bhv;
```

CD568

@E: Guarded blocks are not supported

A guarded block was encountered. In the following test case, the block statement contains a guarded assignment which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity test_guard is
port (A, en : in std_logic;
      Z : out std_logic);
end test_guard;

architecture bhv of test_guard is
begin
    B1 : block (en = '1')
    begin
        Z <= guarded not A;
    end block b1;
end bhv;
```

Action

Avoid using guarded block statements. Rewriting the architecture as a process with an if statement eliminates the error.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity test_guard is
port (A, en : in std_logic;
      Z : out std_logic);
end test_guard;

architecture bhv of test_guard is
begin
  process (en, A)
  begin
    if (en='1') then
      Z <= not A;
    end if;
  end process;
end bhv;
```

CD569

@E: Expecting 'after' keyword for waveform elements separated by a comma

The after keyword is missing from a comma-separated waveform element. The typical syntax for waveform elements is:

expression after time_expression, expression after time_expression [, ...]

In the test case below, signal temp is assigned '0','1' which is interpreted as two waveform elements. Because the keyword after is missing from the second waveform element, the compiler errors out.

```
library ieee;
use ieee.std_logic_1164.all;

entity test_waveform is
  port (A: in std_logic;
        B: in std_logic;
        C: out std_logic);
end test_waveform;
```

```
architecture rtl of test_waveform is
signal temp:std_logic;
begin
    temp<= '0' after 1 ns,'1' 2 ns;
    c<= a and b and temp;
end rtl;
```

Action

Make sure that the `after` keyword is included in each waveform element. In the corrected test case below, the `after` keyword is added to the second waveform element assigned to signal `temp`.

```
library ieee;
use ieee.std_logic_1164.all;

entity test_waveform is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic);
end test_waveform;

architecture rtl of test_waveform is
signal temp:std_logic;
begin
    temp<= '0' after 1 ns,'1' after 2 ns;
    c<= a and b and temp;
end rtl;
```

CD571

@E: Unsupported block configuration construct

An unsupported block configuration construct was encountered while parsing through a configuration declaration. In the test case below, the block configuration construct for instances U1 and U2 in configuration declaration `my_config` is not a supported construct which causes the error.

```
entity HA is
    port (U,V: in bit;
          X,Y: out bit);
end HA;
```

```
architecture str of Ha is
begin
    x<= u xor v;
    y<= u and v;
end str;

entity comb is
    port (A, B: in bit;
          SUM,SUM_1,CARRY, CARRY_1:out bit);
end comb;

architecture beh of comb is

component dummy
    port (A, B:in bit;
          SUM, CARRY: out bit);
end component;

begin
    U1: dummy port map (A, B, SUM,CARRY);
    U2: dummy port map ((not A), B,SUM_1,CARRY_1);
end beh;

configuration my_config of comb is
    for beh
        for u1: dummy use entity HA port map ( U => A, V => B,
                                                X => SUM,Y=>CARRY);
        for u2: dummy use entity HA port map ( U => A, V => B,
            X => SUM,Y=>CARRY);
        end for ;
    end for ;
end my_config;
```

Action

Be sure to use only supported constructs in a configuration declaration as shown in the corrected test case below.

```
entity HA is
    port (U,V: in bit;
          X,Y: out bit);
end HA;
```

```
architecture str of Ha is
begin
    x<= u xor v;
    y<= u and v;
end str;

entity comb is
    port (A, B: in bit;
          SUM,SUM_1,CARRY, CARRY_1:out bit);
end comb;

architecture beh of comb is

component dummy
    port (A, B:in bit;
          SUM, CARRY: out bit);
end component;

begin
    U1: dummy port map (A, B, SUM,CARRY);
    U2: dummy port map ((not A), B,SUM_1,CARRY_1);
end beh;

configuration my_config of comb is
    for beh
        for u1: dummy use entity HA port map ( U => A, V => B,
                                                X => SUM,Y=>CARRY);
        end for;
        for u2: dummy use entity HA port map ( U => A, V => B,
                                                X => SUM,Y=>CARRY);
        end for ;
    end for ;
end my_config;
```

CD573

@E: Identifier <fa> is not an architecture for <ex3>

A non-existing architecture was selected for an entity using a configuration statement. In the following test case, entity fa has two architectures of which one can be selected using a configuration statement. However, the configuration statement attempts to use non-existing architecture ex3 for fa which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity fa is
port (a,b,cin : in std_logic;
      sum,carry : out std_logic);
end fa;

architecture ex1 of fa is
begin
    sum    <= a xor b xor cin;
    carry <= ((a and b) or ( a and cin) or ( b and cin));
end ex1;

architecture ex2 of fa is
begin
    sum    <= a xor b xor cin;
    carry <= ((a and b) or (cin and (a xor b)));
end ex2;

configuration adder of fa is
    for ex3
        end for;
    end adder;
```

Action

Specifying an existing architecture in the configuration statement corrects the error. For the above example, modifying the configuration as shown below removes the error.

```
configuration adder of fa is
    for ex1
        end for;
    end adder;
```

CD576

@E: Component instantiation <module2> not found in current context.

A label in the list of component labels in a configuration specification statement is not a component instantiation label. The syntax for a configuration specification is:

```
for component_label_list : component_name binding_indication;
```

In the test case below, the label used in the configuration specification statement is MODULE2. However, because there is no component instantiated with this label in the architecture, the compiler errors out.

```
entity HA is
    port (U,V: in bit;
          X,Y: out bit);
end HA;

architecture GATE of HA is
begin
    x<= u xor v;
    y<= u and v;
end;

entity comb is
    port (A, B: in bit;
          SUM, CARRY: out bit);
end comb;

architecture STRUCT of comb is
component HALFADDER
    port (A, B: in bit;
          SUM, CARRY: out bit);
end component;

for MODULE2: HALFADDER use entity work.HA(GATE)
    port map (A, B,SUM,CARRY);
begin
MODULE1: HALFADDER
port map (A, B, SUM,CARRY);
end STRUCT;
```

Action

Be sure to use the labels in the component instantiation as *component_label_list* in the configuration specification statement. In the corrected test case below, the MODULE1 label used in the configuration specification statement is used as the label for instantiating the component.

```
entity HA is
    port (U,V: in bit;
          X,Y: out bit);
end HA;

architecture GATE of HA is
begin
    x<= u xor v;
    y<= u and v;
end;

entity comb is
    port (A, B: in bit;
          SUM, CARRY: out bit);
end comb;

architecture STRUCT of comb is
component HALFADDER
    port (A, B: in bit;
          SUM, CARRY: out bit);
end component;

for MODULE1: HALFADDER use entity work.HA(GATE)
    port map (A, B,SUM,CARRY);
begin
MODULE1: HALFADDER
port map (A, B, SUM,CARRY);
end STRUCT;
```

CD578

@E: Illegal redeclaration

An illegal redeclaration of a component instantiation statement was encountered. In the test case below, the same label (u1) is used in two different component instantiations which causes the error.

```

entity fa is
    port (a,b,cin : in bit ;
          sum, carry : out bit);
end fa;

architecture df of fa is
begin
    sum <= a xor b xor cin;
    carry <= ((a and b) or (a and cin) or (b and cin));
end;

entity fulladder is
generic( N : integer := 2);
    port (a,b : in bit_vector(N-1 downto 0);
          cin : in bit_vector(0 to 0);
          sum : out bit_vector(N-1 downto 0);
          carry : out bit);
end fulladder;

architecture structural of fulladder is
component fa
    port (a,b,cin : in bit;
          sum, carry : out bit);
end component;

signal temp : bit_vector(N downto 0);
begin
    temp(0) <= cin(0);
    u1 : fa port map(a=>a(0), b=>b(0),cin=>temp(0), sum=>sum(0),
                      carry=>temp(1));
    u1 : fa port map(a=>a(1), b=>b(1), cin=>temp(1),
                      sum=>sum(1), carry=>temp(2));
    carry<=temp(2);
end structural;

```

Action

Avoid using illegal redeclarations. A component instantiation statement can be redeclared only in a configuration declaration when providing missing information such as unassociated or open ports or generics. In the corrected test case below, there is no illegal redeclaration.

```

entity fa is
    port (a,b,cin : in bit ;
          sum, carry : out bit);
end fa;

```

```

architecture df of fa is
begin
    sum <= a xor b xor cin;
    carry <= ((a and b) or (a and cin) or (b and cin));
end;

entity fulladder is
generic( N : integer := 2);
port (a,b : in bit_vector(N-1 downto 0);
      cin : in bit_vector(0 to 0);
      sum : out bit_vector(N-1 downto 0);
      carry : out bit);
end fulladder;

architecture structural of fulladder is
component fa
port (a,b,cin : in bit;
      sum, carry : out bit);
end component;

signal temp : bit_vector(N downto 0);
begin
    temp(0) <= cin(0);
    u1 : fa port map(a=>a(0), b=>b(0),cin=>temp(0), sum=>sum(0),
                      carry=>temp(1));
    u2 : fa port map(a=>a(1), b=>b(1), cin=>temp(1), sum=>sum(1),
                      carry=>temp(2));
    carry<=temp(2);
end structural;

```

CD579

@E: Duplicate component instantiation label <u1>.

Duplicate labels appear in the list of component labels in a configuration specification statement. The syntax for a configuration specification is:

for component_label_list : component_name binding_indication;

In the test case below, label u1 appears twice in the *component_label_list* within the configuration specification statement which causes the error.

```

entity HA is
    port (U,V: in bit;
          X,Y: out bit);
end HA;

architecture beh of Ha is
begin
    x<= u xor v;
    y<= u and v;
end beh;

entity halfadder is
    port (A, B: in bit;
          SUM1, CARRY1: out bit;
          SUM, CARRY: out bit);
end halfadder;

architecture beh of halfadder is
component dummy
    port (A, B:in bit;
          SUM, CARRY: out bit);
end component;

for u1,u1: dummy use entity work.HA port map (A,B,SUM,CARRY);
begin
    u1: dummy port map (A=>A, B=>B,sum=>SUM,carry=>CARRY);
    u2:dummy port map (A,B,SUM1,CARRY1);
end beh;

```

Action

Make sure that unique labels are used in the *component_label_list*. In the corrected test case below, the *component_label_list* has two unique labels (u1,u2).

```

entity HA is
    port (U,V: in bit;
          X,Y: out bit);
end HA;

architecture beh of Ha is
begin
    x<= u xor v;
    y<= u and v;
end beh;

```

```

entity halfadder is
    port (A, B: in bit;
          SUM1, CARRY1: out bit;
          SUM, CARRY: out bit);
end halfadder;

architecture beh of halfadder is
component dummy
    port (A, B:in bit;
          SUM, CARRY: out bit);
end component;

for u1,u2: dummy use entity work.HA port map (A,B,SUM,carry);
begin
    u1: dummy port map (A=>A, B=>B,sum=>SUM,carry=>CARRY);
    u2:dummy port map (A,B,SUM1 ,CARRY1 );
end beh;

```

CD580

@E: Expecting identifier for component name

No valid identifier was specified as a *component_name* in the configuration specification statement. The syntax for a configuration specification is:

for *component_label_list* : *component_name binding indication*;

In the test case below, there is no identifier for *component_name* in the configuration specification statement which results in the error.

```

entity HA is
    port (U,V: in bit;
          X,Y: out bit);
end HA;

architecture beh of Ha is
begin
    x<= u xor v;
    y<= u and v;
end beh;

```

```
entity halfadder is
    port (A, B: in bit;
          SUM, CARRY: out bit);
end halfadder;

architecture beh of halfadder is
component dummy
    port (A, B:in bit;
          SUM, CARRY: out bit);
end component;

for u1:entity HA(beh) port map (A,B,SUM,CARRY);
begin
    u1: dummy port map (A, B, SUM, CARRY);
end beh;
```

Action

Make sure an identifier is specified for *component_name*. In the corrected test below, identifier dummy is specified as a *component_name* in the configuration specification statement.

```
entity HA is
    port (U,V: in bit;
          X,Y: out bit);
end HA;

architecture beh of Ha is
begin
    x<= u xor v;
    y<= u and v;
end beh;

entity halfadder is
    port (A, B: in bit;
          SUM, CARRY: out bit);
end halfadder;

architecture beh of halfadder is
component dummy
    port (A, B:in bit;
          SUM, CARRY: out bit);
end component;
```

```
for u1:dummy use entity HA(beh) port map (A,B,SUM,CARRY);
begin
    u1: dummy port map (A, B, SUM, CARRY);
end beh;
```

CD581

@E: Unable to find library <test>.

The referenced library does not exist. When binding components, the VHDL language requires the library to be explicitly defined when the component resides in a library other than the default library. The syntax is *library.package.component*. In the following test case, test is the library, pkg is the package, and MA is the component. Because the referenced library (test) does not exist, the compiler errors out.

```
library ieee;
use ieee.std_logic_1164.all;

package pkg is
    component MA
        port (A, B : in std_logic;
              Z : out std_logic);
    end component;
end package ;

entity and1 is
port (A1, A2 : in std_logic;
      y : out std_logic);
end and1;

architecture df of and1 is
begin
    y <= a1 and a2;
end df;

architecture df1 of and1 is
begin
    y <= a1 or a2;
end df1;
```

```
entity xor1 is
port (X1, X2 : in std_logic;
      Y : out std_logic);
end xor1;

architecture str of xor1 is
begin
  Y <= X1 xor X2;
end str;

entity ha is
port (a, b : in std_logic;
      sum, carry : out std_logic);
end ha;

architecture df of ha is
component MX
port (A, B : in std_logic;
      Z : out std_logic);
end component;

component MA
port (A, B : in std_logic;
      Z : out std_logic);
end component;

begin
  u0 : MX port map (a,b,sum);
  u1 : MB port map (a,b,carry);      ---- test.pkg.MB port map
                                         (a,b,carry); still gives error.
end;

configuration cl of ha is
  for df
    for u1 : test.pkg.MB use entity temp.and1(df1) port map
      (A1 => A, A2 => B, Y => Z);
    end for;
    for u0 : MX use entity xor1(str) port map
      (X1 => A, X2 => B, Y => Z);
    end for;
  end for;
end;
```

Action

Make sure that the VHDL library is set to where the component in the package is defined before compiling. In the above test case, setting the library to test (instead of the default library work) would eliminate the error.

CD582

@E: Unable to find package <pkg1>

The package specified could not be found in the referenced library. When binding components, the VHDL language requires the library to be explicitly defined when the component resides in a library other than the default library. The syntax is *library.package.component*. As an example, assume that work is the library, pkg1 is the package, and MB is the component. If the referenced package (pkg1) does not exist in library work, the compiler issues the above error message.

Action

Make sure that the package and library are in agreement before compiling. For the above example, component MB must reside in package pkg1 and must be compiled to library work to eliminate this error.

CD583

@E: Unable to find Package <pkg> in library <temp>

The package specified could not be found in the referenced library. When binding components, the VHDL language requires the library to be explicitly defined when the component resides in a library other than the default library. The syntax is *library.package.component*. As an example, assume that temp is the library and that pkg is the package. If the referenced package (pkg) exists, but not in the referenced library (temp), the compiler issues the above error message.

Action

Make sure that the VHDL library is set to where the component in the package is defined before compiling. In response to the above error message, selecting Project->Set VHDL Library and entering the appropriate library name would eliminate this error.

CD584

@E: Unable to find component <mb> in current context.

The component specified could not be found in the specified library and package. When binding components, the VHDL language requires the library to be explicitly defined when the component resides in a library other than the default library. The syntax is *library.package.component*. In the following test case, work is the library and pkg is the package. If component MB does not exist in package pkg in library work, the compiler errors out.

```
library ieee;
use ieee.std_logic_1164.all;

package pkg is
    component MA
        port (A, B : in std_logic;
              Z : out std_logic);
    end component;
end package ;

entity and1 is
    port (A1, A2 : in std_logic;
          y : out std_logic);
end and1;

architecture df of and1 is
begin
    y <= a1 and a2;
end df;

architecture df1 of and1 is
begin
    y <= a1 or a2;
end df1;
```

```
entity xor1 is
port (X1, X2 : in std_logic;
      Y : out std_logic);
end xor1;

architecture str of xor1 is
begin
    Y <= X1 xor X2;
end str;

entity ha is
port (a, b : in std_logic;
      sum, carry : out std_logic);
end ha;

architecture df of ha is
component MX
port (A, B : in std_logic;
      Z : out std_logic);
end component;

component MA
port (A, B : in std_logic;
      Z : out std_logic);
end component;

begin
    u0 : MX port map (a,b,sum);
    u1 : MB port map (a,b,carry);      ---- work.pkg.MB port map
                                         (a,b,carry); still gives error.
end;

configuration c1 of ha is
    for df
        for ul : work.pkg.MB use entity temp.and1(df1) port map
            (A1 => A, A2 => B, Y => Z);
        end for;
        for u0 : MX use entity xor1(str) port map
            (X1 => A, X2 => B, Y => Z);
        end for;
    end for;
end;
```

Action

Define the component to be bound to an entity within the architecture declarative region or in a package. Inserting the following code segment into the test case eliminates the error.

```
component MB
port (A, B : in std_logic;
      Z : out std_logic);
end component;
```

CD585

@E: Configuration specification FOR ALL: <mx> .. is already defined. Has to be the last specification for a component.

A for all statement appeared before a for instance statement in a configuration specification as shown in the code segment below.

```
for ALL:
  MX use entity xorl(str) port map ( X1 => A, X2 => B, Y => Z);
end for;

for u2:
  MX use entity xorl(str) port map ( X1 => A, X2 => B, Y => Z);
end for;
```

Action

Make sure that a for all statement follows a for statement in a configuration specification. Changing the order of the statements as shown below eliminates this syntax error.

```
for u2:
  MX use entity xorl(str) port map ( X1 => A, X2 => B, Y => Z);
end for;

for ALL:
  MX use entity xorl(str) port map ( X1 => A, X2 => B, Y => Z);
end for;
```

CD586

@E: Configuration specification FOR OTHERS: <mx> .. is already defined. Has to be the last specification for a component.

A for others statement appears before a for instance statement in a configuration specification as shown in the code segment below.

```
for others:  
    MX use entity xorl(str) port map (X1 => A, X2 => B, Y => Z);  
end for;  
  
for u0:  
    MX use entity xorl(str) port map (X1 => A, X2 => B, Y => Z);  
end for;
```

Action

Make sure that a for others statement follows a for statement in a configuration specification. Changing the order of the statements as shown below eliminates this syntax error.

```
for u2:  
    MX use entity xorl(str) port map (X1 => A, X2 => B, Y => Z);  
end for;  
  
for others:  
    MX use entity xorl(str) port map (X1 => A, X2 => B, Y => Z);  
end for;
```

CD587

@E: Expecting library or entity name.

An *entity_name* or *library_name* did not follow the keyword **entity** in the binding indication of a configuration specification statement. The syntax of a configuration specification is:

for component_label_list : component_name binding_indication;

In the syntax statement, *binding_indication* is of the form:

```
use entity entity_name [ (architecture_name) ]
[generic map (generic_association_list) ]
[port map (port_association_list) ]
```

In the test case below, an entity name following the keyword entity is missing which causes the compiler to error out.

```
entity HA is
    port (U,V: in bit;
          X,Y: out bit);
end HA;

architecture beh of Ha is
begin
    x<= u xor v;
    y<= u and v;
end beh;

entity halfadder is
    port (A, B: in bit;
          SUM, CARRY: out bit);
end halfadder;

architecture beh of halfadder is
component dummy
    port (A, B:in bit;
          SUM, CARRY: out bit);
end component;

for u1:dummy use entity .(beh) port map (A,B,SUM,CARRY);
begin
    u1: dummy port map (A, B, SUM,CARRY);
end beh;
```

Action

Be sure to specify an entity name in the configuration specification statement. In the corrected test case below, entity name HA is added to the configuration specification statement following the entity keyword.

```
entity HA is
    port (U,V: in bit;
          X,Y: out bit);
end HA;
```

```
architecture beh of Ha is
begin
    x<= u xor v;
    y<= u and v;
end beh;

entity halfadder is
    port (A, B: in bit;
          SUM, CARRY: out bit);
end halfadder;

architecture beh of halfadder is
component dummy
    port (A, B:in bit;
          SUM, CARRY: out bit);
end component;

for u1:dummy use entity HA(beh) port map (A,B,SUM,CARRY);
begin
    u1: dummy port map (A, B, SUM,CARRY);
end beh;
```

CD588

@E: Expecting entity name.

An entity declaration was incomplete. The syntax for an entity declaration is:

```
entity entity_name is port (port_name:port_direction port_type;...); end;
```

The entity declaration in the following test case is incomplete which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity
  (a, b, cin: in std_logic;
   sum, cout:out std_logic );
end adder;
```

```
architecture behave of adder is
begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;
```

Action

Make sure that the entity declaration contains a user-defined entity name and the correct VHDL syntax as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity adder is
    port (a, b, cin: in std_logic;
          sum, cout:out std_logic );
end adder;

architecture behave of adder is
begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;
```

CD589

@E: Expecting architecture name.

An architecture name was not specified during component binding. In the following code segment, instance u2 of component MX is bound to entity xor1 whose architecture is not specified within its () parentheses which results in the above error.

```
for u2 : MX use entity xor1() port map (X1 => A, X2 => B, Y => Z);
end for;
```

Action

Including the architecture name within the parentheses as shown in the first code segment or removing the parentheses as shown in the second code segment eliminates the error. Note that if the parentheses are omitted, the compiler uses the last compiled architecture of that entity.

```
for u2 : MX use entity xor1(str) port map
  (X1 => A, X2 => B, Y => Z);
end for;

for u2 : MX use entity xor1 port map
  (X1 => A, X2 => B, Y => Z);
end for;
```

CD590

@E: Instantiated entity <and1> has not been analyzed.

An indicated entity could not be found during component instantiation and binding. In the following example, xor1 is the lower level entity, and ha is the top-level entity. The top-level entity has two components – MA and MX. Component MA is bound to entity and1 and component MX is bound to xor1. However, lower level entity and1 is missing which results in the error – instantiated entity and1 has not been analyzed. The above error also occurs if the top-level entity is compiled before the lower level entity.

```
library ieee;
use ieee.std_logic_1164.all;

entity xor1 is
port (X1, X2 : in std_logic;
      Y : out std_logic);
end xor1;

architecture str of xor1 is
begin
  Y <= X1 xor X2;
end str;

library ieee;
use ieee.std_logic_1164.all;
```

```

entity ha is
port (a, b : in std_logic;
      sum, carry : out std_logic);
end ha;

architecture df of ha is
component MX
port (A, B : in std_logic;
      Z : out std_logic);
end component;

component MA
port (A, B : in std_logic;
      Z : out std_logic);
end component;

begin
  u0 : MX port map (a,b,sum);
  u1 : MA port map (a,b,carry);
end;

configuration cl of ha is
  for df
    for u1 : MA use entity and1(df) port map
      ( A1 => A, A2 => B, Y => Z);
    end for;
    for u0 : MX use entity xor1(str) port map
      ( X1 => A, X2 => B, Y => Z);
    end for;
  end for;
end;

```

Action

Include all lower level entities before compiling the top-level entity. Also, make sure that all lower level entities are covered before the top-level entity. For the above example, include the entity `and1` as shown in the following example.

```

library ieee;
use ieee.std_logic_1164.all;

entity and1 is
port (A1, A2 : in std_logic;
      y : out std_logic);
end and1;

```

```
architecture df of and1 is
begin
    y <= a1 and a2;
end df;

library ieee;
use ieee.std_logic_1164.all;

entity xor1 is
port (X1, X2 : in std_logic;
      Y : out std_logic);
end xor1;

architecture str of xor1 is
begin
    Y <= X1 xor X2;
end str;

library ieee;
use ieee.std_logic_1164.all;

entity ha is
port (a, b : in std_logic;
      sum, carry : out std_logic);
end ha;

architecture df of ha is
component MX
port (A, B : in std_logic;
      Z : out std_logic);
end component;

component MA
port (A, B : in std_logic;
      Z : out std_logic);
end component;

begin
    u0 : MX port map (a,b,sum);
    u1 : MA port map (a,b,carry);
end;

configuration cl of ha is
    for df
        for u1 : MA use entity and1(df) port map
            (A1 => A, A2 => B, Y => Z);
        end for;
    end for;

```

```
for u0 : MX use entity xor1(str) port map
  (X1 => A, X2 => B, Y => Z);
end for;
end for;
end;
```

CD591

@E: Object <temp> is not a library.

An incorrect library was specified during component binding or the component to be bound was not compiled into that library. The following code segment illustrates this error.

```
configuration cl of ha is
  for df
    for u1 : MA use entity temp.and1(df) port map
      (A1 => A, A2 => B, Y => Z);
    end for;
    for u0 : MX use entity xor1(str) port map
      (X1 => A, X2 => B, Y => Z);
    end for;
  end for;
end;
```

In the above segment, temp is the target library where entity and1 with architecture df is to be compiled. If entity and1 with architecture df is not compiled into this library, the compiler errors out.

Action

Specify the library where a particular component must be compiled using Set VHDL Library options under the Project menu to eliminate the error.

CD593

@E: Instantiated entity <and1> has no architecture <df1> specified

An entity was instantiated by component binding, but its architecture was not specified. In the following test case, entity and1 is defined, but its architecture is missing. The top level binds instance u1 to entity and1 with architecture df1, but because the compiler cannot find architecture df1 of and1, it reports the above error.

```
library ieee;
use ieee.std_logic_1164.all;

entity and1 is
port (A1, A2 : in std_logic;
      y : out std_logic);
end and1;

architecture df of and1 is
begin
    y <= a1 and a2;
end df;

entity xor1 is
port (X1, X2 : in std_logic;
      Y : out std_logic);
end xor1;

architecture str of xor1 is
begin
    Y <= X1 xor X2;
end str;

entity ha is
port (a, b : in std_logic;
      sum, carry : out std_logic);
end ha;

architecture df of ha is
component MX
port (A, B : in std_logic;
      Z : out std_logic);
end component;
```

```

component MA
port (A, B : in std_logic;
      Z : out std_logic);
end component;

begin
  u0 : MX port map (a,b,sum);
  u1 : MA port map (a,b,carry);
end;

configuration c1 of ha is
  for df
    for u1 : MA use entity temp.and1(df1) port map
      ( A1 => A, A2 => B, Y => Z);
    end for;
    for u0 : MX use entity xor1(str) port map
      ( X1 => A, X2 => B, Y => Z);
    end for;
  end for;
end;

```

Action

Specify the appropriate architecture of an entity before binding it to an instance. For the above test case, inclusion of the following architecture eliminates the error.

```

architecture df1 of and1 is
begin
  y <= a1 or a2;
end df1;

```

CD594

@E: Expecting library or configuration name.

Neither *configuration_name* nor *library.configuration_name* was specified in the binding indication of a configuration specification statement. The general syntax of a configuration specification is:

for component_label_list : component_name binding_indication;

In the above syntax, *binding_indication* is of the form:

use configuration *configuration_name* | *library.configuration_name*

In the test case below, a configuration name is not specified in the binding indication for instance u1 which results in the error.

```
entity fa is
    port (a,b,cin : in bit;
          sum,carry : out bit);
end fa;

architecture adder of fa is
begin
    sum <= a xor b xor cin;
    carry <= ((a and b) or ( a and cin) or ( b and cin));
end adder;

architecture sub of fa is
begin
    sum <= a xor b xor cin;
    carry <= (not a) and ( b or cin);
end sub;

configuration adder_config of fa is
    for adder
        end for;
    end adder_config;

configuration sub_config of fa is
    for sub
        end for;
    end sub_config;

entity fulladder is
    port (a,b,cin : in bit;
          sum,carry : out bit);
end fulladder;

architecture test of fulladder is
component fa is
    port (a,b,cin : in bit;
          sum,carry : out bit);
end component;
```

```
for ul: fa use configuration;
begin
    ul: fa port map (a,b,cin,sum,carry);
end test;
```

Action

Be sure to specify either a configuration name or a library configuration name in the use configuration statement. In the test case below, specifying work.adder_config as the configuration name corrects the error.

```
entity fa is
    port (a,b,cin : in bit;
          sum,carry : out bit);
end fa;

architecture adder of fa is
begin
    sum <= a xor b xor cin;
    carry <= ((a and b) or ( a and cin) or ( b and cin));
end adder;

architecture sub of fa is
begin
    sum <= a xor b xor cin;
    carry <= (not a) and ( b or cin);
end sub;

configuration adder_config of fa is
    for adder
        end for;
    end adder_config;

configuration sub_config of fa is
    for sub
        end for;
    end sub_config;

entity fulladder is
    port (a,b,cin : in bit;
          sum,carry : out bit);
end fulladder;
```

```

architecture test of fulladder is
component fa is
    port (a,b,cin : in bit;
          sum,carry : out bit);
end component;

for u1: fa use configuration work.adder_config ;
begin
    u1: fa port map (a,b,cin,sum,carry);
end test;

```

CD595

@E: Expecting configuration name.

The library name prefix (*library*) was specified for *library.configuration_name* without including the actual configuration name in the binding indication of the configuration specification statement. The general syntax of a configuration specification is:

for component_label_list : component_name binding_indication;

In the above syntax, *binding_indication* is of the form:

use configuration configuration_name | library.configuration_name

In the configuration specification for instance u1 in the test case below, the incomplete *library.configuration_name* argument (“work.” without a configuration name) results in the error.

```

entity fa is
    port (a,b,cin : in bit;
          sum,carry : out bit);
end fa;

architecture adder of fa is
begin
    sum <= a xor b xor cin;
    carry <= ((a and b) or ( a and cin) or ( b and cin));
end adder;

```

```

architecture sub of fa is
begin
    sum <= a xor b xor cin;
    carry <= (not a) and ( b or cin);
end sub;

configuration adder_config of fa is
    for adder
        end for;
    end adder_config;

configuration sub_config of fa is
    for sub
        end for;
    end sub_config;

entity fulladder is
    port (a,b,cin : in bit;
          sum,carry : out bit);
end fulladder;

architecture test of fulladder is
component fa is
    port (a,b,cin : in bit;
          sum,carry : out bit);
end component;

for u1: fa use configuration work.;
begin
    u1: fa port map (a,b,cin,sum,carry);
end test;

```

Action

Be sure to specify only a configuration name string or a complete *library.configuration_name* string in a use configuration statement. In the test case below, specifying work.adder_config in the use configuration statement corrects the error.

```

entity fa is
    port (a,b,cin : in bit;
          sum,carry : out bit);
end fa;

```

```
architecture adder of fa is
begin
    sum <= a xor b xor cin;
    carry <= ((a and b) or ( a and cin) or ( b and cin));
end adder;

architecture sub of fa is
begin
    sum <= a xor b xor cin;
    carry <= (not a) and ( b or cin);
end sub;

configuration adder_config of fa is
    for adder
        end for;
    end adder_config;

configuration sub_config of fa is
    for sub
        end for;
    end sub_config;

entity fulladder is
    port (a,b,cin : in bit;
          sum,carry : out bit);
end fulladder;

architecture test of fulladder is
component fa is
    port (a,b,cin : in bit;
          sum,carry : out bit);
end component;

for u1: fa use configuration work.adder_config;
begin
    u1: fa port map (a,b,cin,sum,carry);
end test;
```

CD596

@E: Cannot find configuration <adder_config1>.

The configuration specified by *configuration_name* or *library.configuration_name* in the binding indication of a configuration specification could not be located. The general syntax of a configuration specification is:

```
for component_label_list : component_name binding_indication;
```

In the above syntax, *binding_indication* is of the form:

```
use configuration configuration_name | library.configuration_name
```

In the test case below, configuration name work.adder_config1 in the configuration specification for instance u1 does not exist in library work which results in the error.

```
entity fa is
    port (a,b,cin : in bit;
          sum,carry : out bit);
end fa;

architecture adder of fa is
begin
    sum <= a xor b xor cin;
    carry <= ((a and b) or ( a and cin) or ( b and cin));
end adder;

architecture sub of fa is
begin
    sum <= a xor b xor cin;
    carry <= (not a) and ( b or cin);
end sub;

configuration adder_config of fa is
    for adder
        end for;
    end adder_config;

configuration sub_config of fa is
    for sub
        end for;
    end sub_config;
```

```
entity fulladder is
    port (a,b,cin : in bit;
          sum,carry : out bit);
end fulladder;

architecture test of fulladder is
component fa is
    port (a,b,cin : in bit;
          sum,carry : out bit);
end component;

for u1: fa use configuration work.adder_config1;
begin
    u1: fa port map (a,b,cin,sum,carry);
end test;
```

Action

Be sure to specify a valid configuration name for the current context. In the corrected test case below, the name of the configuration in the use configuration statement is specified as work.adder_config which is a declared configuration.

```
entity fa is
    port (a,b,cin : in bit;
          sum,carry : out bit);
end fa;

architecture adder of fa is
begin
    sum <= a xor b xor cin;
    carry <= ((a and b) or ( a and cin) or ( b and cin));
end adder;

architecture sub of fa is
begin
    sum <= a xor b xor cin;
    carry <= (not a) and ( b or cin);
end sub;

configuration adder_config of fa is
    for adder
    end for;
end adder_config;
```

```
configuration sub_config of fa is
    for sub
        end for;
end sub_config;

entity fulladder is
    port (a,b,cin : in bit;
          sum,carry : out bit);
end fulladder;

architecture test of fulladder is
component fa is
    port (a,b,cin : in bit;
          sum,carry : out bit);
end component;

for u1: fa use configuration work.adder_config ;
begin
    u1: fa port map (a,b,cin,sum,carry);
end test;
```

CD597

@E: Cannot find configuration <ieee.adder_config>.

The named configuration could not be found in the library specified by *library.configuration_name*. The general syntax of a configuration specification is:

for component_label_list : component_name binding_indication;

In the above syntax, *binding_indication* is of the form:

use configuration configuration_name | library.configuration_name

In the configuration specification for instance u1 in the test case below, the configuration name is specified as ieee.adder_config. However, because this configuration does not exist in library ieee, the compiler errors out.

```
entity fa is
    port (a,b,cin : in bit;
          sum,carry : out bit);
end fa;

architecture adder of fa is
begin
    sum <= a xor b xor cin;
    carry <= ((a and b) or ( a and cin) or ( b and cin));
end adder;

architecture sub of fa is
begin
    sum <= a xor b xor cin;
    carry <= (not a) and ( b or cin);
end sub;

configuration adder_config of fa is
    for adder
        end for;
    end adder_config;

configuration sub_config of fa is
    for sub
        end for;
    end sub_config;

entity fulladder is
    port (a,b,cin : in bit;
          sum,carry : out bit);
end fulladder;

architecture test of fulladder is
component fa is
    port (a,b,cin : in bit;
          sum,carry : out bit);
end component;

for u1: fa use configuration ieee.adder_config;
begin
    u1: fa port map (a,b,cin,sum,carry);
end test;
```

Action

Be sure to use only configurations that are declared in a library in the binding indication of the configuration specification statement. In the corrected test case below, the name of the configuration is specified as work.adder_config (configuration adder_config is assumed to be declared in library work).

```
entity fa is
    port (a,b,cin : in bit;
          sum,carry : out bit);
end fa;

architecture adder of fa is
begin
    sum <= a xor b xor cin;
    carry <= ((a and b) or ( a and cin) or ( b and cin));
end adder;

architecture sub of fa is
begin
    sum <= a xor b xor cin;
    carry <= (not a) and ( b or cin);
end sub;

configuration adder_config of fa is
    for adder
        end for;
    end adder_config;

configuration sub_config of fa is
    for sub
        end for;
    end sub_config;

entity fulladder is
    port (a,b,cin : in bit;
          sum,carry : out bit);
end fulladder;

architecture test of fulladder is
component fa is
    port (a,b,cin : in bit;
          sum,carry : out bit);
    end component;
```

```

for ul: fa use configuration work.adder_config;
begin
    ul: fa port map (a,b,cin,sum,carry);
end test;

```

CD598

@E: Generic is not specified at component decl.

A generic was not specified in a component declaration. In the test case below, attempting to map generic t in the configuration specification (which is not specified in the component declaration) causes the error.

```

entity HA is
generic (k: integer:=4);
port (a,b: in bit_vector(k-1 downto 0);
      cin: in bit_vector(k-1 downto 0);
      s: out bit_vector(k-1 downto 0);
      cout: out bit_vector(k-1 downto 0));
end HA;

architecture beh of Ha is
begin
    s<= a xor b xor cin;
    cout<=(a and b)or (a and cin) or (b and cin);
end beh;

entity halfadder is
port (a,b: in bit_vector(3 downto 0);
      cin: in bit_vector (3 downto 0);
      s: out bit_vector(3 downto 0);
      cout: out bit_vector(3 downto 0));
end halfadder;

architecture beh of halfadder is
component HA1 is
port (a,b: in bit_vector(3 downto 0);
      cin: in bit_vector (3 downto 0);
      s: out bit_vector(3 downto 0);
      cout: out bit_vector(3 downto 0));
end component;

```

```

for u1: hal use entity work.ha generic map (t)
    port map (a,b,cin,cout);
begin
    u1: HA1  port map (a,b,cin,s,cout);
end beh;

```

Action

Be sure to use the generics specified in the component declaration for generic mapping in the configuration specification. In the corrected test case below, the generic is removed from component HA1 to prevent generic mapping from occurring in the configuration specification.

```

entity HA is
generic (k: integer:=4);
port (a,b: in bit_vector(k-1 downto 0);
      cin: in bit_vector(k-1 downto 0);
      s: out bit_vector(k-1 downto 0);
      cout: out bit_vector(k-1 downto 0));
end HA;

architecture beh of Ha is
begin
    s<= a xor b xor cin;
    cout<=(a and b)or (a and cin) or (b and cin);
end beh;

entity halfadder is
port (a,b: in bit_vector(3 downto 0);
      cin: in bit_vector (3 downto 0);
      s: out bit_vector(3 downto 0);
      cout: out bit_vector(3 downto 0));
end halfadder;

architecture beh of halfadder is
component HA1 is
port (a,b: in bit_vector(3 downto 0);
      cin: in bit_vector (3 downto 0);
      s: out bit_vector(3 downto 0);
      cout: out bit_vector(3 downto 0));
end component;

for u1: hal use entity work.ha  port map (a,b,cin,cout);
begin
    u1: HA1  port map (a,b,cin,s,cout);
end beh;

```

CD602

@E: Illegal association (no associations may follow an others association)

Assignments follow an assignment using others. VHDL language semantics require that an others statement be the last statement in an assignment. In the following test case, the others => '0' statement is not the last statement in the assignment which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity test is
port (clk , reset : in std_logic;
      a , b : in std_logic_vector(15 downto 0);
      q : out std_logic_vector(31 downto 0));
end test;

architecture bhv of test is
begin
  process(clk, reset)
  begin
    if(reset = '1') then
      q <= (others => '0', '1') ;
    elsif(clk'event and clk = '1') then
      q <= a * b;
    end if;
  end process;
end bhv;
```

Action

Make sure that the others statement is the last statement in an assignment as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
```

```

entity test is
port (clk , reset : in std_logic;
      a , b : in std_logic_vector(15 downto 0);
      q : out std_logic_vector(31 downto 0));
end test;

architecture bhv of test is
begin
  process(clk, reset)
  begin
    if(reset = '1') then
      q <= ('1', others => '0') ;
    elsif(clk'event and clk = '1') then
      q <= a * b;
    end if;
  end process;
end bhv;

```

CD603

@W: Variable <v> read before being assigned? This may cause a simulation mismatch

A variable is being used (read) before it is assigned a value. The statements in a process are sequential. Because simulation tools can schedule events, the simulation can be controlled to always assign a variable before it is read. Because there is no scheduling in synthesis, if the assignment statements are out of order, the variable would be read before its is assigned a value, and post synthesis simulations may not meet RTL simulations.

Note that if the variable is read under an event-triggered block, the compiler will not report any warning; in a combinational process; the warning is more appropriate as described in the following three examples.

Example 1

In the example below, the intention is to build two registers, register o with an asynchronous load and register v with an asynchronous reset.

```

library ieee;
use ieee.std_logic_1164.all;

```

```
entity test1 is port
  (reset,clk: in std_logic;
   o: out std_logic);
end entity;

architecture bhv of test1 is
begin
  process (reset,clk)
  variable v: std_logic;
  begin
    if reset='1' then
      o <= v;
      v := '0';
    elsif clk'event and clk='1' then
      o <= v;
      v := not v;
    end if;
  end process;
end bhv;
```

In the above example, the mismatch occurs because the process is not retriggered when v changes value during reset. Activation of reset causes o to load value v before reset; in the synthesized design, v is loaded after reset. The problem is that when the variable is read in a combinational path in the process, the read is not guarded by the 'event expression.

The mismatch occurs when the variable is read in a combinational path without writing it first which causes the synthesis tool to generate a feedback path from the final value of the variable after process execution to the input of the process. Whenever the variable is updated, the combinational parts of the synthesized hardware that read the variable are re-evaluated because of the changing input, but the simulator does not retrigger the process when the variable is updated.

Example 2

In the example below, the process causes a mismatch when a and b are both TRUE. The synthesis tool, in determining that v is identical to b, builds o as a latch holding the value of v (=b) when a is TRUE.

```
library ieee;
use ieee.std_logic_1164.all;
```

```

entity test1 is port
  (a,b: in std_logic;
   o: out std_logic );
end entity;

architecture bhv of test1 is
begin
  process(a,b)
  variable v: std_logic;
  begin
    if a='1' then
      o <= v;
    end if;
    if b='1' then
      v := '1';
    else
      v := '0';
    end if;
  end process;
end bhv;

```

The mismatch occurs when **a** and **b** are both TRUE. The synthesis tool latches TRUE (since **b** is TRUE) but, because the read is made in a combinational path of the process, the simulator latches the old value of **v** and proceeds to set **v** to TRUE.

Example 3

In the following example, the process describes both a clocked and a combinational behavior.

```

library ieee;
use ieee.std_logic_1164.all;

entity test1 is port
  (a,clk: in std_logic;
   o: out std_logic );
end entity;

architecture bhv of test1 is
begin
  process(clk,a)
  variable v: std_logic;
  begin
    if a ='1' then
      o <= v;

```

```

        end if;
        if clk'event and clk='1' then
            v := not v;
        end if;
    end process;
end bhv;

```

In the above example, if `a` is not a reset branch, `o` is synthesized as a latch with `v` as an input (`v` is a toggling flip-flop). If a clock edge occurs while `a` is TRUE, `v` is clocked and the new value of `v` is propagated through the latch. The simulator latches the old value and updates `v` afterwards. The problem is again that the read before (or without) write occurs in a combinational part of the process.

Action

Make sure that RTL descriptions do not vary between simulation and synthesis which could change the design functionality.

The Synopsys FPGA synthesis tools give the above warning message whenever a variable is read before it is assigned. The intention of the message is to make you aware of the possibility of a mismatch; each case must be individually examined to determine if an actual mismatch exists. In the test case below, there is no mismatch; simply reading the variable before it is assigned results in the message.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity Test is
    port (Clk : in std_logic;
          A,B,D : in std_logic;
          C : out std_logic );
end entity;

architecture RTL of Test is
begin
    process(Clk)
        variable res : std_logic;
        variable d_r : std_logic;
    begin
        res := A xor B xor d_r;
        if rising_edge(Clk) then

```

```

        C <= res;
        d_r := D;
    end if;
end process;
end architecture;

```

CD604

@W: OTHERS clause is not synthesized

An others clause was not synthesized for any of the following reasons:

- The case object could not functionally accept a value other than the explicitly defined values.
- When no transition in a state machine lead to an undefined state.
- When a case statement in a state machine was complete.

When a case object is an enumerated type with a list of values that the case object can take, if the case statement does not use all of the cases or if the state machine has no transitions to other states, the others clause is not synthesized. In the following test case, because identifier arg is a 2-bit vector with all four possible cases defined, the others clause is not synthesized as indicated by the warning message.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (arg : in std_logic_vector(1 downto 0);
          result : out std_logic_vector(2 downto 0));
end test;

architecture beh of test is
begin
    process(arg)
    begin
        case Arg is
            when "00" => Result <= "100";
            when "10" => Result <= "101";
            when "01" => Result <= "110";
            when others => Result <= "111";
        end case;
    end process;
end beh;

```

```
        when "11" => Result <= "111";
        when others => Result <= "000";
    end case;
end process;
end beh;
```

Action

Make sure that an others clause is used only when all possibilities are not covered so that a default assignment for the uncovered cases can be implemented by an others clause. Note that even when all cases are covered, a statement of NULL at the end of the others clause as shown in the modified test case below eliminates the warning.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (arg : in std_logic_vector(1 downto 0);
          result : out std_logic_vector(2 downto 0));
end test;

architecture beh of test is
begin
    process(arg)
    begin
        case Arg is
            when "00" => Result <= "100";
            when "10" => Result <= "101";
            when "01" => Result <= "110";
            when "11" => Result <= "111";
            when others => NULL;
        end case;
    end process;
end beh;
```

CD605

@E: Top level entity <itop> has no ports

No ports were defined for an entity. In the following test case, entity itop does not contain any ports which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity itop is
end entity;

architecture bhv of itop is
begin
end bhv;
```

Action

Specify the ports in the entity to eliminate the error.

CD607

@E: Identifier <ieee> is not declared

An undefined identifier was used as a library name in the use clause *library.package.all*. In the test case below, identifier ieee is not declared as a library which results in the error.

```
use ieee.std_logic_1164.all;
entity comb is
    port (a,b,cin : in std_logic;
          sum,cout : out std_logic);
end comb;

architecture beh of comb is
begin
    sum <= a xor b xor cin;
    cout <= ((a and b) or (cin and (a or b))) ;
end beh;
```

Action

Be sure to declare the identifier as library before using it as a library name. In the corrected test case below, identifier ieee is declared as a library.

```
library ieee;
use ieee.std_logic_1164.all;
```

```

entity comb is
    port (a,b,cin : in std_logic;
          sum,cout : out std_logic);
end comb;

architecture beh of comb is
begin
    sum <= a xor b xor cin;
    cout <= ((a and b) or (cin and (a or b))) ;
end beh;

```

CD609

@W: Index value <0> to <4> could be out of prefix range <0> to <3>

The index used to access a multidimensional array has a range that exceeds the defined range of the array. In the test case below, my_mem is a 2-dimensional array with a range of 0 to 3 whereas index ptr is declared with a range of 0 to 4 which results in the warning.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (clk,rst: in std_logic;
          din : in std_logic_vector(3 downto 0);
          dout: out std_logic_vector(3 downto 0) );
end ;

architecture rtll of test is
type mem  is array (0 to 3) of std_logic_vector(3 downto 0);
signal my_mem:mem;
signal ptr :natural range 0 to 4;
signal count : integer range 3 downto 0;
begin
    process(rst,clk)
begin
    if rst='0' then
        dout<="0000";
        ptr<=1;
        count<=0;
    elsif rising_edge(clk) then
        if (count<3) then

```

```

my_mem(ptr)<=din;
dout<=my_mem(ptr-1);
count<=count+1;
ptr<=count;
end if;
end if;
end process;
end;

```

Action

Make sure that the index used for accessing an array has the same range as that of the array as shown in the corrected test case where both the index and array have a range of 0 to 3.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (clk,rst: in std_logic;
          din : in std_logic_vector(3 downto 0);
          dout: out std_logic_vector(3 downto 0) );
end ;

architecture rtl1 of test is
type mem  is array (0 to 3) of std_logic_vector(3 downto 0);
signal my_mem:mem;
signal ptr :natural range 0 to 3;
signal count : integer range 3 downto 0;
begin
process(rst,clk)
begin
    if rst='0' then
        dout<="0000";
        ptr<=1;
        count<=0;
    elsif rising_edge(clk) then
        if (count<3) then
            my_mem(ptr)<=din;
            dout<=my_mem(ptr-1);
            count<=count+1;
            ptr<=count;
        end if;
    end if;
end process;
end;

```

CHAPTER 11

CD Messages 610 – 921

CD610

@W: Index value <0> to <4> could be out of prefix range <3> downto <0>

The index used to access a multidimensional array has a range that exceeds the defined range of the array. In the test case below, my_mem is a 2-dimensional array with a range of 3 downto 0 whereas index ptr is declared with a range of 0 to 4 which results in the above warning.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (clk,rst: in std_logic;
          din : in std_logic_vector(3 downto 0);
          dout: out std_logic_vector(3 downto 0) );
end ;

architecture rtl1 of test is
type mem  is array (3 downto 0) of std_logic_vector(3 downto 0);
signal my_mem:mem;
signal ptr :natural range 0 to 4;
signal count : integer range 3 downto 0;
begin
    process(rst,clk)
```

```

begin
    if rst='0' then
        dout<="0000";
        ptr<=1;
        count<=0;
    elsif rising_edge(clk) then
        if (count<3) then
            my_mem(ptr)<=din;
            dout<=my_mem(ptr-1);
            count<=count+1;
            ptr<=count;
        end if;
    end if;
end process;
end;

```

Action

Make sure that the index used for accessing an array has the same range as that of the array. In the corrected test case below, both the index and array have a similar range of 3 downto 0 and 0 to 3.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (clk,rst: in std_logic;
          din : in std_logic_vector(3 downto 0);
          dout: out std_logic_vector(3 downto 0) );
end ;

architecture rtl1 of test is
type mem  is array (0 to 3) of std_logic_vector(3 downto 0);
signal my_mem:mem;
signal ptr :natural range 0 to 3;
signal count : integer range 3 downto 0;
begin
    process(rst,clk)
    begin
        if rst='0' then
            dout<="0000";
            ptr<=1;
            count<=0;
        elsif rising_edge(clk) then
            if (count<3) then
                my_mem(ptr)<=din;

```

```

        dout<=my_mem(ptr-1);
        count<=count+1;
        ptr<=count;
    end if;
end if;
end process;
end;

```

CD614

@E: separator required between number and identifier

The compiler expected a number and found a number followed by an identifier or a keyword without an intervening space. In the following test case, the compiler expects a number after the keyword ‘in’ and errors out when it encounters the string 0to.

```

GEN: for i in 0to (N - 1) generate
  u1 : fa port
    map(a=>a(i), b=>b(i), cin=>temp(i), sum=>sum(i),
        carry=>temp(i+1));
  end generate;

```

Action

Make sure that a space is included between a number and an identifier or a keyword wherever a number is expected. In the above test case, adding a space between the numeral 0 and the keyword ‘to’ eliminates the error.

CD615

@W: physical integer constant overflowed limit of 2147483647 (when converted to base units)

The integer value of the secondary unit overflows the limit of 2147483647 (maximum 32-bit positive integer) when a physical type is being converted using a *base-unit* multiplication factor. A physical type contains values that

represent a measurement of some physical quantity such as time or voltage. Values of this type are expressed as integer multiples of a base unit. In the test case below, when the *secondary-unit* ms is converted in terms of base unit fs, it becomes 1000,000,000,000fs which overflows the integer value limit of 2,147,483,647 and results in the warning.

```

library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic);
end comb;

architecture rtl of comb is
type TIME is range -2147483647 to 2147483647
units
    fs;           -- femtosecond
    ps = 1000 fs; -- picosecond
    ns = 1000 ps; -- nanosecond
    us = 1000 ns; -- microsecond
    ms = 1000 us; -- millisecond
end units;

begin
    c <= a and b;
end rtl;

```

Action

Make sure that the physical unit range is defined so that the integer values do not overflow the maximum limit achievable with 32 bits. In the corrected test case below, the physical units remain within the 2147483647 limit when converted in terms of base units.

```

library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (A: in std_logic;
          B: in std_logic;
          C: out std_logic);
end comb;

```

```
architecture rtl of comb is
type TIME is range -2147483647 to 2147483647
units
    fs;          -- femtosecond
    ps = 1000 fs; -- picosecond
    ns = 1000 ps; -- nanosecond
    us = 1000 ns; -- microsecond
end units;

begin
    c <= a and b;
end rtl;
```

CD617

@E: Direction must be "in"

The subprogram specification for a function specified a mode other than in. In the *parameter_list*, the only supported mode for parameters is in (the default). The general syntax of a subprogram specification for a function body is:

```
function function_name (parameter_list)
    return return_type
```

In the test case below, parameter b is declared as mode out which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0));
end seq;
```

```

architecture beh of seq is
function my_and (a: std_logic_vector; b: out std_logic_vector)
    return std_logic_vector
is variable temp: std_logic_vector(a'range);
begin
    temp:= a and b;
    return temp;
end my_and;

begin
process (clk)
begin
    if falling_edge(clk) then
        d_out <= my_and(a_in, b_in);
    end if;
end process;
end beh;

```

Action

Make sure that all parameters are declared as mode in. In the corrected test case below, parameters a and b are implicitly declared as mode in (the default).

```

library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0));
end seq;

architecture beh of seq is
function my_and (a: std_logic_vector; b: std_logic_vector)
    return std_logic_vector
is variable temp: std_logic_vector(a'range);
begin
    temp:= a and b;
    return temp;
end my_and;

```

```
begin
    process (clk)
    begin
        if falling_edge(clk) then
            d_out <= my_and(a_in, b_in);
        end if;
    end process;
end beh;
```

CD621

@E: Attribute <left> is not supported on object

An attribute was applied to an object that does not support that attribute. Attributes such as 'left', 'right', 'high', 'low', and 'ascending' can be applied on scalar types or subtypes. In the test case below, attribute 'left' is applied to signal mem which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb_attribute is
    port (A: in std_logic_vector(1 downto 0);
          B: in std_logic_vector(1 downto 0);
          C: out std_logic_vector(1 downto 0));
end comb_attribute;

architecture rtl of comb_attribute is
subtype memory is integer range 0 to 1;
signal mem : memory;
begin
    c(mem'left)<=a(0) and b(1);
    C(1) <= a(1) and b(0);
end rtl;
```

Action

Be sure to apply attributes only to the types that support them. In the corrected test case below, attribute 'left' is applied to the scalar subtype memory.

```

library ieee;
use ieee.std_logic_1164.all;

entity comb_attribute is
    port (A: in std_logic_vector(1 downto 0);
          B: in std_logic_vector(1 downto 0);
          C: out std_logic_vector(1 downto 0) );
end comb_attribute;

architecture rtl of comb_attribute is
subtype memory is integer range 0 to 1;
begin
    c(memory'left)<=a(0) and b(1);
    C(1) <= a(1) and b(0);
end rtl;

```

CD622

@E: Actual expression must be static

The actual expression in a component instantiation statement is not static. In the test case below, the actual expression (P or Q) is not a static expression which results in the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity and_gate is
    port (a,b : in std_logic_vector(1 downto 0);
          c: out std_logic_vector(1 downto 0));
end and_gate;

architecture beh of and_gate is
begin
    c<= a and b;
end beh;

library ieee;
use ieee.std_logic_1164.all;

```

```
entity comb is
    port (P,Q : in std_logic_vector(1 downto 0);
          R: out std_logic_vector(1  downto 0));
end comb;

architecture beh of comb is
component and_gate
    port (a,b : in std_logic_vector(1 downto 0);
          c: out std_logic_vector(1  downto 0));
end component;

begin
    a: and_gate port map ((P or Q),Q,R);
end beh;
```

Action

Be sure to use only static expressions as actuals in the component instantiation statement as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity and_gate is
    port (a,b : in std_logic_vector(1 downto 0);
          c: out std_logic_vector(1  downto 0));
end and_gate;

architecture beh of and_gate is
begin
    c<= a and b;
end beh;

library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (P,Q : in std_logic_vector(1 downto 0);
          R: out std_logic_vector(1  downto 0));
end comb;

architecture beh of comb is
component and_gate
    port (a,b : in std_logic_vector(1 downto 0);
          c: out std_logic_vector(1  downto 0));
end component;
```

```
begin
    a: and_gate port map (P,Q,R);
end beh;
```

CD623

@W: Shared Variables in packages are not recommended.

A shared variable is declared in a package (shared variables can cause undesirable affects if the package is made visible in any other design). In the test case below, shared variable `temp` is declared inside the package `my_pack` which results in the warning.

```
package my_pack is
    shared variable temp: bit_vector( 1 downto 0 );
end my_pack;

use work.my_pack.all;

entity seq is
    port (A: in bit_vector(1 downto 0 );
          B: in bit_vector(1 downto 0 );
          C,D: buffer bit_vector(1 downto 0 ));
end seq;

architecture rtl of seq is
begin
    process (a)
    begin
        temp:= a;
        d<= not temp;
    end process;

    process (b)
    begin
        temp:=b;
        c <= temp;
    end process;
end rtl;
```

Action

Be sure to declare any shared variables within the architecture of the entity where one or more processes are accessing the variable. In the modified test case below, shared variable `temp` is declared within architecture `rtl`.

```
entity seq is
    port (A: in bit_vector(1 downto 0 );
          B: in bit_vector(1 downto 0 );
          C,D: buffer bit_vector(1 downto 0 ));
end seq;

architecture rtl of seq is
shared variable temp: bit_vector( 1 downto 0 );
begin
    process (a)
    begin
        temp:= a;
        d<= not temp;
    end process;

    process (b)
    begin
        temp:=b;
        c <= temp;
    end process;
end rtl;
```

CD624

@W: If the exponent is negative, the result of ** will be 0 - simulation mismatch possible

The exponent of the `**` operator could be a negative value. In the test case below, the exponent of the `**` operator is `a` and, because `a` is declared as an integer type, it can accept both positive and negative values. When `a` is negative, the result would be 0 which could cause a simulation mismatch.

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

```

entity comb is
    port (a_in: in integer ;
          d_out: out integer);
end comb;

architecture beh of comb is
begin
    d_out <= 8 ** a_in;
end beh;

```

Action

Make sure that the exponent of the `**` operator is a non-negative value. In the corrected test case below, the exponent of `**` operator `a` is declared as a natural type.

```

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

entity comb is
    port (a_in: in natural;
          d_out: out integer);
end comb;

architecture beh of comb is
begin
    d_out <= 8 ** a_in;
end beh;

```

CD625

@W: Bidir assigned to itself - treated as a redundant assignment

A bidirectional port is assigned to itself. This form of assignment is treated as redundant because it has no hardware inference. In the test case below, bidirectional port `a_in` is assigned to itself which results in the warning.

```

library ieee;
use ieee.std_logic_1164.all;

```

```
entity buf is
    port (a_in: inout std_logic_vector(3 downto 0));
end buf;

architecture beh of buf is
begin
    a_in <= a_in;
end beh;
```

Action

Make sure that no bidirectional port is assigned to itself. In the modified test case below, bidirectional port `a_in` is used to assign a value to the output on the positive edge of the `clk` and it is also assigned the value `a`.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (a: in std_logic_vector(3 downto 0);
          clk: in std_logic;
          a_in: inout std_logic_vector(3 downto 0);
          op: out std_logic_vector(3 downto 0) );
end seq;

architecture beh of seq is
begin
    a_in <= a;
    process (clk,a_in)
    begin
        if (clk'event and clk='1') then
            op <= a_in;
        end if;
    end process;
end beh;
```

CD627

@E: Actual associated with a signal parameter must be a signal

During a procedure call, the actual associated with a signal parameter was not a signal. In subprogram calls, the actuals must be the same class as the formals. In the test case below, actual variable temp is associated with formal signal d_out which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0));
end seq;

architecture beh of seq is
procedure my_and (a: std_logic_vector; b: std_logic_vector;
                 signal d_out: out std_logic_vector) is
begin
    d_out<= a and b;
end my_and;

begin
    process (clk)
        variable temp: std_logic_vector(3 downto 0);
    begin
        if falling_edge(clk) then
            my_and(a_in, b_in, temp);
            d_out<=temp;
        end if;
    end process;
end beh;
```

Action

Make sure that the actuals and formals are of the same class in a subprogram call and also within a subprogram. In the corrected test case below, actual temp is a signal associated with formal signal d_out.

```
library ieee;
use ieee.std_logic_1164.all;
```

```

entity seq is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0));
end seq;

architecture beh of seq is
procedure my_and (a: std_logic_vector; b: std_logic_vector;
                 signal d_out: out std_logic_vector) is
begin
    d_out<= a and b;
end my_and;

signal temp : std_logic_vector(3 downto 0);
begin
    process (clk)
    begin
        if falling_edge(clk) then
            my_and(a_in, b_in, temp);
            d_out<=temp;
        end if;
    end process;
end beh;

```

CD628

@E: Actual associated with a variable parameter must be a variable

During a procedure call, the actual associated with a variable parameter was not a variable. In subprogram calls, the actuals must be the same class as the formals. Also, if the parameter mode in a procedure is out or inout, the default class of the parameter is variable. In the test case below, actual signal temp is associated with formal variable d_out which results in the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0));
end seq;

```

```

architecture beh of seq is
procedure my_and (a: std_logic_vector; b: std_logic_vector;
                 d_out: out std_logic_vector) is
begin
    d_out:= a and b;
end my_and;

signal temp: std_logic_vector(3 downto 0);
begin
    process (clk)
    begin
        if falling_edge(clk) then
            my_and(a_in, b_in,temp);
            d_out<=temp;
        end if;
    end process;
end beh;

```

Action

Make sure that the actuals and formals are of the same class in a subprogram call and also within a subprogram. In the corrected test case below, actual temp is a variable associated with formal variable d_out.

```

library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0));
end seq;

architecture beh of seq is
procedure my_and (a: std_logic_vector; b: std_logic_vector;
                 d_out: out std_logic_vector) is
begin
    d_out:= a and b;
end my_and;

begin
    process (clk)
    variable temp: std_logic_vector(3 downto 0);
    begin
        if falling_edge(clk) then

```

```
        my_and(a_in, b_in,temp);
        d_out<=temp;
    end if;
end process;
end beh;
```

CD630

@N: Synthesizing <entity_name>.

The compiler synthesized an entity. The complete entity name includes the library name, entity name, and architecture name separated by periods. In the following example, the entity top_level is synthesized in library work with architecture structural; the corresponding note is “Synthesizing work.top_level.structural.”

```
library ieee;
use ieee.std_logic_1164.all;

entity muxhier is
port (outvec: out std_logic_vector (7 downto 0);
      a_vec, b_vec: in std_logic_vector (7 downto 0);
      sel: in std_logic);
end muxhier;

architecture mux_design of muxhier is -- mux
begin
with sel select
  outvec <= a_vec when '1',
              b_vec when '0',
              "XXXXXXXX" when others;
end mux_design;

library ieee;
use ieee.std_logic_1164.all;

entity reg8 is
port (q: buffer std_logic_vector (7 downto 0);
      data: in std_logic_vector (7 downto 0);
      clk, rst: in std_logic);
end reg8;
```

```
architecture reg8_design of reg8 is -- eight bit register
begin
process (clk, rst)
begin
if rst = '1' then
  q <= X"00";
elsif rising_edge(clk) then
  q <= data;
end if;
end process;
end reg8_design;

library ieee;
use ieee.std_logic_1164.all;

entity rotate is
port (q: buffer std_logic_vector (7 downto 0);
      data: in std_logic_vector (7 downto 0);
      clk, rst, r_l: in std_logic);
end rotate;

architecture rotate_design of rotate is
-- rotates bits or loads
-- when r_l is high, it rotates; if low, it loads data
begin
process (clk, rst)
begin
if rst = '1' then
  q <= X"00";
elsif rising_edge(clk) then
  if r_l = '1' then
    q <= q ( 6 downto 0 ) & q ( 7 ) ;
  else
    q <= data;
  end if;
end if;
end process;
end rotate_design;

-----
--      Top level
-----

library ieee;
use ieee.std_logic_1164.all;
```

```
entity top_level is
port (q: buffer std_logic_vector (7 downto 0);
      a, b: in std_logic_vector (7 downto 0);
      sel, r_l, clk, rst: in std_logic);
end top_level;

architecture structural of top_level is

component muxhier -- component declaration for mux
port (outvec: out std_logic_vector (7 downto 0);
      a_vec, b_vec: in std_logic_vector (7 downto 0);
      sel: in std_logic);
end component;

component reg8 -- component declaration for reg8
port (q: buffer std_logic_vector (7 downto 0);
      data: in std_logic_vector (7 downto 0);
      clk, rst: in std_logic);
end component;

component rotate -- component declaration for rotate
port (q: buffer std_logic_vector (7 downto 0);
      data: in std_logic_vector (7 downto 0);
      clk, rst, r_l: in std_logic);
end component;

-- declare the internal signals here
signal mux_out, reg_out: std_logic_vector (7 downto 0);

begin -- structural description begins
-- instantiate a mux, name it inst1, and wire it up
-- here we connect the mux with positional port mapping
-- (by position)
inst1: muxhier port map (mux_out, a, b, sel);

-- instantiate a rotate, name it inst2, and wire it up
inst2: rotate port map (q, reg_out, clk, rst, r_l);

-- instantiate a reg8, name it inst3, and wire it up
-- reg8 is connected with named port mapping (by name)
-- the port connections can be given in any order
-- Note that the local signal names are on the right of the
-- '>' mapping operators, and the signal names from the
-- component declaration are on the left.

inst3: reg8
port map (clk => clk, data => mux_out,
          q => reg_out, rst => rst);
end structural;
```

Action

Note that the top-level module is the last module analyzed. You can change the top-level module from the Verilog/VHDL tab in the implementation options. The corresponding Tcl command in VHDL or Verilog is:

```
set_option -top_module "top_module_name"
```

CD632

@E: duplicate entity name <comb>

The compiler encountered a duplicate entity name. In the test case below, entity name comb appears twice which causes the error.

```
entity comb is
    port (a,b: in bit;
          c: out bit);
end comb;

architecture beh of comb is
begin
    c<= a and b;
end beh;

entity comb is
    port (a,b: in bit;
          c: out bit);
end comb;

architecture str of comb is
begin
    c<= a or b;
end str;
```

Action

Make sure that every entity definition has a unique name. In the corrected test case below, the two entity definitions are assigned unique entity names.

```
entity comb_1 is
    port (a,b: in bit;
          c: out bit);
end comb_1;

architecture beh of comb_1 is
begin
    c<= a and b;
end beh;

entity comb is
    port (a,b: in bit;
          c: out bit);
end comb;

architecture str of comb is
begin
    c<= a or b;
end str;
```

CD633

@E: duplicate configuration name <adder>

The compiler encountered a duplicate configuration name. In the test case below, configuration name adder appears twice which causes the error.

```
entity fa is
    port (a,b,cin : in bit;
          sum,carry : out bit);
end fa;

architecture ex1 of fa is
begin
    sum <= a xor b xor cin;
    carry <= ((a and b) or ( a and cin) or ( b and cin));
end ex1;

architecture ex2 of fa is
begin
    sum <= a xor b xor cin;
    carry <= (not a) and ( b or cin);
end ex2;
```

```
configuration adder of fa is
    for ex1
    end for;
end adder;

configuration adder of fa is
    for ex2
    end for;
end adder;
```

Action

Make sure that every configuration has a unique name. In the corrected test case below, the two configurations are assigned unique names (adder and adder1).

```
entity fa is
    port (a,b,cin : in bit;
          sum,carry : out bit);
end fa;

architecture ex1 of fa is
begin
    sum <= a xor b xor cin;
    carry <= ((a and b) or ( a and cin) or ( b and cin));
end ex1;

architecture ex2 of fa is
begin
    sum <= a xor b xor cin;
    carry <= (not a) and ( b or cin);
end ex2;

configuration adder of fa is
    for ex1
    end for;
end adder;

configuration adder1 of fa is
    for ex2
    end for;
end adder1;
```

CD638

@W: Signal <sig> is undriven. Either assign the signal a value or remove the signal declaration.

A declared signal is not assigned a value. In the test case below, no value is assigned to signal sig which results in the warning.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (a: in std_logic;
          b: in std_logic;
          out1 : out std_logic);
end;

architecture rtl of seq is
signal sig:std_logic;
begin
    process(a)
    begin
        if rising_edge(a) then
            out1<=b;
        end if;
    end process;
end;
```

Action

Either assign the signal a value or remove the signal declaration. In the corrected test case below, a value (not a) is assigned to signal sig.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (a: in std_logic;
          b: in std_logic;
          out1 : out std_logic);
end;
```

```

architecture rtl of seq is
signal sig:std_logic;
begin
sig<= not a;
process(a)
begin
if rising_edge(a) then
out1<=b or sig;
end if;
end process;
end;
```

CD639

@W: Bit <3> of Signal <a1> is undriven

A bit of a vector signal is not assigned a value. In the test case below, the third bit of signal a1 is not assigned a value which results in the warning.

```

entity comb is
port (a: in bit_vector(2 downto 0);
      b: in bit_vector(3 downto 0);
      c: out bit_vector(3 downto 0));
end entity;

architecture str of comb is
signal a1: bit_vector(3 downto 0);
begin
a1 (2 downto 0)<= a;
c<= a1 and b;
end str;
```

Action

Make sure that all bits of a signal are assigned values. In the test case below, every bit of signal a1 is assigned a value.

```

entity comb is
port (a: in bit_vector(2 downto 0);
      b: in bit_vector(3 downto 0);
      c: out bit_vector(3 downto 0));
end entity;
```

```
architecture str of comb is
signal a1: bit_vector(3 downto 0);
begin
    a1 (3 downto 0)<= a &(not a(0));
    c<= a1 and b;
end str;
```

CD642

@W: Ignoring use clause - library <work1> not found ...

The library declared in the use clause statement could not be found in the scope. In the test case below, library work1 specified in the use clause statement use work1.test.all is not found in the current context which results in the warning.

```
entity test is
    port (A: in bit_vector(1 downto 0);
          B: in bit_vector(1 downto 0);
          C: buffer bit_vector(1 downto 0));
end test;

architecture rtl of test is
begin
    c <= a and b ;
end rtl;

use work1.test.all;

entity comb is
    port (A: in bit_vector(1 downto 0);
          B: in bit_vector(1 downto 0);
          C: out bit_vector(1 downto 0));
end comb;

architecture rtl of comb is
component test is
    port (A: in bit_vector(1 downto 0);
          B: in bit_vector(1 downto 0);
          C: buffer bit_vector(1 downto 0));
end component;

signal c1 : bit_vector(1 downto 0);
```

```
begin
    ul: test port map (a,b,c1);
    c<=c1;
end rtl;
```

Action

Make sure that the library specified in the use clause statement is already declared. In the modified test case below, the default working library work is specified as the library in the use clause statement.

```
entity test is
    port (A: in bit_vector(1 downto 0);
          B: in bit_vector(1 downto 0);
          C: buffer bit_vector(1 downto 0));
end test;

architecture rtl of test is
begin
    c <= a and b ;
end rtl;

use work.test.all;

entity comb is
    port (A: in bit_vector(1 downto 0);
          B: in bit_vector(1 downto 0);
          C: out bit_vector(1 downto 0));
end comb;

architecture rtl of comb is
component test is
    port (A: in bit_vector(1 downto 0);
          B: in bit_vector(1 downto 0);
          C: buffer bit_vector(1 downto 0));
end component;

signal c1 : bit_vector(1 downto 0);

begin
    ul: test port map (a,b,c1);
    c<=c1;
end rtl;
```

CD643

@W: Ignoring use clause – <std_logic_11641> not found ...

Either the library is not already defined or the primary unit specified in the use clause is not already declared in the given library. The primary unit specified in a use clause statement must be declared as either an entity or a package or as a configuration in a given library. In the test case below, identifier std_logic_11641, which is specified as the primary unit in the use clause statement, is neither declared as a package nor as an entity in the ieee library which results in the warning.

```
library ieee;
use ieee.std_logic_11641.all;

entity comb is
    port (a,b: in std_logic;
          c: out std_logic);
end entity;

architecture str of comb is
begin
    c<= a and b;
end str;
```

Action

Make sure that the primary unit specified in the use clause is already declared in the given library. In the modified test case below, the primary unit in the use clause is specified as std_logic_1164 which is already declared as a package within the ieee library.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (a,b: in std_logic;
          c: out std_logic);
end entity;

architecture str of comb is
begin
    c<= a and b;
end str;
```

CD645

@W: Ignoring undefined library <mypack>

A library is referenced in the VHDL code that is not included in the working project, that is not already defined in a standard library such as std or ieee, or that is not in the default library work. In the test case below, the my_pack library referenced in the library statement does not exist which results in the warning.

```
library ieee;
library mypack;
use ieee.std_logic_1164.all;

entity comb is
    port (a,b: in std_logic;
          c: out std_logic);
end;

architecture rtl of comb is
begin
    c<= a and b;
end;
```

Action

Be sure to reference only defined libraries when using a library statement or by including the library file in the working project. In the corrected test case below, there are no references to undefined libraries.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (a,b: in std_logic;
          c: out std_logic);
end;

architecture rtl of comb is
begin
    c<= a and b;
end;
```

CD648

@E: Expression does not match type <std_logic_vector>

The type of the target assignment does not match the target type. In the test case below, b is assigned to out1. However, because b is of type bit_vector and out1 is declared as type std_logic_vector, the mismatch causes error.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (a: in std_logic;
          b: in bit_vector(3 downto 0);
          out1: out std_logic_vector(3 downto 0));
end;

architecture rtl of seq is
begin
    process(a,b)
    begin
        if rising_edge(a) then
            out1<= (b);
        end if;
    end process;
end;
```

Action

Make sure that the type of the target expression is the same as the target type. If the two types are related, use type casting or conversion functions to convert the type of the target expression as shown in the corrected test case below where bit_vector is converted to std_logic_vector and then assigned to out1.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (a: in std_logic;
          b: in bit_vector(3 downto 0);
          out1: out std_logic_vector(3 downto 0));
end;
```

```
architecture rtl of seq is
begin
  process(a,b)
  begin
    if rising_edge(a) then
      out1<=TO_STDLOGICVECTOR(b);
    end if;
  end process;
end;
```

In a process statement, either a sensitivity list or a wait statement must be used to trigger the process. Wait for statements (e.g., wait for 10ns;) and wait statements (e.g., wait;) are ignored by the compiler and can cause RTL/post synthesis simulation mismatches. The wait statements that are honored by the compiler are wait on (e.g., wait on a,b,c;) and wait until used with a clock edge (e.g., wait until rising_edge(clk);). Also make sure that all wait statements in a process are triggered on the identical condition.

CD650

@E: Expecting formal parameter type. Invalid type?

An invalid type is used as the formal parameter type in an alias declaration for a function. In the test case below, mod_myand is declared as an alias for function my_and. However, because formal parameter b (std_logic_vector) is declared as an invalid type (std_logic_vector1), the compiler errors out.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
  port (clk: in std_logic;
        a_in, b_in: in std_logic_vector(3 downto 0);
        d_out: out std_logic_vector(3 downto 0));
end seq;
```

```
architecture beh of seq is
function my_and (a: std_logic_vector; b: std_logic_vector)
    return std_logic_vector
is variable temp: std_logic_vector(a'range);
begin
    temp:= a and b;
    return temp;
end;

alias mod_myand is my_and [std_logic_vector, std_logic_vector];
    return std_logic_vector ];
begin
    process (clk)
begin
    if rising_edge(clk) then
        d_out <= mod_myand(a_in, b_in);
    end if;
    end process;
end beh;
```

Action

Be sure to specify a valid type for the formal parameter in the alias declaration as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0));
end seq;

architecture beh of seq is
function my_and (a: std_logic_vector; b: std_logic_vector)
    return std_logic_vector
is variable temp: std_logic_vector(a'range);
begin
    temp:= a and b;
    return temp;
end;
```

```

alias mod_myand is my_and [std_logic_vector, std_logic_vector
    return std_logic_vector];
begin
    process (clk)
    begin
        if rising_edge(clk) then
            d_out <= mod_myand(a_in, b_in);
        end if;
    end process;
end beh;
```

CD652

@E: Expecting function return type

An alias for a function and the return type of the function are not declared in the signature part. In the test case below, when declaring alias mod_myand for the function my_and, the signature part of the alias declaration does not have a return type for the function which causes the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0));
end seq;

architecture beh of seq is
function my_and (a, b: std_logic_vector) return std_logic_vector
is variable temp: std_logic_vector(a'range);
begin
    temp:= a and b;
    return temp;
end;

alias mod_myand is my_and [std_logic_vector,
    std_logic_vector return];
begin
    process (clk)
    begin
```

```
        if rising_edge(clk) then
            d_out <= mod_myand(a_in, b_in);
        end if;
    end process;
end beh;
```

Action

Be sure to specify the return type of the function in the signature part of the alias declaration as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0));
end seq;

architecture beh of seq is
function my_and (a, b: std_logic_vector) return std_logic_vector
is variable temp: std_logic_vector(a'range);
begin
    temp:= a and b;
    return temp;
end;

alias mod_myand is my_and [std_logic_vector, std_logic_vector
                           return std_logic_vector];
begin
    process (clk)
begin
    if rising_edge(clk) then
        d_out <= mod_myand(a_in, b_in);
    end if;
end process;
end beh;
```

CD653

@E: Expecting return type to match that in original signature: <std_logic_vector>

An alias was declared for a function and the return type in the signature part and in the function declaration were different. In the test case below, when declaring alias mod_myand for the function my_and, the signature part of the alias declaration does not have the same return type as the function my_and which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0));
end seq;

architecture beh of seq is
function my_and (a, b: std_logic_vector) return std_logic_vector
    is variable temp: std_logic_vector(a'range);
begin
    temp:= a and b;
    return temp;
end;

alias mod_myand is my_and [std_logic_vector,std_logic_vector
                           return std_logic];
begin
    process (clk)
    begin
        if rising_edge(clk) then
            d_out <= mod_myand(a_in, b_in);
        end if;
    end process;
end beh;
```

Action

Be sure to specify the same return type as that of the function declaration in the signature part of the alias. In the corrected test case below, the return type of alias declaration mod_myand and the return type of function my_and are the same.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0));
end seq;

architecture beh of seq is
function my_and (a, b: std_logic_vector) return std_logic_vector
    is variable temp: std_logic_vector(a'range);
begin
    temp:= a and b;
    return temp;
end;

alias mod_myand is my_and [std_logic_vector, std_logic_vector
                           return std_logic_vector];
begin
    process (clk)
    begin
        if rising_edge(clk) then
            d_out <= mod_myand(a_in, b_in);
        end if;
    end process;
end beh;
```

CD654

@E: Statement with label "<u3>" not found!

A label in a configuration declaration did not have a corresponding statement with that label. In the test case below, instantiation label u3 is encountered in the configuration declaration, but because no statement with label u3 is found, the compiler errors out.

```
entity HA is
    port (U,V: in bit;
          X,Y: out bit);
end HA;

architecture str of Ha is
begin
    x<= u xor v;
    y<= u and v;
end str;

entity comb is
    port (A, B: in bit;
          SUM,SUM_1,CARRY, CARRY_1:out bit);
end comb;

architecture beh of comb is
component dummy
    port (A, B:in bit;
          SUM, CARRY: out bit);
end component;

begin
    u1: dummy port map (A, B, SUM,CARRY);
    u2: dummy port map ((not A), B,SUM_1,CARRY_1);
end beh;

configuration my_config of comb is
    for beh
        for u1: dummy use entity HA port map (U => A, V => B,
                                                X => SUM,Y=>CARRY);
        end for;
        for u3: dummy use entity HA port map (U => A, V => B,
          X => SUM,Y=>CARRY);
        end for;
    end for;
end my_config;
```

Action

Be sure to use only those labels in a configuration declaration that have an associated block statement as shown in the corrected test case below.

```
entity HA is
    port (U,V: in bit;
          X,Y: out bit);
end HA;

architecture str of Ha is
begin
    x<= u xor v;
    y<= u and v;
end str;

entity comb is
    port (A, B: in bit;
          SUM,SUM_1,CARRY, CARRY_1:out bit);
end comb;

architecture beh of comb is
component dummy
    port (A, B:in bit;
          SUM, CARRY: out bit);
end component;

begin
    u1: dummy port map (A, B, SUM,CARRY);
    u2: dummy port map ((not A), B,SUM_1,CARRY_1);
end beh;

configuration my_config of comb is
    for beh
        for u1: dummy use entity HA port map (U => A, V => B,
                                               X => SUM,Y=>CARRY);
        end for;
        for u2: dummy use entity HA port map (U => A, V => B,
                                               X => SUM,Y=>CARRY);
        end for;
    end for;
end my_config;
```

CD656

@E: Unconstrained output port <sample_out> in instantiation

An unconstrained port is used in an instantiation (ports in an instantiation require an explicit size). In the test case below, port sample_out of instantiated entity add_unconstrained is declared as an unconstrained port which, when used in an instantiation, causes the error.

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

entity add_unconstrained is
    port (sample_in1, sample_in2 : in unsigned;
          sample_out : out unsigned);
end add_unconstrained;

architecture rtl of add_unconstrained is
begin
    sample_out <= sample_in1 + sample_in2;
end rtl;

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

entity type_conversion_of_unconstr_output is
    port (asample_in1, asample_in2 :
          in std_logic_vector(7 downto 0);
          asample_out : out std_logic_vector(7 downto 0));
end type_conversion_of_unconstr_output;

architecture rtl of type_conversion_of_unconstr_output is
component add_unconstrained
    port (sample_in1, sample_in2 : in unsigned;
          sample_out : out unsigned);
end component;

signal asample_in1_tmp : unsigned(7 downto 0);
signal asample_in2_tmp : unsigned(7 downto 0);
signal asample_out_tmp: unsigned(7 downto 0);
begin
    asample_in1_tmp <= unsigned(asample_in1);
    asample_in2_tmp <= unsigned(asample_in2);
```

```
u_add : add_unconstrained
port map (sample_in1 => asample_in1_tmp,
           sample_in2 => asample_in2_tmp,
           std_logic_vector(sample_out) => asample_out);
end rtl;
```

Action

The above error can be eliminated by specifying the size of the instantiated port as shown in the corrected test case below.

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

entity add_unconstrained is
    port (sample_in1 : in unsigned;
          sample_in2 : in unsigned;
          sample_out : out unsigned);
end add_unconstrained;

architecture rtl of add_unconstrained is
begin
    sample_out <= sample_in1 + sample_in2;
end rtl;

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

entity type_conversion_of_unconstr_output is
    port (asample_in1 : in std_logic_vector(7 downto 0);
          asample_in2 : in std_logic_vector(7 downto 0);
          asample_out : out std_logic_vector(7 downto 0));
end type_conversion_of_unconstr_output;

architecture rtl of type_conversion_of_unconstr_output is
component add_unconstrained
    port (sample_in1 : in unsigned;
          sample_in2 : in unsigned;
          sample_out : out unsigned);
end component;
```

```

signal asample_in1_tmp : unsigned(7 downto 0);
signal asample_in2_tmp : unsigned(7 downto 0);
signal asample_out_tmp: unsigned(7 downto 0);
begin
  asample_in1_tmp <= unsigned(asample_in1);
  asample_in2_tmp <= unsigned(asample_in2);
  asample_out <= unsigned(asample_out_tmp);
  u_add : add unconstrained
  port map (sample_in1 => asample_in1_tmp,
            sample_in2 => asample_in2_tmp,
            sample_out => asample_out_tmp);
end rtl;

```

CD658

@E: Found unconstrained array type - expecting array type to be constrained

The base type of an array was specified as an unconstrained array. VHDL does not allow a constrained or unconstrained array type to be declared as an array of an unconstrained array. In the test case below, constrained array my_array is declared as an array of unconstrained array std_logic_vector which causes the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity seq is
  port (A: in std_logic;
        sel: in std_logic;
        B: in std_logic;
        C: out std_logic_vector(1 downto 0));
end seq;

architecture rtl of seq is
type my_array is array(1 downto 0) of std_logic_vector;
  signal temp:my_array;
begin
  temp(0) <= (1=>a ,0=> b);
  temp(1)<= (1=>not a,0=>not b);
  process(sel)
  begin

```

```
if(sel='1') then
    c<=temp(0);
else
    c<=temp(1);
end if;
end process;
end rtl;
```

Action

Make sure that the type of the base array is constrained when declaring an array. In the corrected test case below, the base type of array my_array is std_logic_vector(1 downto 0) which is a constrained array.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (A: in std_logic;
          sel: in std_logic;
          B: in std_logic;
          C: out std_logic_vector(1 downto 0));
end seq;

architecture rtl of seq is
type my_array is array(1 downto 0) of std_logic_vector(1 downto 0);
signal temp:my_array;
begin
temp(0) <= (1=>a ,0=> b);
temp(1)<= (1=>not a,0=>not b);
process(sel)
begin
    if(sel='1') then
        c<=temp(0);
    else
        c<=temp(1);
    end if;
end process;
end rtl;
```

CD663

@E: No feasible entries for subprogram: <*my_and*>

When declaring an alias for a function, either the number of parameters or the parameter type in the signature part did not match the original functional declaration. In the test case below, the number of parameters in the original function *my_and* and in the signature part of the alias *mod_myand* are not same which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0));
end seq;

architecture beh of seq is
function my_and (a, b: std_logic_vector) return std_logic_vector
    is variable temp: std_logic_vector(a'range);
begin
    temp:= a and b;
    return temp;
end;

alias mod_myand is my_and [std_logic_vector
                           return std_logic_vector];
begin
    process (clk)
    begin
        if rising_edge(clk) then
            d_out <= mod_myand(a_in, b_in);
        end if;
    end process;
end beh;
```

Action

Make sure that the number of parameters and the parameter type in the signature part of the alias declaration match the original functional declaration. In the test case below, adding one more parameter (with the same type as in the function declaration) to the signature part of the alias declaration corrects the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (clk: in std_logic;
          a_in, b_in: in std_logic_vector(3 downto 0);
          d_out: out std_logic_vector(3 downto 0));
end seq;

architecture beh of seq is
function my_and (a, b: std_logic_vector) return std_logic_vector
    is variable temp: std_logic_vector(a'range);
begin
    temp:= a and b;
    return temp;
end;

alias mod_myand is my_and [std_logic_vector, std_logic_vector
                           return std_logic_vector];
begin
    process (clk)
    begin
        if rising_edge(clk) then
            d_out <= mod_myand(a_in, b_in);
        end if;
    end process;
end beh;
```

CD666

@W: End of file reached without matching pragma translate_on

The compiler encountered a translate_off directive and then reached the end of the file without finding a corresponding translate_on directive. The synthesis translate_off and translate_on directives isolate non-synthesizable code from synthesizable code. The synthesis translate_off and translate_on directives must be used in pairs with a translate_on directive always following a translate_off directive. In the test case below, the translate_off directive has no matching translate_on directive in the file which results in the above warning.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (a,b: in std_logic;
          c: out std_logic);
end entity;

architecture str of comb is
begin
    c<= a and b;
-- synthesis translate_off
    process(a,b)
    begin
        c<= a and b;
    end str;
```

Action

Be sure to include a translate_on directive following the translate_off directive as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (a,b: in std_logic;
          c: out std_logic);
end entity;
```

```
architecture str of comb is
begin
    c<= a and b;
-- synthesis translate_off
    process(a,b)
    begin
        c<= a and b;
-- synthesis translate_on
end str;
```

CD669

@E: Value <20> exceeded the specified integer range <<1> , <10>>

A signal declared as an integer with a specific range was assigned a value that exceeded the declared range limit. In the test case below, signal a is defined as an integer with a range of 1 to 10. This signal is subsequently assigned a value of 20 which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity add is
    port (din : in integer range 1 to 10;
          q: out integer range 1 to 30);
end add;

architecture beh of add is
signal a: integer range 1 to 10;
begin
    a<=20;
    q<=a + din;
end beh;
```

Action

Make sure that the value assigned to the constrained integer is within the range of the constraint as shown in the corrected example below.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity add is
    port (din : in integer range 1 to 10;
          q: out integer range 1 to 30);
end add;

architecture beh of add is
signal a: integer range 1 to 10;
begin
    a<=5;
    q<=a + din;
end beh;
```

CD678

@E: Corrupt file pointer

This error occurs when the compiler is unable to open the referenced file and indicates that the file is either not present or not readable.

As a typical example, this error occurs when a referenced init file cannot be found at the indicated location. When referencing an init file in the source file, the absolute or relative path of the file must be specified correctly.

Action

See the line in the HDL source (referenced in the message) which includes a read from the file that cannot be accessed. For example:

```
DPROM_FILE_NAME : string := "deprom_56.txt"
```

Make sure that the path and read permissions to this file are correct. Note that if the init file is not present in the src directory, the implementation directory is then searched for the file.

CD708

@E: Not a Concurrent Statement

The compiler encountered a sequential statement in the concurrent block of the architecture outside of the process. In the test case below, a sequential case statement is used in the concurrent part of the architectural block which results in the error.

```
entity comb is
    port (a,b,cin : in bit;
          sum,cout : out bit);
end comb;

architecture df of comb is
signal temp:bit;
begin
    temp<= a xor b;
    sum<=temp xor cin;
    case (temp) is
        when '1'=> cout<=cin;
        when '0'=> cout<=a;
    end df;
```

Action

Make sure that any sequential statements are used within the process. In the test case below, using the case statement within the process corrects the error.

```
entity comb is
    port (a,b,cin : in bit;
          sum,cout : out bit);
end comb;

architecture df of comb is
signal temp:bit;
begin
    temp<= a xor b;
    sum<=temp xor cin;
    process (temp,a,cin)
    begin
        case(temp) is
```

```

        when '1'=> cout<=cin;
        when '0'=> cout<=a;
    end case;
end process;
end df;

```

CD710

@E: prefix of qualified expression must be a type mark

The prefix of a qualified expression was not a type mark. In the following test case, the qualified expression is written incorrectly which results in the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (clk,rst : in std_logic;
          data : in bit_vector(1 downto 0);
          dout : out bit_vector(1 downto 0));
end test;

architecture test_arch of test is
begin
    process(clk)
    begin
        if(clk'event and clk = '1') then
            if(rst = '1') then
                dout <= "00";
            else
                dout <= bit'vector'(data);
            end if;
        end if;
    end process;
end test_arch;

```

Action

Be sure to use a type mark as a prefix to a qualified expression and also to write the qualified expression in a proper format as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (clk,rst : in std_logic;
          data : in bit_vector(1 downto 0);
          dout : out bit_vector(1 downto 0));
end test;

architecture test_arch of test is
begin
    process(clk)
    begin
        if(clk'event and clk = '1') then
            if(rst = '1') then
                dout <= "00";
            else
                dout <= bit_vector'(data);
            end if;
        end if;
    end process;
end test_arch;
```

CD711

@N: Allowing multiple process write for multi-port RAM <mem>

The compiler found code for a multi-port RAM. Generally, the compiler does not allow a write to the same variable in a multiple process. However, because multi-port RAM requires this type of code, the write operation is permitted. In the following test case, the compiler allows the assignment of variable mem to two different procedural blocks as indicated by the above message.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
```

```
entity test is
    port (data0,data1 : in std_logic_vector(7 downto 0);
          waddr0,waddr1 : in std_logic_vector(7 downto 0);
          we0,we1,clk0,clk1: in std_logic;
          q0,q1: out std_logic_vector(7 downto 0));
end test;

architecture ram of test is
type mem_type is array (255 downto 0) of
    std_logic_vector (7 downto 0);
signal mem : mem_type;
signal reg_addr0,reg_addr1: std_logic_vector(7 downto 0);
begin
q0 <= mem(conv_integer (reg_addr0));
q1 <= mem(conv_integer (reg_addr1));
process (clk0)
begin
    if rising_edge(clk0) then
        reg_addr0 <= waddr0;
        if (we0 = '1') then
            mem(conv_integer (waddr0)) <= data0;
        end if;
    end if;
end process;

process (clk1)
begin
    if rising_edge(clk1) then
        reg_addr1 <= waddr1;
        if (we1 = '1') then
            mem(conv_integer (waddr1)) <= data1;
        end if;
    end if;
end process;
end ram;
```

Action

Informative message; no user action required.

CD713

@E: component: end identifier <fa2> does not match

The name in an end statement did not match the component name in the first line of the component declaration. In the test case below, identifier fa2 in the end statement does not match the component name (fa) in the first line of the component declaration which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity fa is
    port (a,b,cin : in std_logic;
          sum, carry : out std_logic);
end fa;

architecture df of fa is
begin
    sum <= a xor b xor cin;
    carry <= ((a and b) or (a and cin) or (b and cin));
end;

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (a,b,cin : in std_logic;
          sum, carry : out std_logic);
end test;

architecture structural of test is
component fa
    port (a,b,cin : in std_logic;
          sum, carry : out std_logic);
end component fa2;

begin
    u1 : fa port map(a=>a, b=>b, cin=>cin, sum=>sum, carry=>carry);
end structural;
```

Action

Make sure that the same component-name identifier is used in both the first and last lines of the component declaration. In the corrected test case below, end identifier fa now matches the component-name identifier in the first line of the component declaration.

```
library ieee;
use ieee.std_logic_1164.all;

entity fa is
    port (a,b,cin : in std_logic;
          sum, carry : out std_logic);
end fa;

architecture df of fa is
begin
    sum <= a xor b xor cin;
    carry <= ((a and b) or (a and cin) or (b and cin));
end;

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (a,b,cin : in std_logic;
          sum, carry : out std_logic);
end test;

architecture structural of test is
component fa
    port (a,b,cin : in std_logic;
          sum, carry : out std_logic);
end component fa;

begin
    u1 : fa port map(a=>a, b=>b,cin=>cin, sum=>sum,carry=>carry);
end structural;
```

CD715

@E: Cast of incompatible types

A type cast of incompatible types was attempted. The VHDL language allows restricted type casting (i.e., explicitly converting values between closely related types). In the test case below, converting temp1 of type bit_vector to SIGNED is illegal as bit_vector and signed are not directly related.

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

entity comb is
    port (A: in bit_vector(1 downto 0);
          B: in bit_vector( 1 downto 0);
          C: out signed(1 downto 0));
end comb;

architecture rtl of comb is
signal temp1: bit_vector(1 downto 0);
begin
    temp1<= a and b;
    c<= SIGNED((temp1));
end rtl;
```

Action

Be sure to use compatible types when using type casting. In the following test case, converting bit_vector to std_logic_vector using a user-defined function and then converting to signed using type casting corrects the error.

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

entity comb is
    port (A: in bit_vector(1 downto 0);
          B: in bit_vector(1 downto 0);
          C: out signed(1 downto 0));
end comb;
```

```
architecture rtl of comb is
signal temp1: bit_vector(1 downto 0);
begin
    temp1<= a and b;
    c<= SIGNED (TO_STDLOGICVECTOR(temp1));
end rtl;
```

CD716

@E: Expression has ambiguous type

An identifier was assigned a value that did not match the type of identifier. In the test case below, integer variable count is assigned bit value 0 which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (q: out std_logic_vector(7 downto 0);
          d : in std_logic_vector(7 downto 0);
          clk : in std_logic);
end seq;

architecture test of seq is
begin
    process(clk)
        variable count: integer;
    begin
        if(clk'event and clk='1') then
            if count<=10 then
                count:= count+1;
            else
                count:='0';
                q<=d;
            end if;
        end if;
    end process;
end test;
```

Action

Make sure that the target and the assignment expression are of the same type. To eliminate the error, assign variable count an integer value of 0 as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
    port (q: out std_logic_vector(7 downto 0);
          d : in std_logic_vector(7 downto 0);
          clk : in std_logic);
end seq;

architecture test of seq is
begin
    process(clk)
        variable count: integer;
    begin
        if(clk'event and clk='1') then
            if count<=10 then
                count:= count+1;
            else
                count:=0;
                q<=d;
            end if;
        end if;
    end process;
end test;
```

CD717

@E: entity: end identifier <temp> does not match

The name in an entity end statement did not match the name at the beginning of the entity declaration. In the following test case, the entity declaration name is adder, but the end statement name is temp which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;
```

```

entity adder is
    port (a, b, cin:std_logic;
          sum, cout:out std_logic);
end temp;

architecture behave of adder is
begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;

```

Action

Make sure that the entity declaration name and the end declaration name are the same. To eliminate the error, replace `temp` with `adder` in the `end` statement as shown in the corrected test case below.

```

library ieee;
use ieee.std_logic_1164.all;

entity adder is
    port (a, b, cin:std_logic;
          sum, cout:out std_logic);
end adder;

architecture behave of adder is
begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;

```

The entity name specified in the `end` statement is optional.

CD718

@E: doubled '_' is illegal in identifier

Two consecutive underscore characters were encountered in an identifier. A basic identifier in VHDL is composed of a sequence of one or more characters. A legal character is an upper-case letter (A...Z), a lower case letter (a...z), a digit (0...9), or the underscore (_) character. The first character in a basic identifier must be a letter and the last character cannot be an underscore.

Also, two underscores cannot appear consecutively (illegal construct in VHDL). In the following test case, the input port `addr_1` includes two consecutive underscore characters which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (addr_1 : in std_logic_vector(2 downto 0);
          output : out std_logic_vector(2 downto 0));
end comb;

architecture beh of comb is
begin
    process(addr_1)
    begin
        case addr_1 is
            when "000" => output <= "101";
            when "001" => output <= "100";
            when "010" => output <= "000";
            when "011" => output <= "110";
            when "100" => output <= "001";
            when "101" => output <= "010";
            when "110" => output <= "011";
            when "111" => output <= "111";
        end case;
    end process;
end beh;
```

Action

Make sure to use only a single underscore character in an identifier. To eliminate the error in the above test case, delete the extra underscore in each occurrence of `addr_1`.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
    port (addr_1 : in std_logic_vector(2 downto 0);
          output : out std_logic_vector(2 downto 0));
end comb;
```

```

architecture beh of comb is
begin
    process(addr_1)
    begin
        case addr_1 is
            when "000" => output <= "101";
            when "001" => output <= "100";
            when "010" => output <= "000";
            when "011" => output <= "110";
            when "100" => output <= "001";
            when "101" => output <= "010";
            when "110" => output <= "011";
            when "111" => output <= "111";
        end case;
    end process;
end beh;
```

CD719

@E: character '_' may not end an identifier

The last character of an identifier was an underscore. A basic identifier in VHDL is composed of a sequence of one or more characters. A legal character is an upper-case letter (A...Z), a lower case letter (a...z), a digit (0...9), or the underscore (_) character. The first character in a basic identifier must be a letter and the last character cannot be an underscore. Also, two underscores cannot appear consecutively. In the following test case, ‘_’ is the last character of one of the input names which results in the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity adder is
    port (a, b_, cin:std_logic;
          sum, cout:out std_logic );
end adder;

architecture behave of adder is
begin
    sum <= (a xor b_) xor cin;
    cout <= (a and b_) or (a and cin) or (b_ and cin);
end behave;
```

Action

Use only alpha-numerics as the last character when specifying basic identifiers. To eliminate the error, use a legal name for the identifier as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity adder is
    port (a, b, cin:std_logic;
          sum, cout:out std_logic);
end adder;

architecture behave of adder is
begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;
```

CD721

@W: syn_enum_encoding should be used on types. Please change syn_encoding to syn_enum_encoding

A `syn_encoding` attribute is being used to define the encoding style of user-defined types (the `syn_encoding` attribute is obsolete and has been replaced by the `syn_enum_encoding` attribute). In the following test case, the `syn_encoding` attribute in the architecture block results in the warning.

```
library ieee;
use ieee.std_logic_1164.all;

entity shift_enum is
    port (clk, rst: bit;
          O: out std_logic_vector(2 downto 0));
end shift_enum;

architecture behave of shift_enum is
type state_type is (S0, S1, S2);
attribute syn_encoding: string;
attribute syn_encoding of state_type : type is "001 010 101";
signal machine : state_type;
```

```

begin
  process (clk, rst)
  begin
    if rst = '1' then
      machine <= S0;
    elsif clk = '1' and clk'event then
      case machine is
        when S0 => machine <= S1;
        when S1 => machine <= S2;
        when S2 => machine <= S0;
        when others => null;
      end case;
    end if;
  end process;
  with machine select
    O <= "001" when S0,
    "010" when S1,
    "101" when S2;
  end behave;

```

Action

Use attribute `syn_enum_encoding` to define the encoding style of any user-defined types as shown in the corrected test case below.

```

library ieee;
use ieee.std_logic_1164.all;

entity shift_enum is
  port (clk, rst: bit;
        O: out std_logic_vector(2 downto 0));
end shift_enum;

architecture behave of shift_enum is
type state_type is (S0, S1, S2);
attribute syn_enum_encoding: string;
attribute syn_enum_encoding of state_type : type is "001 010 101";
signal machine : state_type;
begin
  process (clk, rst)
  begin
    if rst = '1' then
      machine <= S0;
    elsif clk = '1' and clk'event then
      case machine is
        when S0 => machine <= S1;

```

```
        when S1 => machine <= S2;
        when S2 => machine <= S0;
        when others => null;
    end case;
end if;
end process;
with machine select
  O <= "001" when S0,
  "010" when S1,
  "101" when S2;
end behave;
```

CD722

@E: Generic <bus_size> has not been given a value

A generic was defined in an entity without an assigned value. In VHDL, it is illegal to have an undefined generic in an entity; a value must be given either in the component declaration or in the instantiation using a generic map. For top-level generics, an implementation options setting in the synthesis tool can be used. In the following test case, the generic bus_size is not given a value which causes the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity parity is
  generic (bus_size : integer );
  port (input_bus : in std_logic_vector (bus_size-1 downto 0);
        even_numbits, odd_numbits : out std_logic);
end parity;

architecture behave of parity is
begin
  process (input_bus)
  variable temp: std_logic;
  begin
    temp := '0';
    for i in input_bus'low to input_bus'high loop
      temp := temp xor input_bus(i);
```

```

        end loop;
        odd_numbits <= temp;
        even_numbits <= not temp;
    end process;
end behave;

```

Action

Make sure that all generics are declared with constant values. Assign default values to declarations and change the values accordingly through port map statements. To correct the error in the above test case, replace the line as indicated in any one of the following solutions to make sure that the generic has a constant value.

Assign a Default Value to the Generic in Entity Declaration

```

library ieee;
use ieee.std_logic_1164.all;

entity parity is
    generic (bus_size : integer := 8);
    port (input_bus : in std_logic_vector (bus_size-1 downto 0);
          even_numbits, odd_numbits : out std_logic );
end parity;

architecture behave of parity is
begin
    process (input_bus)
    variable temp: std_logic;
    begin
        temp := '0';
        for i in input_bus'low to input_bus'high loop
            temp := temp xor input_bus(i);
        end loop;
        odd_numbits <= temp;
        even_numbits <= not temp;
    end process;
end behave;

```

Assign a Value in the Component Declaration

```

library ieee;
use ieee.std_logic_1164.all;

```

```

entity parity is
    generic (bus_size : integer);
    port (input_bus : in std_logic_vector (bus_size-1 downto 0);
          even_numbits, odd_numbits : out std_logic);
end parity;

architecture behave of parity is
begin
    process (input_bus)
    variable temp: std_logic;
    begin
        temp := '0';
        for i in input_bus'low to input_bus'high loop
            temp := temp xor input_bus(i);
        end loop;
        odd_numbits <= temp;
        even_numbits <= not temp;
    end process;
end behave;

-----
library ieee;
use ieee.std_logic_1164.all;

entity test is
generic (top_size : integer:=8 );
    port (input_bus: in std_logic_vector (top_size-1 downto 0);
          even, odd: out std_logic);
end;

architecture top of test is
component parity is
    generic (bus_size : integer := 8);
    port (input_bus: in std_logic_vector (bus_size-1 downto 0);
          even_numbits, odd_numbits: out std_logic);
end component;

begin
    test : parity port map (input_bus => input_bus,
                           even_numbits => even, odd_numbits => odd);
end;

```

Assign a Value in the Generic Map

```

library ieee;
use ieee.std_logic_1164.all;

```

```
entity parity is
    generic (bus_size : integer);
    port (input_bus : in std_logic_vector (bus_size-1 downto 0);
          even_numbits, odd_numbits : out std_logic);
end parity;

architecture behave of parity is
begin
    process (input_bus)
        variable temp: std_logic;
    begin
        temp := '0';
        for i in input_bus'low to input_bus'high loop
            temp := temp xor input_bus(i);
        end loop;
        odd_numbits <= temp;
        even_numbits <= not temp;
    end process;
end behave;

-----
library ieee;
use ieee.std_logic_1164.all;

entity test is
    generic (top_size : integer:=8);
    port (input_bus: in std_logic_vector (top_size-1 downto 0);
          even, odd: out std_logic);
end;

architecture top of test is
component parity is
    generic (bus_size: integer);
    port (input_bus: in std_logic_vector (bus_size-1 downto 0);
          even_numbits, odd_numbits: out std_logic);
end component;

begin
    test: parity generic map (bus_size => 8) port map
        (input_bus =>input_bus, even_numbits => even,
         odd_numbits => odd);
end;
```

CD727

@E: Configuration specification FOR ALL: <fa> .. is already defined

A for ALL configuration specification was followed by one or more separate configuration specification statements (the for ALL configuration specification binds all the component instantiations which renders any subsequent configuration specification invalid). In the test case below, the for ALL configuration specification is followed by a for u1 configuration specification which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity fa is
    port (a,b,cin : in std_logic;
          sum,carry : out std_logic);
end fa;

architecture adder of fa is
begin
    sum <= a xor b xor cin;
    carry <= ((a and b) or (a and cin) or (b and cin));
end adder;

architecture sub of fa is
begin
    sum <= a xor b xor cin;
    carry <= (not a) and (b or cin);
end sub;

-----
library ieee;
use ieee.std_logic_1164.all;

entity fulladder is
    port (a,b,cin : in std_logic_vector(1 downto 0);
          sum,carry : out std_logic_vector(1 downto 0));
end fulladder;

architecture test of fulladder is
component fa is
    port (a,b,cin : in std_logic;
          sum,carry : out std_logic);
end component;
```

```

for ALL: fa use entity work.fa(sub);
for u1: fa use entity work.fa(add);
begin
  u1: fa port map (a(0),b(0),cin(0),sum(0),carry(0));
  u2: fa port map (a(1),b(1),cin(1),sum(1),carry(1));
end test;

```

Action

The above error can be eliminated using any of the following three methods.

1. Use only a for ALL configuration specification:

```

library ieee;
use ieee.std_logic_1164.all;

entity fa is
  port (a,b,cin : in std_logic;
        sum,carry : out std_logic);
end fa;

architecture adder of fa is
begin
  sum <= a xor b xor cin;
  carry <= ((a and b) or (a and cin) or (b and cin));
end adder;

architecture sub of fa is
begin
  sum <= a xor b xor cin;
  carry <= (not a) and (b or cin);
end sub;

-----
library ieee;
use ieee.std_logic_1164.all;

entity fulladder is
  port (a,b,cin : in std_logic_vector(1 downto 0);
        sum,carry : out std_logic_vector(1 downto 0));
end fulladder;

architecture test of fulladder is
component fa is
  port (a,b,cin : in std_logic;
        sum,carry : out std_logic);
end component;

```

```
end component;

for ALL: fa use entity work.fa(sub);
begin
    u1: fa port map (a(0),b(0),cin(0),sum(0),carry(0));
    u2: fa port map (a(1),b(1),cin(1),sum(1),carry(1));
end test;
```

2. Use separate configuration specifications statements:

```
library ieee;
use ieee.std_logic_1164.all;

entity fa is
    port (a,b,cin : in std_logic;
          sum,carry : out std_logic);
end fa;

architecture adder of fa is
begin
    sum <= a xor b xor cin;
    carry <= ((a and b) or (a and cin) or (b and cin));
end adder;

architecture sub of fa is
begin
    sum <= a xor b xor cin;
    carry <= (not a) and (b or cin);
end sub;
```

```
library ieee;
use ieee.std_logic_1164.all;

entity fulladder is
    port (a,b,cin : in std_logic_vector(1 downto 0);
          sum,carry : out std_logic_vector(1 downto 0));
end fulladder;

architecture test of fulladder is
component fa is
    port (a,b,cin : in std_logic;
          sum,carry : out std_logic);
end component;
```

```

for u1: fa use entity work.fa(sub);
for u2: fa use entity work.fa(adder);
begin
    u1: fa port map (a(0),b(0),cin(0),sum(0),carry(0));
    u2: fa port map (a(1),b(1),cin(1),sum(1),carry(1));
end test;

```

3. Specify an individual configuration specification statement followed by a for others statement:

```

library ieee;
use ieee.std_logic_1164.all;

entity fa is
    port (a,b,cin : in std_logic;
          sum,carry : out std_logic);
end fa;

architecture adder of fa is
begin
    sum <= a xor b xor cin;
    carry <= ((a and b) or (a and cin) or (b and cin));
end adder;

architecture sub of fa is
begin
    sum <= a xor b xor cin;
    carry <= (not a) and (b or cin);
end sub;

-----
library ieee;
use ieee.std_logic_1164.all;

entity fulladder is
    port (a,b,cin : in std_logic_vector(1 downto 0);
          sum,carry : out std_logic_vector(1 downto 0));
end fulladder;

architecture test of fulladder is
component fa is
    port (a,b,cin : in std_logic;
          sum,carry : out std_logic);
end component;

```

```

for u1: fa use entity work.fa(sub);
for others: fa use entity work.fa(adder);
begin
    u1: fa port map (a(0),b(0),cin(0),sum(0),carry(0));
    u2: fa port map (a(1),b(1),cin(1),sum(1),carry(1));
end test;

```

CD728

**@E: Configurations for component: <fa> have already been defined.
Use FOR OTHERS instead.**

The above error occurs when an individual configuration specification is followed by a for ALL configuration specification statement (a for ALL configuration specification binds all the component instantiations which conflicts with individual configuration specification). In the test case below, a for ALL configuration specification follows the for u1 configuration specification which causes the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity fa is
    port (a,b,cin : in std_logic;
          sum,carry : out std_logic );
end fa;

architecture adder of fa is
begin
    sum <= a xor b xor cin;
    carry <= ((a and b) or (a and cin) or (b and cin));
end adder;

architecture sub of fa is
begin
    sum <= a xor b xor cin;
    carry <= (not a) and (b or cin);
end sub;

-----
library ieee;
use ieee.std_logic_1164.all;

```

```

entity fulladder is
    port (a,b,cin : in std_logic_vector(1 downto 0);
          sum,carry : out std_logic_vector(1 downto 0));
end fulladder;

architecture test of fulladder is
component fa is
    port (a,b,cin : in std_logic;
          sum,carry : out std_logic);
end component;

for u1: fa use entity work.fa(sub);
for all: fa use entity work.fa(adder);
begin
    u1: fa port map (a(0),b(0),cin(0),sum(0),carry(0));
    u2: fa port map (a(1),b(1),cin(1),sum(1),carry(1));
end test;

```

Action

To above error can be eliminated using any of the following three methods.

1. Use only a for ALL configuration specification:

```

library ieee;
use ieee.std_logic_1164.all;

entity fa is
    port (a,b,cin : in std_logic;
          sum,carry : out std_logic);
end fa;

architecture adder of fa is
begin
    sum <= a xor b xor cin;
    carry <= ((a and b) or (a and cin) or (b and cin));
end adder;

architecture sub of fa is
begin
    sum <= a xor b xor cin;
    carry <= (not a) and (b or cin);
end sub;

-----
library ieee;
use ieee.std_logic_1164.all;

```

```

entity fulladder is
    port (a,b,cin : in std_logic_vector(1 downto 0);
          sum,carry : out std_logic_vector(1 downto 0));
end fulladder;

architecture test of fulladder is
component fa is
    port (a,b,cin : in std_logic;
          sum,carry : out std_logic);
end component;

for ALL: fa use entity work.fa(sub);
begin
    u1: fa port map (a(0),b(0),cin(0),sum(0),carry(0));
    u2: fa port map (a(1),b(1),cin(1),sum(1),carry(1));
end test;

```

2. Use separate configuration specifications statements:

```

library ieee;
use ieee.std_logic_1164.all;

entity fa is
    port (a,b,cin : in std_logic;
          sum,carry : out std_logic);
end fa;

architecture adder of fa is
begin
    sum <= a xor b xor cin;
    carry <= ((a and b) or (a and cin) or (b and cin));
end adder;

architecture sub of fa is
begin
    sum <= a xor b xor cin;
    carry <= (not a) and (b or cin);
end sub;

-----
library ieee;
use ieee.std_logic_1164.all;

entity fulladder is
    port (a,b,cin : in std_logic_vector(1 downto 0);
          sum,carry : out std_logic_vector(1 downto 0));
end fulladder;

```

```

architecture test of fulladder is
component fa is
    port (a,b,cin : in std_logic;
          sum,carry : out std_logic);
end component;

for u1: fa use entity work.fa(sub);
for u2: fa use entity work.fa(adder);
begin
    u1: fa port map (a(0),b(0),cin(0),sum(0),carry(0));
    u2: fa port map (a(1),b(1),cin(1),sum(1),carry(1));
end test;

```

3. Follow the individual configuration specification statement with a for others statement:

```

library ieee;
use ieee.std_logic_1164.all;

entity fa is
    port (a,b,cin : in std_logic;
          sum,carry : out std_logic);
end fa;

architecture adder of fa is
begin
    sum <= a xor b xor cin;
    carry <= ((a and b) or (a and cin) or (b and cin));
end adder;

architecture sub of fa is
begin
    sum <= a xor b xor cin;
    carry <= (not a) and (b or cin);
end sub;

```

```

library ieee;
use ieee.std_logic_1164.all;

entity fulladder is
    port (a,b,cin : in std_logic_vector(1 downto 0);
          sum,carry : out std_logic_vector(1 downto 0));
end fulladder;

```

```

architecture test of fulladder is
component fa is
    port (a,b,cin : in std_logic;
          sum,carry : out std_logic);
end component;

for u1: fa use entity work.fa(sub);
for others: fa use entity work.fa(adder);
begin
    u1: fa port map (a(0),b(0),cin(0),sum(0),carry(0));
    u2: fa port map (a(1),b(1),cin(1),sum(1),carry(1));
end test;

```

CD729

@E: Component declaration has <2> generics but entity declares only <1> generics

The number of generics declared in the component declaration did not match the number of generics in the entity itself. For the following test case, two generics are declared in the component, but only one exists in the entity definition which results in the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity parity is
generic (bus_size : integer := 8);
    port (input_bus: in std_logic_vector (bus_size-1 downto 0);
          even_numbits, odd_numbits: out std_logic);
end parity;

architecture behave of parity is
begin
    process (input_bus)
        variable temp: std_logic; begin
            temp := '0';
            for i in input_bus'low to input_bus'high loop
                temp := temp xor input_bus(i);

```

```
        end loop;
        odd_numbits <= temp;
        even_numbits <= not temp;
    end process;
end behave;

library ieee;
use ieee.std_logic_1164.all;

entity test is
generic (bus_size : integer := 8);
    port (input_bus: in std_logic_vector (bus_size-1 downto 0);
          even, odd : out std_logic);
end;

architecture top of test is
component parity is
generic (bus_size : integer := 8; width : integer := 4);
    port (input_bus: in std_logic_vector (bus_size-1 downto 0);
          even_numbits, odd_numbits: out std_logic);
end component;

begin
    test : parity port map (input_bus => open,
                           even_numbits => even, odd_numbits => odd);
end;
```

Action

Make sure that the number of generics in the entity definition, the component definition, and the instantiation are the same. To eliminate the error, change the value in the component declaration as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity parity is
generic (bus_size : integer := 8);
    port (input_bus: in std_logic_vector (bus_size-1 downto 0);
          even_numbits, odd_numbits: out std_logic);
end parity;
```

```
architecture behave of parity is
begin
    process (input_bus)
        variable temp: std_logic; begin
            temp := '0';
            for i in input_bus'low to input_bus'high loop
                temp := temp xor input_bus(i);
            end loop;
            odd_numbits <= temp;
            even_numbits <= not temp;
        end process;
    end behave;

library ieee;
use ieee.std_logic_1164.all;

entity test is
generic (bus_size : integer := 8);
    port (input_bus: in std_logic_vector (bus_size-1 downto 0);
          even, odd : out std_logic);
end;

architecture top of test is
component parity is
generic (bus_size : integer := 8);
    port (input_bus: in std_logic_vector (bus_size-1 downto 0);
          even_numbits, odd_numbits: out std_logic);
end component;

begin
    test : parity port map (input_bus => open,
                           even_numbits => even, odd_numbits => odd);
end;
```

CD730

@W: Component declaration has <4> ports but entity declares <3> ports

There is a mismatch between the number of the ports of an entity and the corresponding component. In the test case below, entity parity has three ports and the corresponding component declaration has four ports which results in the warning.

```
library ieee;
use ieee.std_logic_1164.all;

entity parity is
    generic (bus_size : integer );
    port (input_bus: in std_logic_vector (bus_size-1 downto 0);
          even_numbits, odd_numbits : out std_logic);
end parity;

architecture behave of parity is
begin
    process (input_bus)
        variable temp: std_logic;
    begin
        temp := '0';
        for i in input_bus'low to input_bus'high loop
            temp := temp xor input_bus(i);
        end loop;
        odd_numbits <= temp;
        even_numbits <= not temp;
    end process;
end behave;

-----
library ieee;
use ieee.std_logic_1164.all;

entity test is
    generic (top_size : integer:=8 );
    port (input_bus: in std_logic_vector (top_size-1 downto 0);
          even, odd: out std_logic);
end;

architecture top of test is
component parity is
    generic (bus_size: integer);
    port (input_bus,input_bus2: in std_logic_vector
          (bus_size-1 downto 0);
          even_numbits, odd_numbits: out std_logic);
end component;

begin
    test: parity generic map (bus_size => 8) port map
        (input_bus =>input_bus, even_numbits => even,
         odd_numbits => odd);
end;
```

Action

Make sure that the same number of ports are defined in the entity and component declarations as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity parity is
    generic (bus_size : integer);
    port (input_bus: in std_logic_vector (bus_size-1 downto 0);
          even_numbits, odd_numbits : out std_logic);
end parity;

architecture behave of parity is
begin
    process (input_bus)
    variable temp: std_logic;
    begin
        temp := '0';
        for i in input_bus'low to input_bus'high loop
            temp := temp xor input_bus(i);
        end loop;
        odd_numbits <= temp;
        even_numbits <= not temp;
    end process;
end behave;

-----
library ieee;
use ieee.std_logic_1164.all;

entity test is
    generic (top_size : integer:=8);
    port (input_bus: in std_logic_vector (top_size-1 downto 0);
          even, odd: out std_logic);
end;

architecture top of test is
component parity is
    generic (bus_size: integer );
    port (input_bus: in std_logic_vector (bus_size-1 downto 0);
          even_numbits, odd_numbits: out std_logic);
end component;
```

```
begin
    test: parity generic map (bus_size => 8) port map
        (input_bus =>input_bus, even_numbits => even,
         odd_numbits => odd);
end;
```

CD731

@E: Port <*out3*> does not exist in component <*mycomp*>

The port map statement in a configuration specification referenced a non-existent port of the component. In the test case below, the port map statement in the configuration specification references non-existent port out3 of component my_comp which causes error.

```
library ieee;
use ieee.std_logic_1164.all;

entity mycomp is
generic (size: integer:=2);
    port (a,b: in std_logic_vector(size-1 downto 0);
          c,d: out std_logic_vector(size-1 downto 0));
end mycomp;

architecture myarch of mycomp is
begin
    c <= a and b;
    d <= a or b;
end myarch;

library ieee;
use ieee.std_logic_1164.all;

entity myTopDesign is
    port (in1,in2: in std_logic_vector(1 downto 0);
          out1,out2: out std_logic_vector(1 downto 0));
end myTopDesign;

architecture myarch2 of myTopDesign is
component mycomp
    port (in1,in2: in std_logic_vector(1 downto 0);
          out1,out2: out std_logic_vector(1 downto 0));
end component;
```

```
for inst1: mycomp use entity work.mycomp(myarch)
port map (a=>in1,b=>in2,c=>out1,d=>out3);
begin
inst1: mycomp port map (in1=>in1,in2=>in2,out1=>out1,out2=>out2);
end myarch2;
```

Action

Make sure that the port map statement in the configuration specification references only specified ports of the component as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity mycomp is
generic (size: integer:=2);
port (a,b: in std_logic_vector(size-1 downto 0);
      c,d: out std_logic_vector(size-1 downto 0));
end mycomp;

architecture myarch of mycomp is
begin
  c <= a and b;
  d <= a or b;
end myarch;

library ieee;
use ieee.std_logic_1164.all;

entity myTopDesign is
port (in1,in2: in std_logic_vector(1 downto 0);
      out1,out2: out std_logic_vector(1 downto 0));
end myTopDesign;

architecture myarch2 of myTopDesign is
component mycomp
port (in1,in2: in std_logic_vector(1 downto 0);
      out1,out2: out std_logic_vector(1 downto 0));
end component;

for inst1: mycomp use entity work.mycomp(myarch)
port map (a=>in1,b=>in2,c=>out1,d=>out2);
begin
inst1: mycomp port map (in1=>in1,in2=>in2,out1=>out1,out2=>out2);
end myarch2;
```

CD737

@E: Could not find deferred binding for architecture <st> of entity <ha>

The architecture referenced in an entity instantiation statement was not an architecture of the entity to be instantiated. In the test case below, the entity instantiation statement references entity HA with architecture st. However, because architecture st is not declared as an architecture of entity HA, the compiler errors out.

```
entity HA is
    port (U,V: in bit;
          X,Y: out bit);
end HA;

architecture str of HA is
begin
    x<=u xor v;
    y<=u and v;
end str;

entity comb is
    port (A, B: in bit;
          SUM, CARRY: out bit);
end comb;

architecture beh of comb is
begin
    ul: entity work.HA(st) port map (A, B, SUM,CARRY);
end beh;
```

Action

Be sure to specify the correct name of the declared architecture. In the corrected test case below, the architecture name in the entity instantiation statement is changed to str to match the name of the declared architecture in entity HA.

```
entity HA is
    port (U,V: in bit;
          X,Y: out bit);
end HA;
```

```
architecture str of HA is
begin
    x<=u xor v;
    y<=u and v;
end str;

entity comb is
    port (A, B: in bit;
          SUM, CARRY: out bit);
end comb;

architecture beh of comb is
begin
    u1: entity work.HA(str) port map (A, B, SUM,CARRY);
end beh;
```

CD797

@W: Incomplete case generate statement - add more cases or a when others

The case generate statement in the HDL code has at least one case choice that was not defined.

For the following test case, genval has four possible values ("00", "01", "10", "11"), but only two are listed in the case generate statement ("00" and "11").

```
library IEEE;
use IEEE.std_logic_1164.all;

entity myTopDesign is
    generic (genval: bit_vector(1 downto 0) := "10");
    port (in1: in bit; out1: out bit);
end myTopDesign;

architecture myarch2 of myTopDesign is
begin
```

```
a1: case genval generate
    when "00" =>
        out1 <= in1;
    when "11" =>
        out1 <= not in1;
    end generate;

end myarch2;
```

Action

Make sure that when clauses inside the HDL code are complete. Two possible ways to make the case generate statement complete are shown below.

You can add additional when clauses:

```
a1: case genval generate
    when "00" =>
        out1 <= in1;
    when "01" =>
        out1 <= '0';
    when "10" =>
        out1 <= in1;
    when "11" =>
        out1 <= not in1;
    end generate;
```

You can add a when others clause:

```
a1: case genval generate
    when "00" =>
        out1 <= in1;
    when "11" =>
        out1 <= not in1;
    when others =>
        out1 <= '1';
    end generate;
```

CD829

@E: Expecting library name

The library name was not specified while declaring the library in VHDL. In the following test case, the VHDL compiler is expecting a library name for the library declaration and errors out when a library name is not found.

```
library;
use ieee.std_logic_1164.all;

entity adder is
    port (a, b, cin: in std_logic;
          sum, cout: out std_logic);
end adder;

architecture behave of adder is
begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;
```

Action

Make sure that the library name is declared in the library clause as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity adder is
    port (a, b, cin: in std_logic;
          sum, cout: out std_logic);
end adder;

architecture behave of adder is
begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;
```

CD840

@E: Left argument of exponentiation must evaluate to an integer power of 2

Both the VHDL compiler and the Verilog compiler currently support exponentiation for base 2 only. The exponent must be a positive number and represent the number of left shifts to the value 2. If the base is not 2, the compiler reports the above error.

Verilog Test Case

In the following Verilog test case, the base for the assign statement is a variable which causes the Verilog compiler to error out with the above message.

```
module error (
    input clk,
    input [3:0] a, b,
    output reg [4:0] q);
    wire [2:0] temp;

    always@(posedge clk)
    begin
        q <= temp | a;
    end

    assign temp = a ** b;
endmodule
```

VHDL Test Case

In the following VHDL test case, exponentiation in the temp statement is not base 2 which causes the VHDL compiler to error out with the above message.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity test is
  port (
    clk: in std_logic;
    a_in, b_in: in integer range 0 to 15;
    d_out: out integer
  );
end test;

architecture beh of test is
signal temp : integer;
begin
  process (clk)
  begin
    if rising_edge(clk) then
      d_out <= temp;
    end if;
  end process;
  temp <= a_in ** b_in;
end beh;
```

Action

Use an exponentiation operator for base 2.

Corrected Verilog Test Case

Replace a with 2 in the assign statement.

```
module test (
  input clk,
  input [3:0] a, b,
  output reg [4:0] q);
  wire [2:0] temp;

  always@(posedge clk)
  begin
    q <= temp | a;
  end

  assign temp = 2 ** b;
endmodule
```

Corrected VHDL Test Case

Replace a_in with 2 in the temp statement.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (
        clk: in std_logic;
        a_in, b_in: in integer range 0 to 15;
        d_out: out integer
    );
end test;

architecture beh of test is
signal temp : integer;
begin
    process (clk)
    begin
        if rising_edge(clk) then
            d_out <= temp;
        end if;
    end process;
    temp <= 2 ** b_in;
end beh;

```

Exponentiation is only supported for base 2. If the two arguments are positive constant integers, the exponentiation is calculated by the tool. If one of them is a variable, then it can only be the right argument of the exponentiation function as in the function 2^{**b} .

CD844

@E: Cannot implement 'pred of left-most bound in physical type

VHDL includes a predefined set of attributes that provides information about the values included for certain types. The attributes associated with physical types are 'pred, 'succ, 'leftof, and 'rightof.

The above error occurs when the 'pred attribute is used with the left-most value defined in the physical type. T'pred(x) is defined as the value in T at a position that is one less than that of x. For physical types, the position numbers start at zero for the first element listed and increase by one for each element to the right. In the following test case, DISTANCE'pred(0 meters) is illegal as there is no value present at the position one less than 0 meters

```
library ieee;
use ieee.std_logic_1164.all;

entity myinst is
    port (in1: in bit_vector(2 downto 0);
          out1: out bit_vector(2 downto 0));
end myinst;

architecture instArch of myinst is
    type DISTANCE is range 0 to 10000 units
        meters;
        kilometers = 1000 meters;
    end units;
    signal disVal: DISTANCE;

begin
    disVal <= DISTANCE'pred(0 meters);
    out1 <= not in1;
end instArch;
```

Action

This is a syntax error. Make sure that the 'pred is not used in association with the left-most element of the physical type. To eliminate this error in the above test case, change the 'pred reference as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity myinst is
    port (in1: in bit_vector(2 downto 0);
          out1: out bit_vector(2 downto 0));
end myinst;

architecture instArch of myinst is
    type DISTANCE is range 0 to 10000 units
        meters;
        kilometers = 1000 meters;
    end units;
    signal disVal: DISTANCE;

begin
    disVal <= DISTANCE'pred(0);
    out1 <= not in1;
end instArch;
```

CD845

@E: Cannot implement 'leftof of left-most bound in physical type

VHDL includes a predefined set of attributes that provides information about the values included for certain types. The attributes associated with physical types are 'pred, 'succ, 'leftof, and 'rightof.

The above error occurs when the 'leftof attribute is used with the left-most value defined in the physical type. T'pred(x) is defined as the value in T at a position that is one less than that of x. For physical types, the position numbers start at zero for the first element listed and increase by one for each element to the right. In the following test case, DISTANCE'leftof(0 meters) is illegal as there is no value present at the position one less than 0 meters

```
library ieee;
use ieee.std_logic_1164.all;

entity myinst is
    port (in1: in bit_vector(2 downto 0);
          out1: out bit_vector(2 downto 0));
end myinst;

architecture instArch of myinst is
    type DISTANCE is range 0 to 10000 units
        meters;
        kilometers = 1000 meters;
    end units;
    signal disVal: DISTANCE;

begin
    disVal <= DISTANCE'leftof(0 meters);
    out1 <= not in1;
end instArch;
```

Action

This is a syntax error. Make sure that the 'leftof is not used in association with the left-most element of the enumeration type. To eliminate this error in the above test case, change the 'leftof reference as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity myinst is
    port (in1: in bit_vector(2 downto 0);
          out1: out bit_vector(2 downto 0));
end myinst;

architecture instArch of myinst is
    type DISTANCE is range 0 to 10000 units
        meters;
        kilometers = 1000 meters;
    end units;
    signal disVal: DISTANCE;

begin
    disVal <= DISTANCE'leftof(2 meters);
    out1 <= not in1;
end instArch;
```

CD846

@E: Cannot implement 'succ of right-most bound in physical type

VHDL includes a predefined set of attributes that provides information about the values included for certain types. The attributes associated with physical types are 'pred', 'succ', 'leftof', and 'rightof'.

The above error occurs when the 'succ attribute is used with the right-most value defined in the physical type. T'succ(x) is defined as the value in T at position one greater than that of x where T is the data type and x is a value of that type. For physical types, the position numbers start at zero for the first element listed and increase by one for each element to the right. In the following test case, DISTANCE'succ (s1) is illegal because there is no value at a position that is greater than that of s1.

```
library ieee;
use ieee.std_logic_1164.all;

entity myinst is
    port (in1: in bit_vector(2 downto 0);
          out1: out bit_vector(2 downto 0));
end myinst;
```

```
architecture instArch of myinst is
    type DISTANCE is range 0 to 10000 units
        meters;
        kilometers = 1000 meters;
    end units;
    signal disVal: DISTANCE;

begin
    disVal <= DISTANCE'succ(10 kilometers);
    out1 <= not in1;
end instArch;
```

Action

This is a syntax error. Make sure that the 'succ' is not used in association with the right-most element of the enumeration type. To eliminate this error in the above test case, change the 'succ' reference as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity myinst is
    port (in1: in bit_vector(2 downto 0);
          out1: out bit_vector(2 downto 0));
end myinst;

architecture instArch of myinst is
    type DISTANCE is range 0 to 10000 units
        meters;
        kilometers = 1000 meters;
    end units;
    signal disVal: DISTANCE;

begin
    disVal <= DISTANCE'succ(10 kilometers);
    out1 <= not in1;
end instArch;
```

CD847

@E: Cannot implement 'rightof of right-most bound in physical type

VHDL has a predefined set of attributes that provide information about the values included for certain types. The attributes associated with physical types are 'succ, 'pred, 'leftof, and 'rightof.

The above error occurs when the 'rightof attribute is used with the right-most value defined in the physical type. T'succ(x) is defined as the value in T at position one greater than that of x where T is the data type and x is a value of that type. For physical types, the position numbers start at zero for the first element listed and increase by one for each element to the right. In the following test case, DISTANCE'rightof is illegal because there is no value at a position that is greater than **that of s1**.

```
library ieee;
use ieee.std_logic_1164.all;

entity myinst is
    port (in1: in bit_vector(2 downto 0);
          out1: out bit_vector(2 downto 0));
end myinst;

architecture instArch of myinst is
    type DISTANCE is range 0 to 10000 units
        meters;
        kilometers = 1000 meters;
    end units;
    signal disVal: DISTANCE;

begin
    disVal <= DISTANCE'rightof(10 kilometers);
    out1 <= not in1;
end instArch;
```

Action

This is a syntax error. Make sure that the 'rightof is not used in association with the right-most element of the enumeration type. To eliminate this error in the above test case, change the 'rightof reference as shown in the corrected test case below.

```

library ieee;
use ieee.std_logic_1164.all;

entity myinst is
    port (in1: in bit_vector(2 downto 0);
          out1: out bit_vector(2 downto 0));
end myinst;

architecture instArch of myinst is
    type DISTANCE is range 0 to 10000 units
        meters;
        kilometers = 1000 meters;
    end units;
    signal disVal: DISTANCE;

begin
    disVal <= DISTANCE'rightof(10 kilometers);
    out1 <= not in1;
end instArch;

```

CD865

@E: Expression ident:<a_type> does not have a position value

An expected integer or enumeration for a vector position was not found. In the test case below, a_type is a type rather than the required integer position value.

```

entity test is
    port (outVal: out bit_vector(1 downto 0));
end test;

architecture rtl of test is
    type data_t is record
        vec : bit_vector(1 downto 0);
    end record data_t;

    signal dval: data_t;
    subtype a_type is natural range 3 downto 0;
begin
    dval.vec <= (a_type => '0') & (1 => '1');
end rtl;

```

Action

Make sure that only integers (or enumeration literals) are used in position values as shown in the corrected test case below.

```
entity test is
    port (outVal: out bit_vector(1 downto 0));
end test;

architecture rtl of test is
    type data_t is record
        vec : bit_vector(1 downto 0);
    end record data_t;

    signal dval: data_t;
    subtype a_type is natural range 3 downto 0;
begin
    dval.vec <= (0 => '0') & (1 => '1');
end rtl;
```

CD870

@E: Index into null range vector

An attempt was made to index into a null range vector. In the case below, the input bit vector range is incorrectly specified as 0 downto 2 (the first range value is less than the second range value).

```
entity myinst is
    port (in1: in bit_vector(0 downto 2);
          out1: out bit);
end myinst;

architecture instArch of myinst is
begin
    out1 <= in1(1);
end instArch;
```

Action

Make sure that all vector ranges are specified correctly.

```

entity myinst is
    port (in1: in bit_vector(2 downto 0);
          out1: out bit);
end myinst;

architecture instArch of myinst is
begin
    out1 <= in1(1);
end instArch;

```

CD871

@E: Invalid range. Left side has null integer range

The integer range specified is null. In the case below, the integer range is incorrectly specified as 0 downto 2 (the first range value is less than the second range value).

```

entity myinst is
    port (in1: in integer;
          out1: out integer range 0 downto 2);
end myinst;

architecture instArch of myinst is
begin
    out1 <= in1;
end instArch;

```

Action

Make sure that all integer ranges are specified correctly.

```

entity myinst is
    port (in1: in integer;
          out1: out integer range 2 downto 0);
end myinst;

architecture instArch of myinst is
begin
    out1 <= in1;
end instArch;

```

CD876

@E: Width mismatch in store (<variable> vpsize=<size>, valuesize=<size>)

The above error occurs when an attempt is made to assign a value to a variable and the width of the value to be assigned does not match the width of the variable.

Action

Make sure that the value width matches the width of the target variable.

CD880

@E: Unable to read character. Attempt to read past end of line.

An attempt was made to read a character using the read function from std_textio.vhd when there were no more characters to be read from that line. In the following test case, the file oneChar.txt has a line with only a single character. Attempting to read a second character from the file results in the error.

```
use std.textio.all;

entity readData is
    port(
        dataOut : out character);
end entity readData;

architecture rtl of readData is
function load(fileName : string) return character is
    variable data : character ;
    variable dataLine : line ;
    file dataFile : text open READ_MODE is fileName;
```

```
begin
    readline(dataFile,dataLine);
    read(dataLine,data); -- first character read is OK
    read(dataLine,data); -- Error here because there is
    -- no second character on the line
    return data;
end load;

signal val : character := load("onechar.txt");
begin
    dataOut <= val;
end RTL;
```

CD882

@E: Unable to type expression

An expression included an incorrect type argument.

Action

Look for a prior error message describing why the expression could not be typed, for example, “No matching overload for *expression*.” If the type is not required, this error is transparent.

CD883

@E: Cannot assign a scalar to an array

A scalar is incorrectly assigned to an array. In the case below, scalar a(0) is assigned to array a(0 downto 0) which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity sub is port (
    a, b : in std_logic_vector (1 downto 0);
    x    : out std_logic_vector (1 downto 0));
end sub;

architecture arch of sub is
begin
    x(1) <= a(1) and b(1);
    x(0) <= a(0) and b(0);
end arch;

library ieee;
use ieee.std_logic_1164.all;

entity top is port (
    a, b : in std_logic_vector (1 downto 0);
    e: in bit_vector(1 downto 0);
    x    : out std_logic_vector (1 downto 0));
end top;

architecture arch of top is
begin
    U1 : entity work.sub(arch) port map (
        a(1) => a(1),
        a(0 downto 0) => a(0),
        b(1) => b(1),
        b(0) => b(0),
        x(1) => x(1),
        x(0) => x(0));
end arch;
```

Action

Make sure that the assignment of scalars and arrays is consistent with scalars assigned only to scalars and arrays assigned only to arrays.

```
library ieee;
use ieee.std_logic_1164.all;

entity sub is port (
    a, b : in std_logic_vector (1 downto 0);
    x    : out std_logic_vector (1 down to 0));
end sub;
```

```
architecture arch of sub is
begin
    x(1) <= a(1) and b(1);
    x(0) <= a(0) and b(0);
end arch;

library ieee;
use ieee.std_logic_1164.all;

entity top is port (
    a, b : in std_logic_vector (1 downto 0);
    e: in bit_vector(1 downto 0);
    x   : out std_logic_vector (1 downto 0));
end top;

architecture arch of top is
begin

U1 : entity work.sub(arch) port map (
    a(1) => a(1),
    a(0) => a(0),
    b(1) => b(1),
    b(0) => b(0),
    x(1) => x(1),
    x(0) => x(0));
end arch;
```

CD884

@E: Cannot assign an array to a scalar

An array is incorrectly assigned to a scalar. In the case below, array a(0 downto 0) is assigned to scalar a(0) which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity sub is port (
    a, b : in bit_vector (1 downto 0);
    x   : out bit_vector (1 downto 0));
end sub;
```

```

architecture arch of sub is
begin
    x(1) <= a(1) and b(1);
    x(0) <= a(0) and b(0);
end arch;

library ieee;
use ieee.std_logic_1164.all;

entity top is port (
    a, b : in bit_vector (1 downto 0);
    x     : out bit_vector (1 downto 0));
end top;

architecture arch of top is
begin
U1 : entity work.sub(arch) port map (
    a(1) => a(1),
    b(1) => b(1),
    b(0) => b(0),
    x(1) => x(1),
    x(0) => x(0),
    a(0) => a(0 downto 0));
end arch;

```

Action

Make sure that the assignment of scalars and arrays is consistent with scalars assigned only to scalars and arrays assigned only to arrays.

```

library ieee;
use ieee.std_logic_1164.all;

entity sub is port (
    a, b : in bit_vector (1 downto 0);
    x     : out bit_vector (1 downto 0));
end sub;

architecture arch of sub is
begin
    x(1) <= a(1) and b(1);
    x(0) <= a(0) and b(0);
end arch;

library ieee;
use ieee.std_logic_1164.all;

```

```
entity top is port (
    a, b : in bit_vector (1 downto 0);
    x     : out bit_vector (1 downto 0));
end top;

architecture arch of top is
begin
U1 : entity work.sub(arch) port map (
    a(1) => a(1),
    b(1) => b(1),
    b(0) => b(0),
    x(1) => x(1),
    x(0) => x(0),
    a(0) => a(0));
end arch;
```

CD886

@E: DesignWare entity <entityName> not found

The requested DesignWare entity (*entityName*) could not be found.

Action

If the entity name is correct, make sure the DesignWare foundation library is installed and that the path to the library is set correctly (the path can be set from a dc_root Tcl command or in the Design Compiler Installation Location field on the VHDL (or Verilog) panel of the Implementation Options.

CD888

@E: Decimal-specified bit strings can only contain the decimal digits 0-9

A decimal-specified (D) bit string included a character other than the digits 0 through 9. In the VHDL 2008 test case below, an underscore character is included in the string literal which results in the error.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY bitStrDec IS
    PORT (out5 : out std_logic_vector(4 downto 0));
END ENTITY bitStrDec;

ARCHITECTURE behave OF bitStrDec IS
BEGIN
    out5 <= 5D"1_9";
END ARCHITECTURE behave;
```

Action

Make sure that only the digits 0 through 9 are used in decimal string literals.

CD889

@E: Truncated bits must match remaining left-most bit for signed numbers

The values of the bits being truncated are not the same as the sign bit (for signed, bit-string literals, all truncated bits must match the sign bit). In the following VHDL 2008 test case, the bit being truncated has a value of 1, and the remaining bits of the string are 0000001 which causes the truncated ‘1’ bit not to match the ‘0’ sign bit of the remaining vector.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
```

```

ENTITY truncBitsDoc IS
    PORT (out1 : OUT std_logic_vector(6 DOWNTO 0));
END ENTITY truncBitsDoc;

ARCHITECTURE behave OF truncBitsDoc IS
BEGIN
    out1 <= 7SX"81";
END ARCHITECTURE behave;

```

Action

The example compiles successfully if out1 is changed to a value that includes a ‘1’ as the sign bit; for example, 7SX"C1" (11000001), 7SX"D1" (11010001), 7SX"E1" (11100001), or 7SX"F1" (11110001).

CD890

@E: Truncation of non-zero bits is not allowed in unsigned numbers

The bits being truncated from an unsigned number do not have a value of 0 (only bits with a value of 0 can be truncated in unsigned numbers). In the VHDL 2008 test case below, the out1 value is set to hex 86 (10000110), but the length of the string literal is truncated to only seven bits which results in the error (the high-order bit to be truncated has a value of 1).

```

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY truncBitsDoc IS
    PORT (out1 : OUT std_logic_vector(6 DOWNTO 0));
END ENTITY truncBitsDoc;

ARCHITECTURE behave OF truncBitsDoc IS
BEGIN
    out1 <= 7X"86";
END ARCHITECTURE behave;

```

Action

Either increase the truncation length or limit the value to no more than x7F (01111111).

CD892

@E: Use of DesignWare functions is not supported for package constants

A DesignWare function call is being incorrectly used to generate a constant in VHDL 2008. In the following test case, the statement:

```
constant C1 : signed (15 downto 0) := DWF_mod(A1, B1);
```

calls DesignWare function DWF_mod. Constant C1 is declared in package mypack and is expected to evaluate to a constant. The DesignWare function call results in an instance being created, but does not evaluate to a constant which results in the error.

```
library ieee, dware;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use dware.DW.foundation_arith.all;

package mypack is
    constant A1 : signed(15 downto 0) := "1000011011010010";
    constant B1 : signed(15 downto 0) := "0000011011010010";
    constant C1 : signed (15 downto 0) := DWF_mod(A1, B1);
end package;

library ieee,dware;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.mypack.all;

entity adv_007 is
    port (o1 : out signed (15 downto 0));
end entity;

architecture behv of adv_007 is
begin
    o1 <= C1;
end architecture;
```

Action

Make sure that DesignWare functions are not used to generate constants. Calling a non-DesignWare function eliminates the error.

CD893

@E: File types are not allowed as record or array elements

VHDL does not allow file types to be used in record elements. In the example below, the record element `infile` is a file type.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (outVal: out bit_vector(1 downto 0));
end test;

architecture rtl of test is
type text is file of string;
type data_t is record
    infile : text;
    bank_addr : std_logic_vector(2 downto 0);
end record data_t;

signal dval: data_t;
subtype a_type is natural range 3 downto 0;
begin
    dval.bank_addr <= "001";
end rtl;
```

Action

Make sure that file types are not included in record elements.

CD894

@E: Width mismatch -- typemark size is <size>, qualified expression size is <size>

A type mark is being converted to an expression that has a different width (the width of the type mark and the width of the expression must be the same).

Action

Change the width of the expression to match the width of the type mark.

CD896

@E: Misuse of underscore (_): should only appear between two digits

VHDL cannot interpret real numbers (with an exponent), when the underscore character (_) is not used properly. In the example below, the 2e_blk1 is used as a identifier, however, it implies a number. As a number, the (_) character cannot appear after the e (exponent); only digits can follow this character. Identifiers must begin with an alphabetic character.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is port (
    a : in std_logic;
    b : out std_logic
);
end test;

architecture arch of test is
begin
    2e_blk1 : block
    begin
        b <= a;
    end block;
end arch;
```

Action

You can only use the (_) character between two digits for VHDL real numbers (with an exponent).

CD897

@E: Type mismatch for argument of `image

VHDL; The argument must match the type specified for the `image attribute correctly. In this test, when the argument for the `image attribute is the constant const2, you must specify its type as an integer instead of std_logic_vector.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_arith.all;
use IEEE.STD_LOGIC_unsigned.all; entity Test is
  port (
    clk : in std_logic;
    reset : in std_logic;
    in1 : in std_logic_vector(15 downto 0);
    in2 : in std_logic_vector(15 downto 0);
    out1 : out std_logic_vector(15 downto 0)
  );
end Test;

architecture rtl of Test is
constant const1 : integer range 0 to 15 := 12;
constant const2 : std_logic_vector(15 downto 0) := x"1234";
begin
  process(clk, reset)
  begin
    if(reset='1') then
      out1 <= (others => '0');
    elsif(rising_edge(clk)) then
      assert false report "VALUE of const: " &
        integer'image(const2)
      out1 <= in1 xor in2;
    end if;
  end process;

end architecture;
```

Action

Convert the std_logic_vector to an integer argument, used with the `image attribute as shown below.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_arith.all;
use IEEE.STD_LOGIC_unsigned.all; entity Test is
    port (
        clk : in std_logic;
        reset : in std_logic;
        in1 : in std_logic_vector(15 downto 0);
        in2 : in std_logic_vector(15 downto 0);
        out1 : out std_logic_vector(15 downto 0)
    );
end Test;

architecture rtl of Test is
constant const1 : integer range 0 to 15 := 12;
constant const2 : std_logic_vector(15 downto 0) := x"1234";
begin
    process(clk, reset)
    begin
        if(reset='1') then
            out1 <= (others => '0');
        elsif(rising_edge(clk)) then
            assert false report "VALUE of const: " &
                integer'image(conv_integer(const2))
            out1 <= in1 xor in2;
        end if;
    end process;
end architecture;
```

CD898

@E: Duplicate function declaration or mismatch between subprogram declaration and subprogram body

A VHDL function declaration has been repeated as shown in the test case below:

```
entity dupfunc is
  port
    inval: inbit;
    outval : out bit
  );
end entity dupfunc;
-----
-----
architecture Behavioral of dupfunc is
  function max(aval: bit; bval: bit) return bit is
  begin
    return '1';
  end function max;

  function max(aval: bit; bval: bit) return bit is
  begin
    return '0';
  end function max;
begin
  outval <= inval;
end Behavioral;
```

Action

Remove one of the duplicate function declarations and run synthesis again.

CD899

@E: Type of expression must be locally static

The VHDL LRM requires that certain types of expressions be locally static. This means that the information required for the expression be determinable without executing the VHDL code. In the VHDL 2008 test case below, the aggregate expression assigned to outval contains an expression that is not locally static. The compiler cannot determine the size of the expression that bufferOp returns and generates this message.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_bit.all;
```

```
entity locallyStaticErr is
  port
  (
    inval: inbit;
    inval2: in bit_vector(2 downto 0);
    outval : out bit_vector(3 downto 0)
  );
end entity locallyStaticErr;
-----
-----
architecture Behavioral of locallyStaticErr is
  function bufferOp (bufin : bit_vector) return bit_vector is
  begin
    return bufin;
  end function bufferOp;
begin
  outval <= (inval, bufferOp(inval2));
end Behavioral;
```

Action

You must remove expressions that are not locally static, which depend on the requirements of the VHDL LRM.

CD900

@E: Use clause contains reference to uninstantiated package: <packageName>

The compiler encountered a formal (uninstantiated) generic package that was included in a use clause, which VHDL 2008 prohibits. In the test case below, myTypes is a formal (uninstantiated) package contained in a use clause:

```
package myTypes is
  generic (width: integer := 7);
  subtype nvector is bit_vector(width-1 downto 0);
end package myTypes;
```

```
use work.myTypes.all;
entity myTopDesign is
    port (in1: in nvector; out1: out nvector);
end myTopDesign;

architecture myarch2 of myTopDesign is
begin
    out1 <= in1;
end myarch2;
```

Action

You must instantiate the uninstantiated generic package first. Then, you can include this instantiated package in a `use` clause as shown in the corrected test case below:

```
package myTypes is
    generic (width: integer := 7);
    subtype nvector is bit_vector(width-1 downto 0);
end package myTypes;

package nTypes is new myTypes generic map (width => 2);
use work.nTypes.all;

entity myTopDesign is
    port (in1: in nvector; out1: out nvector);
end myTopDesign;

architecture myarch2 of myTopDesign is
begin
    out1 <= in1;
end myarch2;
```

CD902

@E: Mixing of named and positional associations in array aggregates is not allowed

VHDL does not allow mixing of the named (`4 => '0'`) and positional ("1111111") assignments in an array aggregate as shown in the test case below:

```
entity top is
port (
    out1 : out bit_vector(7 downto 0)
);
end entity;

architecture structural of top is
begin
    out1 <= (4 => '0', "1111111");
end architecture structural;
```

Action

Use only named or positional assignments (not both) in an array aggregate as shown below:

```
entity top is
port (
    out1 : out bit_vector(7 downto 0));
end entity;

architecture structural of top is
begin
    out1 <= (4 => '0', 7 downto 5 => "111", 3 downto 0 => "1111");
end architecture structural;
```

CD903

@E: Choice direction does not match that of the array aggregate

The direction specified on the left-hand side of an aggregate expression mismatches the direction used to assign its value on the right-hand side. In the VHDL 2008 test case below, the direction of the aggregate on the right-hand side is specified as `downto`, but `to` is used on the left-hand side of the expression which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;

entity choiceDirErr is
    port
    (
        inval: instd_logic;
        inval2: in std_logic_vector(3 downto 0);
        outval : out std_logic_vector(3 downto 0);
    )
end entity choiceDirErr;
-----
-----
architecture Behavioral of choiceDirErr is
begin
    (1 to 3 => outval(1 downto 0), 0 => outval(3)) <= inval2;
end Behavioral;
```

Action

You must specify the same directions for the left- and right-hand sides of the aggregate expression.

CD904

@E: First argument should be of type real constant

The COS and SIN function parameters of the IEEE MATH_REAL package must be real constant values.

Action

Use only a real constant value as the parameter in the first argument.

CD905

@E: Excessive runtimes can result from non-constant loop boundaries; use a ranged integer type to improve timing and QoR.

This error occurs when one or both bounds of a for loop are non-constant. When using non-constant loop bounds, use ranged integers that accurately reflect the range of possible values. If an unnecessarily large range is used, the tool may generate unnecessary logic, causing longer runtimes.

In this loop, zlimStart and zlimEnd are non-constant:

```
for I in zlimStart to zlimEnd loop
    if (inVec(I) = '1') then
        cnt := cnt + 1;
    end if;
end loop;
```

Action

Use a ranged integer containing the smallest and largest values for the loop bounds.

For the above loop, instead of declaring zlimStart and zlimEnd as integers, use the expected range values:

```
zlimStart: in integer range 0 to 3;
zlimEnd: in integer range 0 to 7;
```

CD906

@E: Unsupported mapping for generic <*mapping*>

This error occurs when the compiler does not support a generic mapping between a component and an entity in a configuration.

Action

Component declaration must match the entity declaration.

CD907

@E: Unsupported mapping for port <*mapping*>

This error occurs when the compiler does not support a port mapping between a component and an entity in a configuration.

Action

Component declaration must match the entity declaration.

CD912

@E: Error while reading from line to real.

This error occurs when reading a value from a file that is expected to be in a real number format, but is not.

Action

Verify that the value being read from the file is in a floating point format.

CD915

@E: Expecting conditional analysis relation (=, /=, <, <=, >, >=).

The VHDL 2019 LRM lays out which conditional analysis relational operators are allowed. This error is given when an invalid relational operator is found.

Action

Check that the relational operator used in a conditional analysis expression is valid.

CD916

@E: Expecting string literal.

Conditional analysis identifiers must be compared against quoted string values. For example,

```
'if (DEBUG_LEVEL = "2")
```

Action

Verify whether the identifier is compared against a quoted string.

CD917

@E: Expecting conditional analysis expression.

A conditional_analysis_expression as specified in section 24.2 of the VHDL 2019 LRM may use and, or, xor, xnor. Other operators are not allowed.

Action

Check to see if the conditional analysis expression is using only permitted operators.

CD918

@E: `end/`else/`elsif without corresponding `if keyword.

A conditional analysis clause must be started with a `if.

Action

Check to see if there is a missing `if before the `end/`else/`elsif clause.

CD919

@E: `elsif keyword is not allowed after `else.

Use of `elsif is not allowed after a `else clause has been used.

Action

Look to see if the `else clause of the conditional analysis expression is in the right place.

CD920

@E: Expecting valid conditional analysis directive: `if, `elsif, `else, `end.

The ` character denotes a conditional analysis directive and must be one of `if, `elsif, `else, `end

Action

Use a valid conditional analysis directive

CD921

@E: Missing `end statement. End of file was encountered without a `end for the `if clause started on line %d.

The file ended without finding the matching end statement for a conditional analysis clause.

Action

Verify whether the conditional analysis clause that was started on the specified line is ended.

CHAPTER 12

CG Messages 100 – 208

CG100

@E: Reference to unknown variable <test>

An undefined variable is used in an assignment. In the following test case, test is a signal that is used in the assignment of q, but test is not declared which causes the compiler to error out with the above message.

```
module dff(q, data1, data2, clk);
    output q;
    input data1, data2, clk;
    reg q;

    always @(posedge clk)
    begin
        q <= data1 | test;
    end

endmodule
```

Action

This is a syntax error. To correct the error, make sure that all signals are declared as either a register or net type before they are used in an assignment. To eliminate the error in the above test case, declare the signal test as shown in the corrected test case below.

```
moduledff(q, data1, data2, clk);
output q;
input data1, data2, clk;
reg q;
wire test;

always @(posedge clk)
begin
    q <= data1 | test;
end

endmodule
```

CG103

@E: Expecting expression

There is a missing, incorrect, or incomplete expression in the design. In the following test case, the missing assignment causes the compiler to error out with the above message.

```
modulederror(input[3:0]i, j,
              input clk,
              output[3:0] out2 );
reg[3:0] temp;

always@(posedge clk)
begin
    temp <= j;
end

assign out2 = ; // missing assignment
endmodule
```

Action

To eliminate the error in the above test case, make sure that there is a complete and valid expression for the assignment as shown in the corrected test case below.

```
module error(input[3:0]i, j,
             input clk,
             output[3:0] out2 );
reg[3:0] temp;
always@(posedge clk)
begin
    temp <= j;
end
assign out2 = temp; // missing assignment
endmodule
```

CG104

@W: Unsized number in concatenation is 32 bits

An unsized number is used in a concatenation. In the test case below, unsized number 'b1' within the concatenation causes the compiler to issue the above message.

```
module comb(test,a);
output [7:0] test;
input a;
assign test[6:0] = {7{'b1}};
assign test [7] = a;
endmodule
```

Action

This warning can be avoided by always using sized numbers. In the corrected test case below, the size of the number is explicitly specified (7{1'b1}).

```
module comb(test,a);
output [7:0] test;
input a;
assign test[6:0] = {7{1'b1}};
assign test [7] = a;
endmodule
```

CG105

@E: range on instances are legal in Vlog-2001 mode

Verilog 2001 allows arrays of instances to be instantiated in a single instantiation statement. The compiler supports this Verilog 2001 feature. In this case, the Verilog 2001 switch is not set in the project options. The following test case uses the Verilog 2001 array of instances feature, but the Verilog 2001 switch is not enabled.

```
module array_of_inst(clk, d, q);
    input clk;
    input [7:0] d;
    output [7:0] q;
    my_dff r[7:0] (clk, d, q);
endmodule

module my_dff(clk, d, q);
    input clk, d;
    output q;
    reg q;
    always @(posedge clk) q<= d;
endmodule
```

Action

Make sure that the Verilog 2001 switch is enabled before compiling your design. You can turn on this feature using one of the following methods:

- include the Tcl command `set_option -vlog_std v2001` in your project file
- in the interactive mode, check the small box associated with Verilog 2001 on the Verilog tab of the Implementation options accessible from the project view in the user interface.

CG108

@E: Number <1'bx> contains bit values other than 0 or 1

occurs while specifying a range expression using a sized number if the constant value is other than 0 or 1. In the range expression for inputs a,b in the test case below, the LSB part of the range is declared as 1'bx which causes the compiler to error out with the above message.

```
module comb (out, a, b);
    output [1:0]out;
    input [1:1'bx]a,b;
    assign out=~(a & b);
endmodule
```

Action

Use either 0 or 1 when specifying a range expression using sized numbers. In the corrected test case below, the LSB part of the range is declared properly as 1'b0.

```
module comb (out, a, b);
    output [1:0]out;
    input [1:1'b0]a,b;
    assign out=~(a & b);
endmodule
```

CG109

@E: Multiple reg declarations in UDP

The Verilog language allows you to specify User-Defined Primitives (UDPs). However, the compiler does not support such primitives currently. This error occurs when the compiler does a syntax check on the UDP declaration and finds multiple declarations such as in the following test case for q.

```

primitive d_edge_ff (q, clk, data);
output q;
reg q;
reg q;
input data, clk;
initial q = 0;
table
// CLK  DATA  Q (state)  Q (next)
(01)  0      : ?       : 0;
(01)  1      : ?       : 1;
(0x)  1      : 1       : 1;
(0x)  0      : 0       : 0;
//Ignore negative edge of clock
(?0)  ?      : ?       : -;
//Ignore data changes on steady clock :
?      (??)  : ?       : -;
endtable
endprimitive

```

Action

This is a syntax error. To eliminate the error, comment out one of the declarations as shown in the corrected test case below.

```

primitive d_edge_ff (q, clk, data);
output q;
reg q;
// reg q;
input data, clk;
initial q = 0;
table
// CLK  DATA  Q (state)  Q (next)
(01)  0      : ?       : 0;
(01)  1      : ?       : 1;
(0x)  1      : 1       : 1;
(0x)  0      : 0       : 0;
//Ignore negative edge of clock
(?0)  ?      : ?       : -;
//Ignore data changes on steady clock :
?      (??)  : ?       : -;
endtable
endprimitive

```

This error can occur while reading files with UDPs that are not instantiated in the design. Prior to synthesis, the tool checks syntax and also determines if the HDL code can be synthesized before synthesis begins. The compiler

currently does not support UDPs. If a UDP is to be instantiated in the final design, make sure that a synthesizable definition of the module exists to perform the function of the UDP.

CG110

**@E: Missing declaration for **

The Verilog language allows you to specify User-Defined Primitives (UDPs). However, the compiler does not support such primitives currently. In this case, the compiler performed a syntax check on a UDP declaration and found an undeclared element in the UDP terminal list.

In the test case below, terminal b of the UDP udp_or is not declared.

```
primitive udp_or(out,a,b);
  output out;
  input a ;





```

Action

Ensure that all of the terminals in the UDP terminal list are declared.

```
primitive udp_or(out,a,b);
output out;
input a , b;

table
 0 0 : 0;
? 1 : 1;
1 ? : 1;
0 x : x;
x 0 : x;
endtable
endprimitive

module top (a,b,c,out);
input a, b, c;
output out;
wire temp;
assign temp= a & b;
udp_or (out,temp,c);
endmodule
```

This error can occur while reading files with UDPs that are not instantiated in the design. Prior to synthesis, the tool checks syntax and also determines if the HDL code can be synthesized before synthesis begins. The compiler currently does not support UDPs. If a UDP is to be instantiated in the final design, make sure that a synthesizable definition of the module exists to perform the function of the UDP.

CG111

@E: Expecting terminal name

The Verilog language allows you to specify User-Defined Primitives (UDPs). However, the compiler does not support such primitives currently.

An identifier preceded by the keyword input, output, or reg must be one of the identifiers from the terminal list. Also all terminals of a UDP must be scalar (vector terminals are not allowed).

The compiler performs a syntax check on the UDP declaration and finds output port out declared as a vector which results in message.

```
primitive udp_or(out,a,b);
output [1:0] out;
//output ; //(this also gives the same message)
input a ,b;

table
  0  0  :  0;
  ?  1  :  1;
  1  ?  :  1;
  0  x  :  x;
  x  0  :  x;
endtable
endprimitive

module top (a,b,c,out);
input a, b, c;
output out;
wire temp;
assign temp= a & b;
udp_or (out,temp,c);
endmodule
```

Action

Declare the identifier out as a scalar.

```
primitive udp_or(out,a,b);
output out;
input a ,b;

table
  0  0  :  0;
  ?  1  :  1;
  1  ?  :  1;
  0  x  :  x;
  x  0  :  x;
endtable
endprimitive

module top (a,b,c,out);
input a, b, c;
output out;
wire temp;
assign temp= a & b;
udp_or (out,temp,c);
endmodule
```

This error can occur while reading files with UDPs that are not instantiated in the design. Prior to synthesis, the tool checks syntax and also determines if the HDL code can be synthesized before synthesis begins. The compiler currently does not support UDPs. If a UDP is to be instantiated in the final design, make sure that a synthesizable definition of the module exists to perform the function of the UDP.

CG112

@E: Expecting terminal name - possible misspelled name?

The Verilog language allows you to specify User-Defined Primitives (UDPs). However, the compiler does not support such primitives currently.

An identifier preceded by the keyword input, output, or reg must be one of the identifiers from the terminal list.

The compiler performed a syntax check on the UDP declaration and found that an identifier used in the terminal declaration does not match any of the identifiers in the terminal list. In the following test case, the identifier following the keyword output is out1 which does not exist in the terminal list.

```
primitive udp_or(out,a,b);
  output out1;
  input a ,b;

  table
    0  0  :  0;
    ?  1  :  1;
    1  ?  :  1;
    0  x  :  x;
    x  0  :  x;
  endtable
endprimitive

module top (a,b,c,out);
  input a, b, c;
  output out;
  wire temp;
  assign temp= a & b;
  udp_or (out,temp,c);
endmodule
```

Action

Ensure that the identifier following the keyword input, output, or reg is one of the identifiers from the terminal list as shown in the corrected test case given below.

```
primitive udp_or(out,a,b);
output out;
input a ,b;

table
  0  0  :  0;
  ?  1  :  1;
  1  ?  :  1;
  0  x  :  x;
  x  0  :  x;
endtable
endprimitive

module top (a,b,c,out);
input a, b, c;
output out;
wire temp;
assign temp= a & b;
udp_or (out,temp,c);
endmodule
```

This error can occur while reading files with UDPs that are not instantiated in the design. Prior to synthesis, the tool checks syntax and also determines if the HDL code can be synthesized before synthesis begins. The compiler currently does not support UDPs. If a UDP is to be instantiated in the final design, make sure that a synthesizable definition of the module exists to perform the function of the UDP.

CG113

**@E: Direction mismatch for terminal **

The Verilog language allows you to specify User-Defined Primitives (UDPs). However, the compiler currently does not support these primitives.

UDPs can have multiple input terminals, but only one output terminal. In this case, the compiler performs a syntax check on a UDP declaration and finds an identifier other than the left most identifier present in the output terminal declaration.

In the test case below, the compiler considers out as an output terminal as it appears left most in the terminal list, and considers a and b as input terminals. When identifier b is declared as an output terminal, the compiler errors out with the above message.

```
primitive udp_or(out,a,b);
  output b,out;
  input a ;

  table
    0 0 : 0;
    ? 1 : 1;
    1 ? : 1;
    0 x : x;
    x 0 : x;
  endtable
endprimitive

module top (a,b,c,out);
  input a, b, c;
  output out;
  wire temp;
  assign temp= a & b;
  udp_or (out,temp,c);
endmodule
```

Action

Remove identifier b from the output terminal declaration as shown in the corrected test case below.

```
primitive udp_or(out,a,b);
  output out;
  input a,b ;
```

```
table
  0  0  :  0;
  ?  1  :  1;
  1  ?  :  1;
  0  x  :  x;
  x  0  :  x;
endtable
endprimitive

module top (a,b,c,out);
  input a, b, c;
  output out;
  wire temp;
  assign temp= a & b;
  udp_or (out,temp,c);
endmodule
```

This error can occur while reading files with UDPs that are not instantiated in the design. Prior to synthesis, the tool checks syntax and also determines if the HDL code can be synthesized before synthesis begins. The compiler currently does not support UDPs. If a UDP is to be instantiated in the final design, make sure that a synthesizable definition of the module exists to perform the function of the UDP.

CG114

@E: Expecting output terminal <q>

The Verilog language allows you to specify User-Defined Primitives (UDPs). However, the compiler does not support such primitives currently.

In sequential UDPs, the initial statement can be used to initialize the output state. A 1-bit value is assigned to the output, which is declared as `reg`, and no input terminal can be initialized.

The compiler performed a syntax check on a UDP declaration and found that input terminal `d` is initialized in the initial statement (implying an output terminal) which causes the compiler to error out with the above message.

```

primitive udp_latch( q,d,clear,clk);
output q;
reg q;
input clk,clear,d;
initial
d=0;

table
//clkclr d  q  q+
? 1 ? :? :0 ;
1 0 1 :? :1 ;
1 0 0 :? :0 ;
0 ? ? :? :x ;
endtable
endprimitive

module top (a,b,c,out,out1);
input a, b, c;
inout out1;
output out;
wire temp;
assign temp= a & b;
udp_latch (out1,a,temp,c);
endmodule

```

Action

Make sure that only the output terminal is initialized inside an initial statement. The corrected test case initializes output terminal q within the initial statement.

```

primitive udp_latch( q,d,clear,clk);
output q;
reg q;
input clk,clear,d;
initial
q=0;

table
//clkclr d  q  q+
? 1 ? :? :0 ;
1 0 1 :? :1 ;
1 0 0 :? :0 ;
0 ? ? :? :x ;
endtable
endprimitive

```

```
module top (a,b,c,out,out1);
  input a, b, c;
  inout out1;
  output out;
  wire temp;
  assign temp= a & b;
  udp_latch (out1,a,temp,c);
endmodule
```

This error can occur while reading files with UDPs that are not instantiated in the design. Prior to synthesis, the tool checks syntax and also determines if the HDL code can be synthesized before synthesis begins. The compiler currently does not support UDPs. If a UDP is to be instantiated in the final design, make sure that a synthesizable definition of the module exists to perform the function of the UDP.

CG115

@E: Expecting literal initialization value

The Verilog language allows you to specify User-Defined Primitives (UDPs). However, the compiler does not support such primitives currently.

The initialization value for the output terminal must start with one of the following literals:

1'b0, 1'b1, 1'bx, 1'bX, 1'B0, 1'B1, 1'Bx, 1'BX, 1, or 0

The initialization value can also be specified in any radix format by using the appropriate radix character following the first literal. In the test case below, the compiler performs a syntax check on the UDP declaration and finds output q initialized with the illegal value a which results in message.

```
primitive udp_latch( q,d,clear,clk);
  output q;
  reg q;
  input clk,clear,d;
  initial
    q=a;
```

```

table
//clkclr d q q+
? 1 ? :? :0 ;
1 0 1 :? :1 ;
1 0 0 :? :0 ;
0 ? ? :? :x ;
endtable
endprimitive

module top (a,b,c,out,out1);
input a, b, c;
inout out1;
output out;
wire temp;
assign temp= a & b;
udp_latch (out1,a,temp,c);
endmodule

```

Action

Use the following syntax with one of the legal initialization values listed earlier.

initial *output_port_identifier* = *initialization_value* ;

In the corrected test case below, the output is initialized with a proper literal (1'bx).

```

primitive udp_latch( q,d,clear,clk);
output q;
reg q;
input clk,clear,d;
initial
q=1'bx;

table
//clkclr d q q+
? 1 ? :? :0 ;
1 0 1 :? :1 ;
1 0 0 :? :0 ;
0 ? ? :? :x ;
endtable
endprimitive

```

```
module top (a,b,c,out,out1);
  input a, b, c;
  inout out1;
  output out;
  wire temp;
  assign temp= a & b;
  udp_latch (out1,a,temp,c);
endmodule
```

This error can occur while reading files with UDPs that are not instantiated in the design. Prior to synthesis, the tool checks syntax and also determines if the HDL code can be synthesized before synthesis begins. The compiler currently does not support UDPs. If a UDP is to be instantiated in the final design, make sure that a synthesizable definition of the module exists to perform the function of the UDP.

CG117

@E: Expecting keyword table

The Verilog language allows you to specify User-Defined Primitives (UDPs). However, the compiler does not support such primitives currently.

The functionality of a UDP is described using an operations table that follows a terminal declaration or an initial statement (if present). The keyword table must precede the tabular entries.

In the test case below, the error occurs when the compiler does a syntax check on the UDP declaration and cannot find the keyword table before encountering the first table entry.

```
primitive udp_or(out,a,b);
  output out;
  input a,b ;
    0  0  :  0;
    ?  1  :  1;
    1  ?  :  1;
    0  x  :  x;
    x  0  :  x;
  endtable
endprimitive
```

```
module top (a,b,c,out);
  input a, b, c;
  output out;
  wire temp;
  assign temp= a & b;
  udp_or (out,temp,c);
endmodule
```

Action

Include the table keyword prior to the first table entry as shown in the corrected test case below.

```
primitive udp_or(out,a,b);
  output out;
  input a,b ;
  table
    0  0  :  0;
    ?  1  :  1;
    1  ?  :  1;
    0  x  :  x;
    x  0  :  x;
  endtable
endprimitive

module top (a,b,c,out);
  input a, b, c;
  output out;
  wire temp;
  assign temp= a & b;
  udp_or (out,temp,c);
endmodule
```

This error can occur while reading files with UDPs that are not instantiated in the design. Prior to synthesis, the tool checks syntax and also determines if the HDL code can be synthesized before synthesis begins. The compiler currently does not support UDPs. If a UDP is to be instantiated in the final design, make sure that a synthesizable definition of the module exists to perform the function of the UDP.

CG118

@E: Illegal transition value

The Verilog language allows you to specify User-Defined Primitives (UDPs). However, the compiler does not support such primitives currently.

Signal transitions are specified within a UDP declaration. In this case, the compiler performed a syntax check on the UDP functional body and found an illegal signal transition. In the test case below, the second row of the table includes an illegal transition on signal clk which causes the compiler to error out with the above message.

specifying the other value in the bracket.

```
primitive udp_ff( q,d,clear,clk);
  output q;
  reg q;
  input clk,clear,d;
  initial
    q=0;

  table
    // clk    clr    d  q  q+
    ?      1      ? : ? : 0 ;
    (1)   0      1 : ? : 1 ;
    (01)  0      0 : ? : 0 ;
    0      ?      ? : ? : - ;
  endtable
endprimitive

module top (a,b,c,out,out1);
  input a, b, c;
  inout out1;
  output out;
  wire temp;
  assign temp= a & b;
  udp_ff (out1,a,temp,c);
endmodule
```

Action

Specify the correct transition in the table entry. As shown in the corrected test case below, changing the table entry from (1) to (01) specifies the desired positive clock transition.

```
primitive udp_ff( q,d,clear,clk);
output q;
reg q;
input clk,clear,d;
initial
q=0;

table
// clk    clr    d q q+
?      1      ? : ? : 0 ;
(01)  0      1 : ? : 1 ;
(01)  0      0 : ? : 0 ;
0      ?      ? : ? : - ;
endtable
endprimitive

module top (a,b,c,out,out1);
input a, b, c;
inout out1;
output out;
wire temp;
assign temp= a & b;
udp_ff (out1,a,temp,c);
endmodule
```

This error can occur while reading files with UDPs that are not instantiated in the design. Prior to synthesis, the tool checks syntax and also determines if the HDL code can be synthesized before synthesis begins. The compiler currently does not support UDPs. If a UDP is to be instantiated in the final design, make sure that a synthesizable definition of the module exists to perform the function of the UDP.

CG119

@E: Expecting closing)

This error occurs when a closing parenthesis is missing in the design. For every opening parenthesis, there must be a corresponding closing parenthesis. In the following test case, the sum statement contains an open parenthesis with no closing parenthesis. A closing parenthesis is required to group the operators of the expression.

```
library ieee;
use ieee.std_logic_1164.all;

entity adder is
    port (a, b, cin: in std_logic;
          sum, cout: out std_logic );
end adder;

architecture behave of adder is
begin
    sum <= (a xor b xor cin);
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;
```

Action

Make sure that parenthesis are used in pairs. To eliminate the error in the above test case, add a closing parenthesis to the sum statement.

```
library ieee;
use ieee.std_logic_1164.all;

entity adder is
    port (a, b, cin: in std_logic;
          sum, cout: out std_logic );
end adder;

architecture behave of adder is
begin
    sum <= (a xor b xor cin);
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;
```

CG120

@E: Expecting input value

The Verilog language allows you to specify User-Defined Primitives (UDPs). However, the compiler does not support such primitives currently.

The compiler performs a syntax check on the UDP functional body and finds a missing input transition value in the operations table. In the test case below, the value for input a in missing from the second table entry which causes the compiler to error out with the above message.

```
primitive udp_or(out,a,b);
  output out;
  input a,b ;
  table
    0  0  :  0;
    1  :  1;
    1  ?  :  1;
    0  x  :  x;
    x  0  :  x;
  endtable
endprimitive

module top (a,b,c,out,out1);
  input a, b, c;
  inout out1;
  output out;
  wire temp;
  assign temp= a & b;
  udp_or (out1,a,temp,c);
endmodule
```

Action

Add the proper value for the missing input.

```
primitive udp_or(out,a,b);
  output out;
  input a,b ;
```

```

table
  0  0  :  0;
  ?  1  :  1;
  1  ?  :  1;
  0  x  :  x;
  x  0  :  x;
endtable
endprimitive

module top (a,b,c,out,out1);
input a, b, c;
inout out1;
output out;
wire temp;
assign temp= a & b;
udp_or (out1,a,temp,c);
endmodule

```

This error can occur while reading files with UDPs that are not instantiated in the design. Prior to synthesis, the tool checks syntax and also determines if the HDL code can be synthesized before synthesis begins. The compiler currently does not support UDPs. If a UDP is to be instantiated in the final design, make sure that a synthesizable definition of the module exists to perform the function of the UDP.

CG121

@E: Only one edge specification is allowed per row

The Verilog language allows you to specify User-Defined Primitives (UDPs). However, the compiler does not support such primitives currently. This error occurs when the compiler does a syntax check on the UDP declaration that has two edges on a row. The following test case has two edges defined in the first row of the table which causes the compiler to error out with the above message.

```

primitive D_EDGE_FF (Q,CLK,DATA);
  output Q;
  reg Q;
  //reg Q;
  input DATA,CLK;
  initial Q = 0;

```

```

table
// CLK      DATA  Q (state)  Q (next)
(01)(01)  0      : ?       : 0;
(01)      1      : ?       : 1;
(0x)      1      : 1       : 1;
(0x)      0      : 0       : 0;
//Ignore negative edge of clock
(?0)      ?      : ?       : -;
//Ignore data changes on steady clock :
?          (??)  : ?       : -;
endtable
endprimitive

module REG4 (CLK, DIN,DOUT);
input CLK;
input [0:3] DIN;
output [0:3]DOUT;
D_EDGE_FF DLAB0 (DOUT[0], CLK, DIN[0]);
D_EDGE_FF DLAB1 (DOUT[1], CLK, DIN[1]);
D_EDGE_FF DLAB2 (DOUT[2], CLK, DIN[2]);
D_EDGE_FF DLAB3 (DOUT[3], CLK, DIN[3]);
endmodule

```

Action

This is a syntax error. To eliminate the error in the above test case, change the clock edge definition in the first row of the table as shown in the corrected test case below.

```

primitive D_EDGE_FF (Q,CLK,DATA);
output Q;
reg Q;
//reg Q;
input DATA,CLK;
initial Q = 0;
table
// CLK      DATA  Q (state)  Q (next)
(01)      0      : ?       : 0;
(01)      1      : ?       : 1;
(0x)      1      : 1       : 1;
(0x)      0      : 0       : 0;
//Ignore negative edge of clock

```

```

        (?0)      ?      : ?      : -;
//Ignore data changes on steady clock :
        ?      (??)  : ?      : -;
endtable
endprimitive

module REG4 (CLK, DIN,DOUT);
input CLK;
input [0:3] DIN;
output [0:3]DOUT;
D_EDGE_FF DLAB0 (DOUT[0], CLK, DIN[0]);
D_EDGE_FF DLAB1 (DOUT[1], CLK, DIN[1]);
D_EDGE_FF DLAB2 (DOUT[2], CLK, DIN[2]);
D_EDGE_FF DLAB3 (DOUT[3], CLK, DIN[3]);
endmodule

```

This error can occur while reading files with UDPs that are not instantiated in the design. Prior to synthesis, the tool checks syntax and also determines if the HDL code can be synthesized before synthesis begins. The compiler currently does not support UDPs. If a UDP is to be instantiated in the final design, make sure that a synthesizable definition of the module exists to perform the function of the UDP.

CG122

@E: Expecting : before current state

The Verilog language allows you to specify User-Defined Primitives (UDPs). However, the compiler currently does not support such primitives. The compiler performed a syntax check on a UDP declaration that is missing the colon before the current-state declaration. In the following test case, a colon is missing from the first row of the table.

```

primitive D_EDGE_FF (Q,CLK,DATA);
output Q;
reg Q;
//reg Q;
input DATA,CLK;
initial Q = 0;
table
// CLK    DATA   Q (cur state)  Q (next)

```

```

(01) 0      ?      : 0;
(01) 1      : ?      : 1;
(0x) 1      : 1      : 1;
(0x) 0      : 0      : 0;
//Ignore negative edge of clock
(?0) ?      : ?      : -;
//Ignore data changes on steady clock :
?      (??) : ?      : -;
endtable
endprimitive

module REG4 (CLK, DIN,DOUT);
input CLK;
input [0:3]DIN;
output [0:3]DOUT;
D_EDGE_FF DLAB0 (DOUT[0], CLK, DIN[0]);
D_EDGE_FF DLAB1 (DOUT[1], CLK, DIN[1]);
D_EDGE_FF DLAB2 (DOUT[2], CLK, DIN[2]);
D_EDGE_FF DLAB3 (DOUT[3], CLK, DIN[3]);

endmodule

```

Action

This is a syntax error. To eliminate the error, add the required colon to the current-state declaration as shown in the corrected test case below.

```

primitive D_EDGE_FF (Q,CLK,DATA);
output Q;
reg Q;
//reg Q;
input DATA,CLK;
initial Q = 0;
table
// CLK  DATA  Q (cur state)  Q (next)
(01) 0      : ?      : 0;
(01) 1      : ?      : 1;
(0x) 1      : 1      : 1;
(0x) 0      : 0      : 0;
//Ignore negative edge of clock
(?0) ?      : ?      : -;
//Ignore data changes on steady clock :
?      (??) : ?      : -;
endtable
endprimitive

```

```
module REG4 (CLK, DIN,DOUT);
  input CLK;
  input [0:3]DIN;
  output [0:3]DOUT;
  D_EDGE_FF DLAB0 (DOUT[0], CLK, DIN[0]);
  D_EDGE_FF DLAB1 (DOUT[1], CLK, DIN[1]);
  D_EDGE_FF DLAB2 (DOUT[2], CLK, DIN[2]);
  D_EDGE_FF DLAB3 (DOUT[3], CLK, DIN[3]);
endmodule
```

This error can occur while reading files with UDPs that are not instantiated in the design. Prior to synthesis, the tool checks syntax and also determines if the HDL code can be synthesized before synthesis begins. The compiler currently does not support UDPs. If a UDP is to be instantiated in the final design, make sure that a synthesizable definition of the module exists to perform the function of the UDP.

CG123

@E: Expecting level value [<01xb?>] for current state

The Verilog language allows you to specify User-Defined Primitives (UDPs). However, the compiler does not support such primitives currently.

The compiler performed a syntax check on the UDP functional body and found a transition specified for a current state of a sequential design. The current state and output should always have level values (for example, they should have one of the values 0, 1, x, b, or '?'). In the test case below, current state q is assigned transition (01) which causes the compiler to error out with the above message.

```
primitive udp_ff( q,d,clear,clk);
  output q;
  reg q;
  input clk,clear,d;
  initial
    q=0;
```

```

table
// clk    clr    d q q+
?      1      ?: (01):0 ;
(01)  0      0 : ? : 0 ;
0      ?      ? : ? : - ;
endtable
endprimitive

module top (a,b,c,out,out1);
input a, b, c;
inout out1;
output out;
wire temp;
assign temp= a & b;
udp_ff (out1,a,temp,c);
endmodule

```

Action

This syntax error can be eliminated in the above test case by changing the transition value on the q output to a level value (i.e., use one of the values of 0, 1, x, b, or '?'). In the corrected test case below, current state q is assigned the level value ? instead of the transition value (01).

```

primitive udp_ff( q,d,clear,clk);
output q;
reg q;
input clk,clear,d;
initial
q=0;

table
// clk    clr    d q q+
?      1      ? : ? : 0 ;
(01)  0      0 : ? : 0 ;
0      ?      ? : ? : - ;
endtable
endprimitive

```

```

module top (a,b,c,out,out1);
  input a, b, c;
  inout out1;
  output out;
  wire temp;
  assign temp= a & b;
  udp_ff (out1,a,temp,c);
endmodule

```

This error can occur while reading files with UDPs that are not instantiated in the design. Prior to synthesis, the tool checks syntax and also determines if the HDL code can be synthesized before synthesis begins. The compiler currently does not support UDPs. If a UDP is to be instantiated in the final design, make sure that a synthesizable definition of the module exists to perform the function of the UDP.

CG124

@E: Expecting : before output value

The Verilog language allows you to specify User-Defined Primitives (UDPs). However, the compiler does not support such primitives currently. In this case, the compiler performed a syntax check on the UDP declaration and it is missing the colon before the output value. In the test case below, the first row of the table is missing the colon before the output value.

```

primitive D_EDGE_FF(Q,CLK,DATA);
  output Q;
  reg Q;
  //reg Q;
  input DATA,CLK;
  initial Q = 0;
  table
    // CLK      DATA      Q (state)      Q (next)
    (01) 0      : ?          : 0;
    (01) 1      : ?          : 1;
    (0x) 1      : 1          : 1;
    (0x) 0      : 0          : 0;
  //Ignore negative edge of clock

```

```

(?0) ?      : ?      : -;
//Ignore data changes on steady clock :
?      (??) : ?      : -;
endtable
endprimitive

module REG4 (CLK, DIN,DOUT);
input CLK;
input [0:3] DIN;
output [0:3]DOUT;
D_EDGE_FF DLAB0 (DOUT[0], CLK, DIN[0]);
D_EDGE_FF DLAB1 (DOUT[1], CLK, DIN[1]);
D_EDGE_FF DLAB2 (DOUT[2], CLK, DIN[2]);
D_EDGE_FF DLAB3 (DOUT[3], CLK, DIN[3]);

endmodule

```

Action

This syntax error is eliminated by adding the required colon to the Q output in the first row of the table as shown in the corrected test case below.

```

primitive D_EDGE_FF(Q,CLK,DATA);
output Q;
reg Q;
//reg Q;
input DATA,CLK;
initial Q = 0;
table
// CLK  DATA  Q (state)  Q (next)
(01)  0      : ?      : 0;
(01)  1      : ?      : 1;
(0x)  1      : 1      : 1;
(0x)  0      : 0      : 0;
//Ignore negative edge of clock
(?0) ?      : ?      : -;
//Ignore data changes on steady clock :
?      (??) : ?      : -;
endtable
endprimitive

```

```

module REG4 (CLK, DIN,DOUT);
  input CLK;
  input [0:3] DIN;
  output [0:3]DOUT;
  D_EDGE_FF DLAB0 (DOUT[0], CLK, DIN[0]);
  D_EDGE_FF DLAB1 (DOUT[1], CLK, DIN[1]);
  D_EDGE_FF DLAB2 (DOUT[2], CLK, DIN[2]);
  D_EDGE_FF DLAB3 (DOUT[3], CLK, DIN[3]);
endmodule

```

This error can occur while reading files with UDPs that are not instantiated in the design. Prior to synthesis, the tool checks syntax and also determines if the HDL code can be synthesized before synthesis begins. The compiler currently does not support UDPs. If a UDP is to be instantiated in the final design, make sure that a synthesizable definition of the module exists to perform the function of the UDP.

CG125

@E: combination UDP can't use - in output column

The Verilog language allows you to specify User-Defined Primitives (UDPs). However, the compiler does not support such primitives currently.

The – symbol denotes the condition “no change from previous state” which only applies to sequential designs. In the test case below, the – symbol appears in the output column of a combinational UDP which causes the compiler to error out with the above message.

```

primitive udp_or(out,a,b);
  output out;
  input a,b ;
  table
    0 0 : 0;
    ? 1 : -;
    1 ? : 1;
    0 x : x;
    x 0 : x;
  endtable
endprimitive

```

```
module top (a,b,c,out,out1);
  input a, b, c;
  inout out1;
  output out;
  wire temp;
  assign temp= a & b;
  udp_or (temp,a,c);
endmodule
```

Action

This syntax error can be eliminated in the above test case by making sure that the output of a combinational design is not assigned the – symbol. In the corrected test case below, the – symbol is replaced with appropriate entry (no change from previous state is not possible in a combinational design).

```
primitive udp_or(out,a,b);
  output out;
  input a,b ;
  table
    0 0 : 0;
    ? 1 : 1;
    1 ? : 1;
    0 x : x;
    x 0 : x;
  endtable
endprimitive

module top (a,b,c,out,out1);
  input a, b, c;
  inout out1;
  output out;
  wire temp;
  assign temp= a & b;
  udp_or (temp,a,c);
endmodule
```

This error can occur while reading files with UDPs that are not instantiated in the design. Prior to synthesis, the tool checks syntax and also determines if the HDL code can be synthesized before synthesis begins. The compiler currently does not support UDPs. If a UDP is to be instantiated in the final design, make sure that a synthesizable definition of the module exists to perform the function of the UDP.

CG126

@E: Expecting one of [01X] for output column

The Verilog language has the capability of specifying User-Defined Primitives (UDPs). The Verilog compiler does not support such primitives currently. The compiler performed a syntax check on the UDP declaration and encountered a literal other than 0, 1, or X in the output column as shown in the first row of the table in the following test case.

```
primitive
D_EDGE_FF (Q,CLK,DATA);
output Q;
reg Q;

//reg Q;
input DATA,CLK;
initial Q = 0;
table
// CLK    DATA    Q (state)   Q (next)
(01)  0      : ?          : "0";
(01)  1      : ?          : 1;
(0x)  1      : 1          : 1;
(0x)  0      : 0          : 0;
//Ignore negative edge of clock
(?0)  ?      : ?          : -;
//Ignore data changes on steady clock :
?      (??)  : ?          : -;
endtable
endprimitive

module REG4 (CLK, DIN, DOUT);
input CLK;
input [0:3] DIN;
output [0:3]DOUT;
D_EDGE_FF DLAB0 (DOUT[0], CLK, DIN[0]);
D_EDGE_FF DLAB1 (DOUT[1], CLK, DIN[1]);
D_EDGE_FF DLAB2 (DOUT[2], CLK, DIN[2]);
D_EDGE_FF DLAB3 (DOUT[3], CLK, DIN[3]);
endmodule
```

Action

This is a syntax error. To eliminate the error in the above test case, remove the illegal quote characters enclosing the value in the output column at the end of the first row of the table.

```

primitive
D_EDGE_FF (Q,CLK,DATA);
output Q;
reg Q;

//reg Q;
input DATA,CLK;
initial Q = 0;
table
// CLK    DATA   Q (state)   Q (next)
(01)  0      : ?          : 0;
(01)  1      : ?          : 1;
(0x)  1      : 1          : 1;
(0x)  0      : 0          : 0;
//Ignore negative edge of clock
(?0)  ?      : ?          : -;
//Ignore data changes on steady clock :
?      (??)  : ?          : -;
endtable
endprimitive

module REG4 (CLK, DIN, DOUT);
input CLK;
input [0:3] DIN;
output [0:3]DOUT;
D_EDGE_FF DLAB0 (DOUT[0], CLK, DIN[0]);
D_EDGE_FF DLAB1 (DOUT[1], CLK, DIN[1]);
D_EDGE_FF DLAB2 (DOUT[2], CLK, DIN[2]);
D_EDGE_FF DLAB3 (DOUT[3], CLK, DIN[3]);
endmodule

```

This error can occur while reading files with UDPs that are not instantiated in the design. Prior to synthesis, the tool checks syntax and also determines if the HDL code can be synthesized before synthesis begins. The compiler currently does not support UDPs. If a UDP is to be instantiated in the final design, make sure that a synthesizable definition of the module exists to perform the function of the UDP.

CG127

@E: Expecting ; at end of row

The Verilog language includes the capability of specifying User-Defined Primitives (UDPs). The Verilog compiler currently does not support such primitives. In this case, the compiler performed a syntax check on the UDP declaration and the semicolon after a row specification is missing as shown in the first row of the table in the following test case.

```
primitive D_EDGE_FF(Q,CLK,DATA);
output Q;
reg Q;
//reg Q;
input DATA,CLK;
initial Q = 0;
table
// CLK    DATA   Q (state)   Q (next)
(01) 0      : ?          : 0
(01) 1      : ?          : 1;
(0x) 1      : 1          : 1;
(0x) 0      : 0          : 0;
//Ignore negative edge of clock
(?0) ?      : ?          : -;
//Ignore data changes on steady clock :
?      (??) : ?          : -;
endtable
endprimitive

module REG4 (CLK, DIN,DOUT);
input CLK;
input [0:3] DIN;
output [0:3]DOUT;
D_EDGE_FF DLAB0 (DOUT[0], CLK, DIN[0]);
D_EDGE_FF DLAB1 (DOUT[1], CLK, DIN[1]);
D_EDGE_FF DLAB2 (DOUT[2], CLK, DIN[2]);
D_EDGE_FF DLAB3 (DOUT[3], CLK, DIN[3]);
endmodule
```

Action

This is a syntax error. To eliminate the error in the above test case, add the terminating semicolon at the end of the first row as shown in the corrected test case below.

```

primitive D_EDGE_FF(Q,CLK,DATA);
output Q;
reg Q;
//reg Q;
input DATA,CLK;
initial Q = 0;
table
// CLK    DATA   Q (state)   Q (next)
(01)  0       : ?          : 0;
(01)  1       : ?          : 1;
(0x)  1       : 1          : 1;
(0x)  0       : 0          : 0;
//Ignore negative edge of clock
(?0)  ?       : ?          : -;
//Ignore data changes on steady clock :
?      (??)   : ?          : -;
endtable
endprimitive

module REG4 (CLK, DIN,DOUT);
input CLK;
input [0:3] DIN;
output [0:3]DOUT;
D_EDGE_FF DLAB0 (DOUT[0], CLK, DIN[0]);
D_EDGE_FF DLAB1 (DOUT[1], CLK, DIN[1]);
D_EDGE_FF DLAB2 (DOUT[2], CLK, DIN[2]);
D_EDGE_FF DLAB3 (DOUT[3], CLK, DIN[3]);
endmodule

```

The compiler currently does not support UDPs. If a UDP is to be instantiated in the final design, make sure that a synthesizable definition of the module exists to perform the function of the UDP.

CG128

@E: Expecting primitive name

The Verilog language allows you to specify User-Defined Primitives (UDPs). However, the compiler does not support such primitives currently. This error occurs when the compiler finds a UDP without a name while performing a syntax check. The test case below includes a UDP declared without a name in the first line which causes the compiler to error out with the above message.

```

primitive (Q,CLK,DATA);
output Q;
reg Q;
//reg Q;
input DATA,CLK;
initial Q = 0;
table
// CLK    DATA   Q (state)   Q (next)
(01)  0      : ?          : 0;
(01)  1      : ?          : 1;
(0x)  1      : 1          : 1;
(0x)  0      : 0          : 0;
//Ignore negative edge of clock
(?0)  ?      : ?          : -;
//Ignore data changes on steady clock :
?      (??)  : ?          : -;
endtable
endprimitive

module REG4 (CLK, DIN,DOUT);
input CLK;
input [0:3] DIN;
output [0:3]DOUT;
D_EDGE_FF DLAB0 (DOUT[0], CLK, DIN[0]);
D_EDGE_FF DLAB1 (DOUT[1], CLK, DIN[1]);
D_EDGE_FF DLAB2 (DOUT[2], CLK, DIN[2]);
D_EDGE_FF DLAB3 (DOUT[3], CLK, DIN[3]);
endmodule

```

Action

This is a syntax error. To eliminate the error in the test case, add the UDP name to the primitive declaration in the first line as shown in the corrected test case below.

```

primitive D_EDGE_FF (Q,CLK,DATA);
output Q;
reg Q;
//reg Q;
input DATA,CLK;
initial Q = 0;
table
// CLK    DATA   Q (state)   Q (next)
(01)  0      : ?          : 0;
(01)  1      : ?          : 1;

```

```

(0x) 1      : 1      : 1;
(0x) 0      : 0      : 0;
//Ignore negative edge of clock
(?0) ?      : ?      : -;
//Ignore data changes on steady clock :
?      (??) : ?      : -;
endtable
endprimitive

module REG4 (CLK, DIN,DOUT);
input CLK;
input [0:3] DIN;
output [0:3]DOUT;
D_EDGE_FF DLAB0 (DOUT[0], CLK, DIN[0]);
D_EDGE_FF DLAB1 (DOUT[1], CLK, DIN[1]);
D_EDGE_FF DLAB2 (DOUT[2], CLK, DIN[2]);
D_EDGE_FF DLAB3 (DOUT[3], CLK, DIN[3]);

endmodule

```

This error can occur while reading files with UDPs that are not instantiated in the design. Prior to synthesis, the tool checks syntax and also determines if the HDL code can be synthesized before synthesis begins. The compiler currently does not support UDPs. If a UDP is to be instantiated in the final design, make sure that a synthesizable definition of the module exists to perform the function of the UDP.

CG129

@E: Expecting output terminal name

The Verilog language allows you to specify User-Defined Primitives (UDPs). However, the compiler does not support such primitives currently.

The UDP syntax does not allow output terminal declarations within a terminal list. In the test case below, output terminal q is declared in the terminal list which causes the compiler to error out with the above message.

```

primitive udp_ff( output reg q=0,d,clear,clk);
input clk,clear,d;
initial
q=0;

```

```

table
// clk    clr    d q q+
(01)   0      1 : ? : 1 ;
(01)   0      0 : ? : 0 ;
0       ?      ? : ? : - ;
endtable
endprimitive

module top (a,b,c,out);
input a, b, c;
output out;
wire temp;
assign temp= a & b;
udp_ff (out,temp,a,b,c);
endmodule

```

Action

This syntax error can be eliminated in the above test case by making sure that an output terminal declaration does not appear in the terminal list. In the corrected test case below, output port q, which was previously declared in the terminal list, is now correctly declared following the terminal list.

```

primitive udp_ff(q,d,clear,clk);
output q;
reg q;
input clk,clear,d;
initial
q=0;

table
// clk    clr    d q q+
(01)   0      1 : ? : 1 ;
(01)   0      0 : ? : 0 ;
0       ?      ? : ? : - ;
endtable
endprimitive

module top (a,b,c,out);
input a, b, c;
output out;
wire temp;
assign temp= a & b;
udp_ff (out,temp,a,b,c);
endmodule

```

This error can occur while reading files with UDPs that are not instantiated in the design. Prior to synthesis, the tool checks syntax and also determines if the HDL code can be synthesized before synthesis begins. The compiler currently does not support UDPs. If a UDP is to be instantiated in the final design, make sure that a synthesizable definition of the module exists to perform the function of the UDP.

CG130

@E: Expecting input terminal name

The Verilog language allows you to specify User-Defined Primitives (UDPs). However, the compiler does not support such primitives currently.

The UDP syntax does not allow input terminal declarations within a terminal list. In the test case below, input terminal d is declared in the terminal list which causes the compiler to error out with the above message.

```
primitive udp_ff( q,input d,clear,clk );
  output q;
  reg q;
  initial
  q=0;

  table
    // clk  clr   d    q    q+
    (01)  0      1 : ? : 1 ;
    (01)  0      0 : ? : 0 ;
    0      ?      ? : ? : - ;
  endtable
endprimitive

module top (a,b,c,out);
  input a, b, c;
  output out;
  wire temp;
  assign temp= a & b;
  udp_ff (out,temp,a,b,c);
endmodule
```

Action

This syntax error can be eliminated in the above test case by making sure that any input terminal declarations do not appear in the terminal list. In the corrected test case below, input ports q, clear, and clk which were previously declared in the terminal list, are now correctly declared following the terminal list.

```
primitive udp_ff( q,d,clear,clk);
  output q;
  reg q;
  input clk,clear,d;
  initial
    q=0;

  table
    // clk    clr    d    q    q+
    (01)  0      1 : ? : 1 ;
    (01)  0      0 : ? : 0 ;
    0      ?      ? : ? : - ;
  endtable
endprimitive

module top (a,b,c,out);
  input a, b, c;
  output out;
  wire temp;
  assign temp= a & b;
  udp_ff (out,temp,a,b,c);
endmodule
```

This error can occur while reading files with UDPs that are not instantiated in the design. Prior to synthesis, the tool checks syntax and also determines if the HDL code can be synthesized before synthesis begins. The compiler currently does not support UDPs. If a UDP is to be instantiated in the final design, make sure that a synthesizable definition of the module exists to perform the function of the UDP.

CG131

@W: Wired-OR construct is being mapped to an OR type -- possible simulation mismatch.

The compiler encountered a wor construct. The warning indicates that the unsupported wor construct is mapped to an OR gate which can cause a simulation mismatch. In the test case below, output out1 is declared as a wor net which causes the compiler to issue the above warning.

```
module top (out1,a,b);
    input a,b;
    output out1;
    wor out1;

    assign out1=a;
    assign out1=b;

endmodule
```

Action

To eliminate the above warning, use an OR gate directly and avoid using the wor construct.

```
module top (out1,a,b);
    input a,b;
    output out1;

    assign out1=a | b;
endmodule
```

CG132

@W: Wired-AND construct is being mapped to an AND type -- possible simulation mismatch.

The compiler encountered a wand construct. The warning indicates that the unsupported wand construct is mapped to an AND gate which can cause a simulation mismatch. In the test case below, output out1 is declared as a wand net which causes the compiler to issue the above warning.

```
module top (out1,a,b);
  input a,b;
  output out1;
  wand out1;
  assign out1=a;
  assign out1=b;
endmodule
```

Action

To eliminate the above warning, use an AND gate directly and avoid using the wand construct.

```
module top (out1,a,b);
  input a,b;
  output out1;
  assign out1= a & b;
endmodule
```

CG133

@W: Object <gs> is declared but not assigned. Either assign a value or remove the declaration.

A reg data type has been declared and no assignment has been made to the data type. For example, qs in the following test case is declared as a reg data type, but no assignment has been made to it.

```
module dff1(q, qb, d, clk, set, reset);
  input d, clk, set, reset;
  output q, qb;
  // declare q and qb to be reg, because assigned inside always
  reg q, qb;
  reg qs;

  always @(posedge clk or posedge set or posedge reset)
  begin
    if (reset) begin
      q <= 0;
      qb <= 1;
    end else if (set) begin
      q <= 1;
      qb <= 0;
    end else begin
      q <= d;
      qb <= ~d;
    end
  end
endmodule
```

Action

Make sure that the declared data types are assigned values in the design. In the above test case, add logic so that qs is driven and affects the outputs q and qb as shown in the corrected test case below.

```
module dff1(q, qb, d, clk, set, reset);
  input d, clk, set, reset;
  output q, qb;
  // declare q and qb to be reg, because assigned inside always
  reg q, qb;
  reg qs;

  always @(posedge clk or posedge set or posedge reset)
  begin
    if (reset) begin
      q <= 0;
      qb <= 1;
      qs <= 0;
    end else if (set) begin
      q <= 1;
      qb <= 0;
      qs <= 1;
    end else begin
      q <= d;
      qb <= ~d;
    end
  end
endmodule
```

```

    qs <= qs + 1;
    if (qs ) begin
        q <= d;
        qb <= ~d;
    end
end
endmodule

```

CG134

@W: No assignment to bit <0> of <temp>

SystemVerilog; indicates that a bit of a reg vector data type is not assigned a value in the design, but is still being used as an input to another expression. In the example below, the 0th bit of reg vector variable [2:0] temp is not assigned a value, but is being used in the instantiation statement under the generate block which results in the above warning.

```

module faN (input [1:0] a, b, input cin,
            output [1:0] sum, output cout);
    reg [2:0] temp;

    generate
        begin :b1
        genvar i;
        for ( i = 0; i < 2 ; i = i + 1)
        begin : u
            fa u1(.a(a[i]), .b(b[i]), .cin(temp[i]),
                  .sum(sum[i]),.cout(temp[i+1]));
        end
    end ;
endgenerate

assign cout = temp[2] ;
endmodule

module fa (input a,b,cin, output sum,cout);
    assign sum = a ^ b ^ cin;
    assign cout = ((a & b) | ( cin & ( a | b))) ;
endmodule

```

Because the 0 bit of temp is unconnected in the test case, its value is a “don't care.” Also, port cin is not used.

Action

Make sure that all reg data type inputs are assigned some value. In the test case below, the 0 bit of temp is assigned the value of cin prior to instantiation.

```

module faN ( input [1:0] a, b, input cin,
              output [1:0] sum, output cout);
  reg [2:0] temp;
  always @(cin)
    temp[0]=cin;

  generate
    begin :b1
      genvar i;
      for ( i = 0; i < 2 ; i = i + 1)
        begin : u
          fa u1(.a(a[i]), .b(b[i]), .cin(temp[i]),
                .sum(sum[i]),.cout(temp[i+1]));
        end
      end ;
    endgenerate

    assign cout = temp[2] ;
  endmodule

module fa (input a,b,cin, output sum,cout);
  assign sum = a ^ b ^ cin;
  assign cout = ((a & b) | ( cin & ( a | b))) ;
endmodule

```

CG136

@W: Edge and condition mismatch

The edge specified in the sensitivity list of the always block and the condition checked for inside the always block are not the same. In the following test case, the always block is triggered on the positive edge of the reset, but the condition checked within the always block is the negative level of reset.

```
module dff1(q, qb, d, clk, set, reset);
  input d, clk, set, reset;
  output q, qb;
  // declare q and qb to be reg, because assigned inside always
  reg q, qb;

  always @ (posedge clk or posedge set or posedge reset)
  begin
    if (!reset) begin
      q <= 0;
      qb <= 1;
    end else if (set) begin
      q <= 1;
      qb <= 0;
    end else begin
      q <= d;
      qb <= ~d;
    end
  end
endmodule
```

Action

Make sure that the edge specified in the `always` block and the condition for which it checks are the same. In the above test case, the edge required for the `always` block is `posedge` which requires the condition to be checked also to be positive. To eliminate the warning, remove the negative edge requirement from `reset` as shown in the corrected test case below.

```
module dff1(q, qb, d, clk, set, reset);
  input d, clk, set, reset;
  output q, qb;
  // declare q and qb to be reg, because assigned inside always
  reg q, qb;

  always @ (posedge clk or posedge set or posedge reset)
  begin
    if (reset) begin
      q <= 0;
      qb <= 1;
    end else if (set) begin
      q <= 1;
      qb <= 0;
    end else begin
```

```

    q <= d;
    qb <= ~d;
end
end
endmodule

```

With sequential element descriptions, the warning usually appears with the following error:

@E: The logic for qb does not match a standard flip-flop

CG141

@W: Creating black_box for <moduleName>

A reference is made to an undefined module or to a module that is out of the project scope. In the following test case, there is a reference to the undefined module and1. Even though the compiler indicates that it is creating a black box, the design cannot be compiled because the module is actually undefined.

```

module test (a,b,out);
input a,b;
output out;
and1(out,b,a);
endmodule

```

Action

Make sure that any referenced module is described in the scope of the project. In the corrected test case below, referenced module and1 is defined immediately following the top-level module.

```

module test (a,b,out);
input a,b;
output out;
and1(out,b,a);
endmodule

module and1 (output c, input a,b);
assign c= a&b;
endmodule

```

If you are working with UUM libraries, check for the following causes for the black box being created, and correct them before compiling again:

Cause	Action
Missing library_path	Check the Include Paths, Library Paths, Library Extensions and HDL defines Before Parsing section of the compiler log file. Fix any missing information in the lmf file before compiling again.
Missing libext	
Conflicting modules across libraries	Use the config/uselibgrouporder macro to resolve conflicts

CG142

@W: Ignoring driver for net <ds> of type supply0

A supply signal is being driven in a design (supply signals cannot be driven by any value). In the test case below, ds is declared as a net of type supply0. A buffer is attempting to drive the net with a value of 1 which results in the above warning.

```
module top(out1, out2, in1, in2);
    output out1, out2;
    input in1, in2;
    supply0 ds;

    assign out1=in1&in2;
    assign out2=in1||in2;
    buf (ds,1'b1);

endmodule
```

Action

To eliminate the above warning, make sure that no supply signals are being driven by any other signals. In the corrected test case below, the supply net is no longer driven by the buffer (signal temp is used as the buffer output).

```
module top(out1, out2, in1, in2);
output out1, out2;
input in1, in2;
supply0 ds;
wire temp;

assign out1=in1&in2;
assign out2=in1||in2;
buf (temp,1'b1);

endmodule
```

CG144

@W: Ignoring driver for net <ds> of type supply1

A supply signal is being driven in a design (supply signals cannot be driven by any value). In the test case below, ds is declared as a net of type supply1. A buffer is attempting to drive the net with a value of 1 which results in the above warning.

```
module top(out1, out2, in1, in2);
output out1, out2;
input in1, in2;
supply1 ds;

assign out1=in1&in2;
assign out2=in1||in2;
buf (ds,1'b1);

endmodule
```

Action

To eliminate the above warning, make sure that no supply signals are being driven by any other signals. In the corrected test case below, the supply net is no longer driven by the buffer (signal temp is used as the buffer output).

```

module top(out1, out2, in1, in2);
output out1, out2;
input in1, in2;
supply1 ds;
wire temp;

assign out1=in1&in2;
assign out2=in1||in2;
buf (temp,1'b1);

endmodule

```

CG146

@W: Creating black box for empty module <adder>

An empty module is instantiated. In the following test case, the module adder is an empty module instantiated in the adder8 module.

```

module adder(cout, sum, a, b, cin, drive);
parameter size = 8; /* declare a parameter. Default required */
output cout;
output [size-1 :0 ] drive;
output [size-1:0] sum /* synthesis syn_keep =1 */;
// sum uses the size parameter
input cin;
input [size-1:0] a, b; // 'a' and 'b' use the size parameter
//assign {cout, sum} = a + b + cin;
//assign drive = sum;
endmodule

module adder8(cout, sum, a, b, cin,drive );
output cout;
output [7:0] sum,drive;
input [7:0] a, b;
input cin;
adder adder_1 (cout, sum, a, b, cin, drive );
endmodule

```

Action

Make sure that the module includes the module description. In the above test case, add functionality (with assign statements) to the adder module as shown in the corrected test case below.

```
module adder(cout, sum, a, b, cin, drive);
parameter size = 8; /* declare a parameter. Default required */
output cout;
output [size-1:0 ] drive;
output [size-1:0] sum /* synthesis syn_keep =1 */;
// sum uses the size parameter
input cin;
input [size-1:0] a, b; // 'a' and 'b' use the size parameter
assign {cout, sum} = a + b + cin;
assign drive = sum;
endmodule

module adder8(cout, sum, a, b, cin,drive );
output cout;
output [7:0] sum,drive;
input [7:0] a, b;
input cin;
adder adder_1 (cout, sum, a, b, cin, drive );
endmodule
```

If the module is required to be a black box, add the synthesis `syn_black_box` directive as shown below to eliminate this warning.

```
module adder(cout,sum,a,b,cin,drive)/*synthesis syn_black_box*/;
parameter size = 8; /* declare a parameter. Default required */
output cout;
output [size-1:0 ] drive;
output [size-1:0] sum /* synthesis syn_keep =1 */;
// sum uses the size parameter
input cin;
input [size-1:0] a, b; // 'a' and 'b' use the size parameter
assign {cout, sum} = a + b + cin;
assign drive = sum;
endmodule
```

```

module adder8(cout, sum, a, b, cin,drive );
output cout;
output [7:0] sum,drive;
input [7:0] a, b;
input cin;
adder adder_1 (cout, sum, a, b, cin, drive );
endmodule

```

CG147

@E: Different branches of fork cannot write the same variables

A parallel block specified using the fork and join delimiters in Verilog cannot contain assignments to the same signal or variable. In the following test case, assignments are made to the same variable within a fork/join pair which causes the compiler to error out with the above message.

```

module error (
  input clk,
  input [3:0] a,
  output reg [2:0] q );
  reg [2:0] temp;

  always@(posedge clk)
  begin
    fork
      temp = a;
      temp = a[3:1] & a[2:0];
      q = temp;
    join
  end
endmodule

```

Action

Make sure that the variable or signal assignment is made outside the parallel block. can also be avoided by creating a sequential block within the parallel block. The following test case shows how this is done using begin and end delimiters.

```

module test1 (
    input clk,
    input [3:0] a,
    output reg [2:0] q );
    reg [2:0] temp;

    always@(posedge clk)
    begin
        fork
            begin : seq_block
                temp = a;
                temp = a[3:1] & a[2:0];
            end
            q = temp;
        join
    end
endmodule

```

CG149

@E: Invalid array access into <arrayb>

More than one dimension is accessed simultaneously. In the test case below, register array arrayb is a two-dimensional array with one dimension omitted which implies that the value is being assigned to all array elements ([1][0]..[1][255]). The missing dimension results in a syntax error which causes the compiler to error out with the above message.

```

module test (input a,b, input [3:0]a1,b1, output z,
             output [3:0] temp, input clk, in1, in2);
    reg tmp [0:1][1:0];
    reg [3:0] arrayb [7:0][0:255];

    always @(posedge clk)
    begin
        tmp[1][0] <= a ^ b;
        tmp[1][1] <= a & b;
        tmp[0][0] <= a | b;
        tmp[0][1] <= a &~ b;
        arrayb[1] = a1&b1;
    end

```

```

assign z = tmp[in1][in2];
assign temp = arrayb[1][4];
endmodule

```

Action

Make sure that all the dimensions of an array are assigned properly. In the corrected test case below, the missing dimension of arrayb is specified ([4] is assigned as the second dimension; the array element being accessed is arrayb[1][4]).

```

module test (input a,b, input [3:0]a1,b1, output z,
             output [3:0] temp, input clk, in1, in2);
    reg tmp [0:1][1:0];
    reg [3:0] arrayb [7:0][0:255];
    always @(posedge clk)
        begin
            tmp[1][0] <= a ^ b;
            tmp[1][1] <= a & b;
            tmp[0][0] <= a | b;
            tmp[0][1] <= a &~ b;
            arrayb[1][4] = a1&b1;
        end
    assign z = tmp[in1][in2];
    assign temp = arrayb[1][4];
endmodule

```

CG150

@E: Can't mix blocking and non-blocking assignments. Bit <1> of <q>

Blocking and non-blocking statements are assigned to the same 2-bit vector. Procedural statements (statements within an always or initial block) in Verilog can be one of two forms:

- blocking procedural statement (=)
- non-blocking procedural statement(<=).

Blocking procedural assignments are executed before any statements that follow them, whereas in a non-blocking assignment, the assignment to the target is not blocked, but is scheduled to occur at the end of the current time step. These two statements create different logic structures and their results are unsynthesizable when both of these type of statements are assigned to the same vector. The compiler errors out on a design that uses mixed assignment statements for the same vector.

In the test case below, both blocking and non-blocking statements define the value of bit 1 of registered output signal q within an always block statement, which causes the compiler to error out with the above message.

```
module dff1(d, clk, rst, q);
    output [1:0]q;
    input [1:0]d;
    input clk,rst;
    reg [1:0]q;

    always @(posedge clk)
    begin
        if (rst)
            q[1] <= 1'b0; // non-blocking statement
        else
            q[1] = d[1]; // blocking statement
    end
endmodule
```

Action

Use only one type of statement when making an assignment. Use blocking statements to model combinational logic, and use non-blocking statements (shown below) to model sequential logic.

```
module dff1(d, clk, rst, q);
    output [1:0]q;
    input [1:0]d;
    input clk,rst;
    reg [1:0]q;
```

```
always @(posedge clk)
begin
    if (rst)
        q[1] <= 1'b0;
    else
        q[1] <= d[1];
end
endmodule
```

CG153

@E: Synthesis of event variable <e_data> is not supported yet

The synthesis tool does not support “named event control and triggering.” In the test case below, an event (e_data) is defined and an operation is defined based on its triggering. Because named events are not supported, the compiler errors out with the above message.

```
module dff(clk,r_data,q);
    input clk;
    input r_data;
    output reg q;
    event e_data;

    always@(posedge(clk))
        begin
            if(clk)->e_data;
        end

    always @(<e_data>)
        q=r_data;

endmodule
```

Action

Make sure that named events are not used in the design. Event-based triggering can be avoided in the above test case by removing the references as shown in the test case below.

```
moduledff(clk,r_data,q);
  input clk;
  input r_data;
  output reg q;

  always@(posedge clk)
  begin
    q=r_data;
  end

endmodule
```

CG154

@E: Synthesis of real <delta> is not supported

The synthesis tool does not support the real Verilog construct. In the test case below, integer delta is defined as a real which causes the compiler to error out with the above message.

```
modulatop(a,b,c);
  input a,b;
  real delta=3.4;
  output [1:0]c;
  assign c[1]= a+delta;
  assign c[0]= b+delta;
endmodule
```

Action

Make sure that real data type variables are not used in the design. In the corrected test case below, the real value is removed and an integer value is used.

```
modulercg154(a,b,c);
  input a,b;
  wire delta=3;
  output [1:0]c;
  assign c[1]= a+delta;
  assign c[0]= b+delta;
endmodule
```

CG155

@E: Synthesis of realtime <cur_time> is not supported

This error occurs if Verilog 2001 is checked and type realtime is declared. Type realtime is used for simulation to represent time in real value and is not applicable to synthesis. The compiler errors out with this message when the type realtime is encountered.

```
module real_time (ina, inb, out);
    input ina, inb;
    output out;
    realtime cur_time;
    assign out = ina & inb;
endmodule
```

Action

Either comment out the realtime type or use translate_off/translate_on as shown in the corrected test case below.

```
module real_time (ina, inb, out);
    input ina, inb;
    output out;
    /* synthesis translate_off */
    realtime cur_time
    /* synthesis translate_on */

    assign out = ina & inb;
endmodule
```

CG156

@E: Synthesis of trireg, tri0 and tri1 variables is not supported yet

A net in the design is declared as type tri0, tri1, or trireg. In the test case below, signal t is declared as type tri0 which causes the complier to error out with the above message.

```
module testtri (a, b, c, d, x);
  input a, b, c, x;
  output d;
  tri0 t; // use wire t instead
  assign t = a & b | c;
  assign d = t | x;
endmodule
```

Action

Tri0 and tri1 are nets used to model wired logic nets such as a net with more than one driver. The particular characteristic of a tri0 or tri1 net is that if no driver is driving the net, its value is 0 (tri0) or 1 (tri1). To avoid the error in the above test case, change the tri0 declaration as shown in the corrected test case below.

```
module testtri (a, b, c, d, x);
  input a, b, c, x;
  output d;
  wire t;
  assign t = a & b | c;
  assign d = t | x;
endmodule
```

Either a pull up or pull down attribute can be used in conjunction with the port to be tied to high or low respectively when it is not driven by an internal driver.

CG157

@E: Illegal vector reference, indexing into scalar.

A scalar value is read as a vector value through a variable partial-select expression. In the test case below, input variable data1 is declared as scalar, but it is read as a vector using the variable partial-select expression “+:” (the byte1=data1[j+:8] entry causes byte1 to be assigned bits beginning with 0 until bit count 8).

```
module data_buffer (input clk, input data1,
    output reg [7:0]byte1 );
integer j=0;
always@(posedge clk)
byte1= data1[j+:8];
endmodule
```

Action

Ensure that vector variables are read as vectors and that scalar variables are read as scalar. In the corrected test case below, input variable data1 is declared as vector.

```
module data_buffer (input clk, input [7:0]data1,
    output reg [7:0]byte1 );
integer j=0;
always@(posedge clk)
byte1= data1[j+:8];
endmodule
```

CG158

@E: Output connection of a primitive must be scalar.

When instantiating Verilog primitives such as an AND gate or an OR gate, the output port of the primitive must be connected to a scalar signal. The Verilog compiler errors out with the above message when an output is connected to a vector as shown in the following test case.

```
module error(input[3:0] i, j,
            input clk,
            output[3:0] out, out2 );
reg[3:0] temp;

always@(posedge clk)
begin
    temp <= j;
end

assign out2 = temp;
and and_inst1 (out, i[3], j[0]);
endmodule
```

Action

Make sure that the output is connected to a scalar signal as shown in the corrected test case below.

```
module error(input[3:0] i, j,
              input clk,
              output[3:0] out, out2 );
  reg[3:0] temp;

  always@(posedge clk)
  begin
    temp <= j;
  end

  assign out2 = temp;
  and and_inst1 (out[0], i[3], j[0]);
endmodule
```

CG161

@E: parameter <*adder_width*> cannot be found in module <*adder*>.

A parameter is called, but cannot be found in the module specified. In the following test case, module adder_16 instantiates a module adder with explicit in-line parameter calling. However, because the adder module does not contain the same parameter name as the module, the compiler errors out with the above message.

```
// adder.v
// Creating a scalable adder
module adder(cout, sum, a, b, cin);
  parameter width = 1; /* declare a parameter. Default required
  */output cout;
  output [width -1:0] sum; // sum uses the size parameter
  input cin;
  input [width -1:0] a, b; // 'a' and 'b' use the size parameter
  assign {cout, sum} = a + b + cin;
endmodule
```

```
// adder_16.v
`include "adder.v"
module adder16(cout, sum, a, b, cin);
output cout;
/* We are also using a parameter at this level of hierarchy */
parameter size = 16; // We want a 16-bit adder
output [size -1:0] sum;
input [size -1:0] a, b;
input cin;
adder #(.adder_width(16)) my_adder (cout, sum, a, b, cin);
endmodule
```

Action

The parameter defined in the module adder is width and not adder_width as specified in the instantiation. Change the adder line in the adder_16 module to match the parameter name as shown in the corrected test case below.

```
// adder.v
// Creating a scalable adder
module adder(cout, sum, a, b, cin);
parameter width = 1; /* declare a parameter. Default required
*/output cout;
output [width -1:0] sum; // sum uses the size parameter
input cin;
input [width -1:0] a, b; // 'a' and 'b' use the size parameter
assign {cout, sum} = a + b + cin;
endmodule

// adder_16.v
`include "adder.v"
module adder16(cout, sum, a, b, cin);
output cout;
/* We are also using a parameter at this level of hierarchy */
parameter size = 16; // We want a 16-bit adder
output [size -1:0] sum;
input [size -1:0] a, b;
input cin;
adder #(.width(16)) my_adder (cout, sum, a, b, cin);
endmodule
```

CG162

@E: localparam <q> cannot be changed.

A Verilog HDL local parameter (localparam) cannot be directly modified by a defparam statement or by ordered or named parameter value assignment. The localparam keyword is used to define parameters with values that must remain unchanged. In the test case below, the defparam statement attempts to change localparam q which causes the compiler to error out with the above message.

```
module top(a,b,c);
parameter p=5;
input [p-1:0] a,b;
output [p-1:0]c;
defparam u1.q =p;
adder u1(a,b,c);
endmodule

module adder(a,b,c);
localparam q=3;
input [q-1:0] a,b;
output [q-1:0]c;
assign c= a+b;
endmodule
```

Action

Do not use the localparam keyword for parameters that can be changed in the design. In the corrected test case below, parameter q of module adder is defined as a parameter instead of a localparam to allow its value to be changed by the other module.

```
module top(a,b,c);
parameter p=5;
input [p-1:0] a,b;
output [p-1:0]c;
defparam u1.q =p;
adder u1(a,b,c);
endmodule
```

```
module adder(a,b,c);
parameter q=3;
input [q-1:0] a,b;
output [q-1:0]c;
assign c= a+b;
endmodule
```

CG163

@E: Maximum instantiation depth of 250 exceeded.

The instantiation depth exceeds the 250 limit which is caused by attempting to instantiate more than 249 hierarchies.

In the test case below, the two modules continue to instantiate each other until the limit is reached and the compiler errors out with the above message.

```
module a2(a,b,clk);
input a;
input clk;
output b;
top u1(a,b,clk);
endmodule

module top(a,b,clk);
input a;
input clk;
output b;
a2 u1(a,b,clk);
endmodule
```

Action

The instantiation depth is limited to 250. To eliminate the error in the above test case, remove the instantiation from module a2 as shown in the modified test case below.

```
module a2(a,b,clk);
  input a;
  input clk;
  output b;
  assign b = a & clk;
endmodule

module top (a,b,clk);
  input a;
  input clk;
  output b;
  a2 u1(.b(b),.a(a),.clk(clk));
endmodule
```

CG165

@E: Illegal defparam. parameter <size> cannot be found in module <adder>

A parameter is redefined using defparam, but the parameter does not exist in the module definition. In the following test case, the module adder has a parameter called width, but it is referred to as size in the defparam statement.

```
// adder.v
// Creating a scalable adder
module adder(cout, sum, a, b, cin);
  parameter width = 1; /* declare a parameter. Default required */
  output cout;
  output [width-1:0] sum; //sum uses the size parameter
  input cin;
  input [width -1:0] a, b; // 'a' and 'b' use the size parameter
  assign {cout, sum} = a + b + cin;
endmodule

// adder_8.v
`include "adder.v"
module adder8(cout, sum, a, b, cin);
  output cout;
  output [7:0] sum;
```

```
input [7:0] a, b;
input cin;
adder my_adder (cout, sum, a, b, cin);
defparam my_adder.size = 8;
endmodule
```

Action

This is a syntax error that is corrected by matching the parameter name to the name specified in the module. To eliminate the error in the above test case, change the reference in the defparam statement as shown in the corrected test case below.

```
// adder.v
// Creating a scalable adder
module adder(cout, sum, a, b, cin);
parameter width = 1; /* declare a parameter. Default required */
output cout;
output [width-1:0] sum; //sum uses the size parameter
input cin;
input [width -1:0] a, b; // 'a' and 'b' use the size parameter
assign {cout, sum} = a + b + cin;
endmodule

// adder_8.v
`include "adder.v"
module adder8(cout, sum, a, b, cin);
output cout;
output [7:0] sum;
input [7:0] a, b;
input cin;
adder my_adder (cout, sum, a, b, cin);
defparam my_adder.width = 8;
endmodule
```

CG169

@E: Fork with a disable is not synthesizable

A design has a disable statement for a fork-join block, a set of parallel blocks that requires all statements to be executed in parallel in Verilog.

The following test case generates the error because disabling fork-join blocks is not supported. All statements within the fork-join block must be executed in parallel. The test case requires that $c = a \& b$ is to be processed continuously until sel is true at which time processing stops. Because logic required to emulate this behavior cannot be created in hardware, the compiler does not support disabling fork-join blocks and results when a design has code that disables a fork-join block.

```
module error_msg (a, b, c, sel);
    input a, b, sel;
    output c;
    reg c;

    always @(a, b, c)
        fork :PAR_BLK
            c = a & b;
            if (sel)
                disable PAR_BLK;
        join
endmodule
```

Action

Recode the design to eliminate disabling the fork-join block.

CG172

@E: Task recursion is not supported!

A design has a task that calls itself. The following test case uses a recursive task which causes the compiler to error out with the above message.

```
module error_msg (a, c);
    input a;
    output c;
    reg c;
```

```
task tsk;
  input a;
  output c;
  tsk(a,c);
endtask

always@(a, c)
begin
  tsk(a,c);
end

endmodule
```

Action

Recursive tasks are not supported because no logic can be created if there are no definite bounds to the logic described in HDL. To eliminate the error, you must recode to eliminate the recursive task.

CG173

@E: Target of disable is not an enclosing block or task

One sequential (begin-end) block contains a disable statement that disables a different sequential block in the same module. In the following test case, BLK1 has a disable statement that disables BLK2 which causes the compiler to error out with the above message.

```
module error_msg (a, b, c);
  input a, b;
  output c;
  reg c;

  always
  begin:BLK1
    disable BLK2;
  end

  always @ (a, b, c)
  begin :BLK2
    c = a & b;
  end
endmodule
```

```
endmodule
```

Action

To disable a sequential (begin-end) block, the disable statement must be contained in that block. To eliminate the error in the above test case, move the BKL2 disable statement from BLK1 into BLK2 as shown in the corrected test case below.

```
module error_msg (a, b, c);
  input a, b;
  output c;
  reg c;

  always
  begin:BLK1
    // disable BLK2;
  end

  always @(a, b, c)
  begin :BLK2
    c = a & b;
    disable BLK2;
  end

endmodule
```

The synthesis tool creates hardware that emulates the behavior described in RTL. However, Verilog statements such as the disable statement, used in simulation while writing test benches to produce and control stimuli as inputs to the design, are not recommended for synthesis.

CG174

@E: Wait statements are not supported yet

A wait statement is used in the design. The following test case generates the error.

```

module error_msg (a, b, c);
  input [3:0] a, b;
  output c;
  reg c;

  always
  begin
    wait(a);
    c = a & b;
  end
endmodule

```

The synthesis tool creates hardware emulating the behavior described in RTL. However, using Verilog statements such as disable and wait are not recommended for synthesis although they are used in simulation while writing test benches to produce and control stimuli as inputs to the design.

CG179

@N: Removing redundant assignment.

The compiler encountered an assignment assigning to itself. In the following test case, the assignment qrs <= qrs indicates that if the positive clock edge does not occur, the old value of qrs is to be retained. Since a flip-flop is created for qrs, this is always true and hence the assignment qrs <= qrs is implicit and not required.

```

module dff2(qrs_out, data1, clk, set, reset);
  input [2:0] data1;
  input clk, set, reset;
  output [2:0] qrs_out;
  reg [2:0] qrs;
  wire [2:0] qrs_out;

  always @(posedge clk)
  begin
    if (clk) begin
      if (reset) begin
        qrs <= 3'b000;
      end else if (set) begin
        qrs <= 3'b001;
      end else begin

```

```
        qrs <= data1;
    end
end
else begin
    qrs <= qrs;
end
end
assign qrs_out = qrs;
endmodule
```

Action

Assignments, like the one above, are implicit and do not need to be specified in the HDL code. The compiler writes out a note saying that it is removing a redundant statement as it is unnecessary. To eliminate the note for the above test case, comment out the `else` clause as shown in the corrected test case below.

```
module dff2(qrs_out, data1, clk, set, reset);
input [2:0] data1;
input clk, set, reset;
output [2:0] qrs_out;
reg [2:0] qrs;
wire [2:0] qrs_out;

always @(posedge clk)
begin
    if (clk) begin
        if (reset) begin
            qrs <= 3'b000;
        end else if (set) begin
            qrs <= 3'b001;
        end else begin
            qrs <= data1;
        end
    end
    // else begin
    //     qrs <= qrs;
    // end
end
assign qrs_out = qrs;
endmodule
```

CG181

@E: Non-constant part select index

A vector array is indexed using variables and the index range does not evaluate to a constant. In the following test case, the index is specified in terms of a module input which causes the compiler to error out with the above message.

```
module array_test (
    input clk, en,
    input [3:0] addr, raddr,
    input [7:0] data,
    output [1:0] q3,
    output [7:0] q, q2 );
    reg [7:0] mem [15:0];
    parameter test_addr = 14;

    always@(posedge clk)
    if (en)
        mem[addr] <= data;
    assign q = mem[raddr];
    assign q2 = mem[test_addr];
    assign q3 = q2[addr:raddr-1]; // addr is not known at compile time
endmodule
```

Action

Make sure that all index ranges evaluate to a defined range at compile time. To eliminate the error in the above test case, index q2 using a fixed range as shown in the corrected test case below.

```
module array_test(
    input clk, en,
    input [3:0] addr, raddr,
    input [7:0] data,
    output [1:0] q3,
    output [7:0] q, q2);
    reg [7:0] mem [15:0];
    parameter test_addr = 14;
```

```
always@(posedge clk)
if (en)
    mem[addr] <= data;
assign q = mem[raddr];
assign q2 = mem[test_addr];
assign q3 = q2[7:6]; // select index is unambiguous
endmodule
```

CG190

@E: Assignment target for <c> must be a register or integer

Tasks can pass values as outputs based on their functionality. In this case, the output variable that is to accept a value passed by a task is not a reg or integer type. In the test case below, the task passes an output value that is stored in variable c. However, because variable c is of type wire, the compiler errors out.

```
module oper(input a,b,output c);
parameter delay=10;
wire c;
always @ (a or b)
begin
    bit_oper(a,b,c);
end

task bit_oper;
input a,b;
output c;
begin
    #delay c=a & b;
end
endtask

endmodule
```

Action

Make sure that the output value passed by a task is returned to a reg or integer type variable. In the corrected test case below, the value passed by the task is returned to variable reg c, instead of wire c.

```
module oper(input a,b,output c);
parameter delay=10;
reg c;
always @ (a or b)
begin
    bit_oper(a,b,c);
end

task bit_oper;
input a,b;
output c;
begin
    #delay c=a & b;
end
endtask

endmodule
```

CG191

@E: Cannot handle system function <\$test\$plusargs> yet

The synthesis tool does not support the named system function. The example code segment below searches a list of plusargs for the provided string, but is unsupported and causes the compiler to error out.

```
initial begin
    if ($test$plusargs("Hi")) $display("Hi guest.");
    if ($test$plusargs("age")) $display("Please enter your age.");
    if ($test$plusargs("name")) $display("Identify yourself.");
end
```

Action

Ensure your design does not contain any unsupported system functions.

CG193

@E: Divisor must be a positive constant power of 2

In a division operation, the divisor is not one of the following:

- positive integer
- constant value
- value of a power of 2

In the test case below, the division is done on two inputs, therefore the divisor is not a constant value that is known during compilation. This code is not synthesizable which causes the compiler to error out with the above message.

```
module operations (div_out, equal, a, b);
parameter size = 8;
output equal;
output [size -1:0] div_out;
input [size-1:0] a, b; // declare inputs

assign equal = a == b;
assign div_out = a/b;
endmodule
```

Action

Change the value of assign div_out to a/2 as shown in the corrected test case below.

```
module operations (div_out, equal, a, b);
parameter size = 8;
output equal;
output [size -1:0] div_out;
input [size-1:0] a, b; // declare inputs

assign equal = a == b;
assign div_out = a/2;
endmodule
```

Division is supported only for unsigned values. When both the divisor and dividend are constants, the division is calculated by the compiler. For fractional results, the integral part is taken as the result of the division, for example, $3/3 = 1$, $5/4 = 1$, $3/4 = 0$. If the divisor and dividend are not constants, the divisor must be a non-zero positive constant power of 2.

Make sure that all division in the RTL code is done so that the divisor is a positive constant power of 2. In such cases, the logic created is essentially a right shift operation of the dividend. The number of times the shift operation occurs depends on the value of the divisor. For example, if the divisor is 8 ($2^{**}3$), the shift to the right is done three times. In the above test case, change the divisor b to a positive constant power of 2, such as 1, 2, 4, 8, and so on.

CG195

@E: posedge/negedge argument must be one bit, argument has <8> bits

The argument of the posedge or negedge in the always block is a bus or a vector. In the following test case, clk is defined as a vector which causes the compiler to error out with the above message.

```
module reg8(q, data, clk, rst); // eight bit register
output [7:0] q;
input [7:0] data;
input [7:0] clk;
input rst;
reg [7:0] q;

always @(posedge clk)
begin
if (rst)
    q <= 0;
else
    q <= data;
end

endmodule
```

Action

Make sure that the arguments for posedge and/or negedge are one bit wide. To eliminate the error in the above test case, use one bit of the vector clk to trigger the always block as shown in the corrected test case below.

```

module reg8(q, data, clk, rst); // eight bit register
output [7:0] q;
input [7:0] data;
input [7:0] clk;
input rst;
reg [7:0] q;

always @(posedge clk[7])
begin
if (rst)
    q <= 0;
else
    q <= data;
end

endmodule

```

CG196

@E: Exponents with variables must be a power of 2

When calculating exponential functions, the variables must be explicitly defined as constant variables of a power of 2 as it is not possible to synthesize exponential functions with unknown variables. The Verilog compiler errors out with message when the variables in the exponential functions are not clearly defined. The variables must be a constant power of 2 so that a shift operation can be performed to implement the result.

In the test case below, x is defined as an unknown input to the exponential function and is not specifically defined as a constant variable which causes the compiler to error out with the above message.

```

module expn (x, a, b, c);
input a, b, x;
output c;
//parameter x=3; // uncomment this statement and remove
               // signal 'x' as an input signal.
assign c = 3**x & 1'b1 | a | b;
endmodule

```

Action

Use input signal x as a constant to the exponential function by removing signal x as an input signal and uncommenting the parameter statement as shown in the corrected test case below

```
module expn (a, b, c);
  input a, b;
  output c;
  parameter x=3;
  assign c = 3**x & 1'b1 | a | b;
endmodule
```

CG198

@E: Exponents with more than one variable not allowed

When computing exponential functions, the Verilog compiler does not allow operand 1 and operand 2 exponents to be defined as variables. For example, the following conditions can generate this error:

- Both operands are dynamic.
- Operand 1 is a negative constant power of 2 and operand 2 is dynamic (signed/unsigned).
- Operand 1 is a negative constant non power of 2 (1, 2, 3, ...) and operand 2 is dynamic (signed/unsigned) as shown in the test case below:

```
`ifdef synthesis
module top(din1,din2,dout1,dout2);
`else
module top_rtl (din1,din2,dout1,dout2);
`endif

input integer din1 [3:0];//signed
input logic unsigned [31:0] din2 [3:0];//unsigned
output logic signed [127:0] dout1;//signed output
output logic unsigned [127:0] dout2;//unsigned output
```

```
`ifdef synthesis
sub
`else
sub_rtl
`endif
U1 (.*);

defparam U1.vall = 2 ** 2;
defparam U1.val2 = 2 ** 3;
endmodule

`ifdef synthesis
module sub
`else
module sub_rtl
`endif

(din1,din2,dout1,dout2);
input integer din1 [3:0];//signed
input logic unsigned [31:0] din2 [3:0];//unsigned
output logic signed [127:0] dout1;//signed output
output logic unsigned [127:0] dout2;//unsigned output
parameter vall=1,val2=1;

//dynamic scenarios
assign dout1 = -3 ** din1[0][31:28];
assign dout2 = -(66/2) ** din2[3][14:11];
endmodule
```

Action

Make sure to follow the conditions below when using the exponential function:

- Operand 1 is dynamic and operand 2 is a positive constant.
- Either operand 1 or operand 2 is a constant.
- Operand 1 is a power of 2 positive integer and operand 2 is dynamic (signed/unsigned).

CG199

@E: Synthesis of real numbers not supported

The compiler does not support the real Verilog construct. In the test case below, parameter param1 is assigned the real value 3.1 which causes the compiler to error out with the above message.

```
module multiplier (operand1, operand2, result,clk);
parameter param1=3.1;
input operand1;
input operand2;
input clk;
output reg result;

always@(clk,operand1,operand2)
begin
    if (clk)
        result = (operand1 * operand2)*param1;
    else
        result = (operand1 * operand2)/param1;
end
endmodule
```

Action

Ensure that real value assignments and real data type variables are not used in the design. In the corrected test case below, real value 3.1 is replaced by integer value 3 for parameter param1.

```
module multiplier (operand1, operand2, result,clk);
parameter param1=3;
input operand1;
input operand2;
input clk;
output reg result;

always@(clk,operand1,operand2)
begin
    if (clk)
        result = (operand1 * operand2)*param1;
    else
        result = (operand1 * operand2)/param1;
end
```

```
endmodule
```

CG200

@E: Recursive task/function invocation

A design has a task or function that calls itself. The following test case generates the above error.

```
module error_msg (a, c);
  input a;
  output c;
  assign c = fnctn(a);

  function fnctn;
    input a;
    fnctn = fnctn(a);
  endfunction

endmodule
```

Action

Recursive tasks and functions are not supported in the compiler. Whenever RTL describes logic without definite bounds, such as recursive functions and tasks, the compiler cannot create hardware to emulate that behavior.

CG201

@E: Argument count mismatch for function <parity>

A function is called with a different number of arguments than is specified in the function declaration. The test case below generates the error because the function parity is called with two arguments, parity(a,b), but it is defined as having one argument by the code: input [0:31] set; shown in the test case below.

```
module error_msg (a, b, c);
    input [31:0] a, b;
    output c;
    assign c = parity(a, b);

    function parity;
        input [0:31] set;
        reg [0:3] ret;
        integer j;
        begin
            ret = 0;
            for (j= 0; j <= 31; j= j + 1)
                if (set [j] == 1)
                    ret = ret + 1;
                parity = ret % 2;
        end
    endfunction

endmodule
```

Action

To eliminate this syntax error, change the number of function arguments called in the assignment line as shown in the corrected test case below.

```
module error_msg (a, b, c);
    input [31:0] a, b;
    output c;
    assign c = parity (a);

    function parity;
        input [0:31] set;
        reg [0:3] ret;
        integer j;
        begin
            ret = 0;
            for (j= 0; j <= 31; j= j + 1)
                if (set [j] == 1)
                    ret = ret + 1;
                parity = ret % 2;
        end
    endfunction

endmodule
```

CG202

@E: Function did not assign return value

A function does not have any return value. The following test case generates the above error.

```
module error_msg (a, b, c);
    input [31:0] a, b;
    output c;
    assign c = parity(a);

    function parity;
        input [0:31] set;
        reg [0:3] ret;
        integer j;

        begin
            ret = 0;
            for (j= 0; j <= 31; j= j + 1 )
                if (set [j] == 1)
                    ret = ret + 1;
                    // parity = ret % 2;
            end
        endfunction

    endmodule
```

Action

This is a syntax error. In Verilog, the function definition declares a register internal to the function with the same name and range as the function. A function returns a value by assigning a value to this register explicitly in the function definition. An assignment to this register must therefore be present within the function definition. If no range is specified with the function definition, then a 1-bit return value is assumed. To eliminate the error in the above test case, remove the comment characters to enable the function return as shown in the corrected test case below.

```
module error_msg (a, b, c);
    input [31:0] a, b;
    output c;
    assign c = parity(a);
```

```
function parity;
  input [0:31] set;
  reg [0:3] ret;
  integer j;

begin
  ret = 0;
  for (j= 0; j <= 31; j= j + 1 )
    if (set [j] == 1)
      ret = ret + 1;
    parity = ret % 2;
  end
endfunction

endmodule
```

CG203

@E: Negative shift distance

A design has a signal shifted by a negative number. The following test case generates the above error.

```
module error26 (a, c);
  input [3:0] a;
  output [3:0] c;
  assign c = a << -2;
endmodule
```

Action

This is a syntax error. In Verilog, negative constants such as -2, -3, and so on, cannot be used in association with >> (right shift) and << (left shift) operations. Signed shift operations must use the arithmetic shift operations: <<< (arithmetic shift-left) and >>> (arithmetic shift right). Recode the design and use the arithmetic shift operations when doing signed arithmetic operations.

CG208

@E: Unknown expr type <3>

The type of the expression does not match the expected type. In the following test case, the index expression is specified as a real number which results in the message because the expression indexing the vector must evaluate to an integer.

```
module test (input[3:0] i, j,
             input clk,
             output[3:0] out, out2 );
  reg[3:0] temp;

  always@(posedge clk)
  begin
    temp = j;
  end

  assign out2 = temp;
  and (out[0], i[3.0], j[0]); // 3.0 is not an integer type
endmodule
```

Action

Make sure that all expressions evaluate to their expected types. In the test case below, signal i is correctly indexed to an integer value (3).

```
module test (input[3:0] i, j,
             input clk,
             output[3:0] out, out2 );
  reg[3:0] temp;

  always@(posedge clk)
  begin
    temp = j;
  end

  assign out2 = temp;
  and (out[0], i[3], j[0]);
endmodule
```

CHAPTER 13

CG Messages 209 – 313

CG209

@E: expecting generate item

An invalid statement or declaration was encountered within a generate statement. In the following test case, there is no end keyword before the endgenerate statement within the generate block which causes the compiler to error out with the above message.

```
module multiplier (operand1, operand2, result);
parameter operand1_width = 4, operand2_width =2;
input [operand1_width -1:0]operand1;
input [operand2_width -1:0] operand2;
output [operand1_width +operand2_width -2:0] result;
assign result = operand1 * operand2;
endmodule

module level2(input [15:0]op1,input [7:0]op2,output [22:0]result);
    generate
        begin
            multiplier #(operand1_width(16),operand2_width(8))
                test2 (op1, op2, result);
        endgenerate
    endmodule
```

Action

To eliminate the above error, make sure that there are no invalid statements within a generate block. In the corrected test case below, an end statement for the corresponding begin statement has been added inside the generate block.

```
module multiplier (operand1, operand2, result);
parameter operand1_width = 4, operand2_width =2;
input [operand1_width -1:0]operand1;
input [operand2_width -1:0] operand2;
output [operand1_width +operand2_width -2:0] result;
assign result = operand1 * operand2;
endmodule

module level2(input [15:0]op1,input [7:0]op2,output [22:0]result);
generate
begin
multiplier #(.operand1_width(16),.operand2_width(8))
test2 (op1, op2, result);
end
endgenerate
endmodule
```

CG210

@E: Expecting block name after :

The compiler detected a generate statement without a valid identifier (generate_block_identifier) specified after the “:” in the begin statement.

In the following test case, there is no identifier specified after the “:” in the begin statement under the generate-block in top-level module adder_32bit which causes the compiler to error out with the above message.

```
module adder8 (sum, c0, a, b, cin);
input [7:0]a, b;
output [7:0]sum;
output [0:0] c0;
input [0:0] cin;
assign {c0, sum} = a + b + cin;
endmodule
```

```

module adder_32bit (a, b, sum, cin_in, c0_out);
  input [31:0]a, b;
  output [31:0]sum;
  input cin_in;
  output c0_out;
  wire [4:0]c0;
  assign c0[0] = cin_in;

  generate
    genvar i;
    begin :
      for (i=0; i<=3; i =i + 1)
        begin:u
          adder8 add (sum [8*i+7:8*i], c0[i+1],
                      a [8*i+7:8*i],b[8*i+7:8*i], c0[i]);
        end
    end
  endgenerate

  assign c0_out = c0[4];
endmodule

```

Action

Specify a valid identifier (generate_block_identifier) after the “:” in the begin statement of the generate block as shown in the corrected test case below.

```

module adder8 (sum, c0, a, b, cin);
  input [7:0]a, b;
  output [7:0]sum;
  output [0:0] c0;
  input [0:0] cin;
  assign {c0, sum} = a + b + cin;
endmodule

module adder_32bit (a, b, sum, cin_in, c0_out);
  input [31:0]a, b;
  output [31:0]sum;
  input cin_in;
  output c0_out;
  wire [4:0]c0;
  assign c0[0] = cin_in;
  generate
    genvar i;
    begin:u1
      for (i=0; i<=3; i =i + 1)

```

```

begin :u
    adder8 add (sum [8*i+7:8*i], c0[i+1], a [8*i+7:8*i],
    b[8*i+7:8*i], c0[i]);
end
end
endgenerate

assign c0_out = c0[4];
endmodule

```

CG211

@E: Expecting at least one case item

The Verilog compiler expects case statements to include at least one case element. In the following test case, the case statement does not include a case element (the elements are commented out) which causes the compiler to error out with the above message.

```

module muxnew3(out, a, b, c, d, select);
output out;
input a, b, c, d;
input [3:0] select;
reg out;

always @(select or a or b or c or d)
begin
    casez (select) // synthesis full_case
    // 4'b???1: out = a;
    // 4'b??1?: out = b;
    // 4'b?1??: out = c;
    // 4'b1???: out = d;
    endcase
end
endmodule

```

Action

Make sure that all case statements used in the HDL code have at least one case branch. To eliminate the error in the above test case, uncomment the case branches as shown in the corrected test case below.

```
module muxnew3(out, a, b, c, d, select);
output out;
input a, b, c, d;
input [3:0] select;
reg out;

always @(select or a or b or c or d)
begin
    casez (select) // synthesis full_case
        4'b???1: out = a;
        4'b??1?: out = b;
        4'b?1???: out = c;
        4'b1????: out = d;
    endcase
end
endmodule
```

CG213

@W: Unrecognized attribute <parallel_case1> on case statement

An undefined compiler attribute on a case statement was encountered. In the test case below, the compiler detects unrecognized attribute parallel_case1 on the case statement which results in the above warning.

```
module comb( input [1:0]arg, output reg result,input a,b,c,d);
always @((arg,a,b,c,d))
(*parallel_case1=1*)
    case (arg)
        2'b00 : result = d;
        2'b10 : result = a;
        2'b01 : result = b;
        2'b11 : result =c;
    endcase
endmodule
```

Action

The warning can be avoided by using recognized attributes such as parallel_case or full_case on a case statement. In the modified test case below, legal attribute parallel_case is applied to the case statement.

```

module comb( input [1:0]arg, output reg result,input a,b,c,d);
always @ (arg,a,b,c,d)
(*parallel_case=1*)
  case (arg)
    2'b00 : result = d;
    2'b10 : result = a;
    2'b01 : result = b;
    2'b11 : result =c;
  endcase
endmodule

```

CG217

@E: task/functions in generate is not supported yet

A task or function definition is included within a generate statement (task/function definitions are not yet supported in generate statements. In the following test case, the function definition causes the compiler to error out with the above message.

```

module my_task (
  input clk,
  input [3:0] a,
  output [3:0] q );
  reg [3:0] temp;
  parameter gen = 0;
  generate
    function [3:0] and_4;
      input [3:0] a, b;
      and_4 = a & b;
    endfunction
    if (gen == 1)
      assign q = a | temp;
    else
      assign q = and_4(a, temp);
  endgenerate
  always@(posedge (clk) )
begin
  if (clk)
    temp <= a;
end

```

```
endmodule
```

Action

Define the function outside the generate block as shown in the corrected test case below.

```
module my_task (
    input clk,
    input [3:0] a,
    output [3:0] q );
    reg [3:0] temp;
    parameter gen = 0;
    function [3:0] and_4;
        input [3:0] a, b;
        and_4 = a & b;
    endfunction

    generate
        if (gen == 1)
            assign q = a | temp;
        else
            assign q = and_4(a, temp);
    endgenerate

    always@(posedge (clk) )
    begin
        if (clk)
            temp <= a;
    end
endmodule
```

CG219

@E: init of For generate should evaluate to constant

The compiler detects a for loop with a generate statement where the initial value of the genvar variable is not a constant. In the following test case, the genvar variable i of the for generate structure depends on the input test for the initial value. This value does not evaluate to a constant which causes the compiler to error out with the above message.

```
module adder8 (sum, c0, a, b, cin);
  input [7:0] a, b;
  output [7:0] sum;
  output [0:0] c0;
  input [0:0] cin;
  assign {c0, sum} = a + b + cin;
endmodule

module adder_32bit (a, b, sum, cin_in, c0_out, test);
  input [31:0] a, b;
  output [31:0] sum;
  input cin_in;
  input [2:0] test;
  output c0_out;
  wire [4:0] c0;
  assign c0[0] = cin_in;

  generate
    genvar i;
    for (i=test; i<=3; i = i + 1)
    begin : u
      adder8 add (sum [8*i+7:8*i], c0[i+1], a[8*i+7:8*i],
                  b[8*i+7:8*i], c0[i]);
    end
  endgenerate
endmodule
```

Action

Edit the code to assign an initial constant value to the genvar in the for loop generate statement as shown in the corrected test case below.

```
module adder8 (sum, c0, a, b, cin);
  input [7:0] a, b;
  output [7:0] sum;
  output [0:0] c0;
  input [0:0] cin;
  assign {c0, sum} = a + b + cin;
endmodule
```

```

module adder_32bit (a, b, sum, cin_in, c0_out, test);
    input [31:0] a, b;
    output [31:0] sum;
    input cin_in;
    input [2:0] test;
    output c0_out;
    wire [4:0] c0;
    assign c0[0] = cin_in;

    generate
        genvar i;
        for (i=0; i<=3; i = i + 1)
        begin : u
            adder8 add (sum [8*i+7:8*i], c0[i+1], a[8*i+7:8*i],
                         b[8*i+7:8*i], c0[i]);
        end
    endgenerate
endmodule

```

The genvar variable is an integer used as an index control within generate loops. Acceptable genvar values are positive numbers or 0. In a generate for loop, the index variable must be a genvar, both assignments in the for loop control must be assigned to the same genvar, and the contents must be within a named begin-end block.

CG220

@E: genvar for initial and increment for for-generate different?

The initial and incremental values of a for loop in a generate statement do not use the same genvar. The following test case errors out because two genvar variables are used: one for the initial value (i) and one for the increment value(j).

```

module adder8 (sum, c0, a, b, cin);
    input [7:0]a, b;
    output [7:0]sum;
    output [0:0] c0;
    input [0:0] cin;
    assign {c0, sum} = a + b + cin;
endmodule

```

```
module adder_32bit (a, b, sum, cin_in, c0_out);
    input [31:0]a, b;
    output [31:0]sum;
    input cin_in;
    output c0_out;
    wire [4:0]c0;
    assign c0[0] = cin_in;

    generate
        genvar i,j;
        for (i=0; i<=3; j = j + 1)
        begin : u
            adder8 add (sum [8*i+7:8*i], c0[i+1], a [8*i+7:8*i],
                         b[8*i+7:8*i], c0[i]);
        end
    endgenerate

    assign c0_out = c0[4];
endmodule
```

Action

Define the initial value and the increment value of the for loop with the same genvar variable as shown in the corrected test case below.

```
module adder8 (sum, c0, a, b, cin);
    input [7:0]a, b;
    output [7:0]sum;
    output [0:0] c0;
    input [0:0] cin;
    assign {c0, sum} = a + b + cin;
endmodule

module adder_32bit (a, b, sum, cin_in, c0_out);
    input [31:0]a, b;
    output [31:0]sum;
    input cin_in;
    output c0_out;
    wire [4:0]c0;
    assign c0[0] = cin_in;
```

```
generate
genvar i;
for (i=0; i<=3; i =i + 1)
begin : u
    adder8 add (sum [8*i+7:8*i], c0[i+1], a [8*i+7:8*i],
                 b[8*i+7:8*i], c0[i]);
end
endgenerate

assign c0_out = c0[4];
endmodule
```

The genvar variable is an integer used as an index control within generate loops. Acceptable genvar values are positive numbers or 0. In a generate for loop, the index variable must be a genvar, both assignments in the for loop control must be assigned to the same genvar, and the contents must be within a named begin-end block.

CG221

@E: increment of For-generate should evaluate to constant

When the compiler detects a for loop generate statement in which the increment of the loop cannot be evaluated as a constant, the compiler errors out with the above message.

The following test case generates because the increment of the genvar variable *i* of the for loop generate statement depends on the input test which cannot be evaluated to a constant.

```
module adder8 (sum,c0,a,b,cin);
  input [7:0]a,b;
  output [7:0]sum;
  output [0:0] c0;
  input [0:0] cin;
  assign {c0,sum} = a + b + cin;
endmodule
```

```
module adder_32bit(a,b,sum,cin_in,c0_out,test);
    input [31:0]a,b;
    output [31:0]sum;
    input cin_in;
    input [2:0] test;
    output c0_out;
    wire [4:0]c0;
    assign c0[0] = cin_in;

    generate
        genvar i;
        for (i=0; i<=3; i = i + test)
        begin : u
            adder8 add (sum [8*i+7:8*i], c0[i+1], a [8*i+7:8*i],
                         b[8*i+7:8*i], c0[i]);
        end
    endgenerate
endmodule
```

Action

Make sure that the increment of the for loop generate statement always evaluates to a constant expression. To correct the error in the above test case, change the increment value in the for loop generate statement as shown in the corrected test case below.

```
module adder8 (sum,c0,a,b,cin);
    input [7:0]a,b;
    output [7:0]sum;
    output [0:0] c0;
    input [0:0] cin;
    assign {c0,sum} = a + b + cin;
endmodule

module adder_32bit(a,b,sum,cin_in,c0_out,test);
    input [31:0]a,b;
    output [31:0]sum;
    input cin_in;
    input [2:0] test;
    output c0_out;
    wire [4:0]c0;
    assign c0[0] = cin_in;
```

```
generate
  genvar i;
  for (i=0; i<=3; i = i + 1)
    begin : u
      adder8 add (sum [8*i+7:8*i], c0[i+1], a [8*i+7:8*i],
                   b[8*i+7:8*i], c0[i]);
    end
  endgenerate
endmodule
```

The genvar variable is an integer used as an index control within generate loops. Acceptable genvar values are positive numbers or 0. In a generate for loop, the index variable must be a genvar, both assignments in the for loop control must be assigned to the same genvar, and the contents must be within a named begin-end block.

CG222

@E: expecting attribute name

An *attribute-instance* can appear in a Verilog description as a module item, statement, or port connection or as a prefix attached to a declaration. It can also appear as a suffix to an operator or as a Verilog function name in an expression.

occurs in the following test case when an attribute name between two “*” operators in the assign statement is missing which causes the compiler to error out with the above message.

```
module att(a,b,c,d);
  input b,c,d;
  output a;
  assign a= b?(**)c:d;
endmodule
```

Action

Ensure that a valid attribute name is specified where required. In the test case below, the attribute name no_glitch is insert between the two “*” operators in the assign statement to correct the error.

```
module att(a,b,c,d);
  input b,c,d;
  output a;
  assign a= b?(*no_glitch*)c:d;
endmodule
```

CG234

@E: Duplicate definition of '<a>'

There are duplicate declarations for the same signal.

Verilog Example

In the following Verilog test case, port a is used as both a port and as an integer which causes the compiler to error out with the above message.

```
module test (a, b, clk, out);
  input a, b;
  input clk;
  output out;
  reg out;
  integer a;

  always @(posedge clk)
  begin
    out <= a & b;
  end

endmodule
```

Action

Use unique names for signals in the design. To eliminate the error in the above Verilog test case, either comment out the line **integer a;** or change the line to indicate a unique signal (depending on your design requirements) as shown in the corrected test case below.

```
module test (a, b, clk, out);
    input a, b;
    input clk;
    output out;
    reg out;
    integer c;

    always @(posedge clk)
    begin
        out <= a & b;
    end

endmodule
```

Duplicate definitions of signals, constants, or variables are found in the design.

VHDL Example

In the following VHDL test case, the signal 'a1' is declared twice which results in the above message.

```
library ieee;
use ieee.std_logic_1164.all;

entity adder is
    port (a, b, cin:in std_logic;
          sum, cout:out std_logic );
end adder;

architecture behave of adder is
signal a1:std_logic;
constant a1:std_logic:='0';
begin
    sum <= (a xor b xor cin);
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;
```

Action

Remove any duplicate definitions from your design. To eliminate this error in the above VHDL test case, comment out one of the 'a1' declarations.

```
library ieee;
use ieee.std_logic_1164.all;
```

```

entity adder is
    port ( a, b, cin:in std_logic;
           sum, cout:out std_logic );
end adder;

architecture behave of adder is
signal a1:std_logic;
--constant a1:std_logic:='0';
begin
    sum <= (a xor b xor cin);
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;

```

CG237

@E: Assignment target <> should be a genvar type

The compiler detects a for generate structure where the loop index is not defined as a genvar variable. The following test case shows how this error is generated. Index j of the for generate structure is defined as an integer and not as a genvar variable as required by the Verilog language.

```

module adder8 (sum, c0, a, b, cin);
    input [7:0]a, b;
    output [7:0]sum;
    output [0:0] c0;
    input [0:0] cin;
    assign {c0, sum} = a + b + cin;
endmodule

module adder_32bit(a, b, sum, cin_in, c0_out, test);
    input [31:0]a, b;
    output [31:0]sum;
    input cin_in;
    input [2:0] test;
    output c0_out;
    wire [4:0]c0;
    integer j;
    assign c0[0] = cin_in;
    generate
        for (j=0; j<=3; j =j + 1)

```

```
begin : u
adder8 add (sum[8*j+7:8*j], c0[j+1],
             a[8*j+7:8*j], b[8*j+7:8*j], c0[j]);
end
endgenerate
endmodule
```

Action

Define the index of the loop in the for generate statement as a genvar variable. For the above test case, you can eliminate the syntax error by making the changes shown in the corrected test case below.

```
module adder8 (sum, c0, a, b, cin);
  input [7:0]a, b;
  output [7:0]sum;
  output [0:0] c0;
  input [0:0] cin;
  assign {c0, sum} = a + b + cin;
endmodule

module adder_32bit(a, b, sum, cin_in, c0_out, test);
  input [31:0]a, b;
  output [31:0]sum;
  input cin_in;
  input [2:0] test;
  output c0_out;
  wire [4:0]c0;
  //integer j;
  genvar j;
  assign c0[0] = cin_in;
  generate
    for (j=0; j<=3; j = j + 1)
      begin : u
        adder8 add (sum[8*j+7:8*j], c0[j+1],
                    a[8*j+7:8*j], b[8*j+7:8*j], c0[j]);
      end
    endgenerate
  endmodule
```

The genvar variable is an integer used as an index control within generate loops. Acceptable genvar values are positive numbers or 0. In a generate for loop, the index variable must be a genvar, both assignments in the for loop control must be assigned to the same genvar, and the contents must be within a named begin-end block.

CG238

@E: integer type allowed for output ports only.

A port by default is a net. Only an output port can be redeclared as a reg register. occurs if the integer type (register type) is used for an input port.

In the following test case, an integer type declaration is associated with an input which causes the compiler to error out with the above message.

```
module test (input a, input integer b, output out);
  assign out = a & b;
endmodule
```

Action

This is a syntax error. To eliminate the error in the above test case, change the port type in the port list as shown in the corrected test case below.

```
module test (input a, input b, output out);
  assign out = a & b;
endmodule
```

Net types include wire, wor, wand, tri, trior, triand, tireg, tri1, tri0, supply0, and supply1. The Synopsys FPGA Verilog compiler currently does not support the tireg, tri1, and tri0 net types, and does not support the supply0 and supply1 net types. Register types include reg, integer, time, real, and realtime. When instantiating modules, make sure that all input ports are tied to one of the supported net types.

CG239

@E: time type allowed for output ports only.

A port by default is a net. However, it can be explicitly declared as a net. An output port can be optionally redeclared as a reg register. Supported net types include wire, wor, wand, tri, trior, and triand. Reg types include reg, integer, time, real, and realtime. occurs if the time type (register type) is used for an input port. In the following test case, an input is declared as type time which causes the compiler to error out with the above message.

```
module error (
    input[3:0]i,
    input clk,
    input time my_time,
    output[3:0] out2 );
    reg[3:0] temp;

    always@(posedge clk)
    begin
        temp = i;
    end

    assign out2 = temp;

endmodule
```

Action

This is a syntax error. Make sure that all time types are used with output or inout ports. To eliminate the error in the above test case, edit the code as shown in the corrected test case below.

```
module error (
    input[3:0]i,
    input clk,
    // input time my_time,
    output[3:0] out2 );
    reg[3:0] temp;

    always@(posedge clk)
    begin
        temp = i;
    end
```

```
assign out2 = temp;  
endmodule
```

CG240

@E: An input or inout port should be of type net

A port, by default, is a net. Only output ports can be optionally redeclared as registers. This error occurs when a register type is used with an input port. In the following test case, a `reg` declaration is associated with an input port.

```
module test (input a, input reg b, output out);  
  assign out = a & b;  
endmodule
```

Action

Change the declaration of input port `b` as shown in the corrected test case below.

```
module test (input a, input wire b, output out);  
  assign out = a & b;  
endmodule
```

Net types include `wire`, `wor`, `wand`, `tri`, `trior`, `triand`, `trireg`, `tri1`, `tri0`, `supply0`, and `supply1`. The Synopsys FPGA Verilog compiler currently does not support the `trireg`, `tri1`, and `tri0` net types, and does not support the `supply0` and `supply1` net types. Register types include `reg`, `integer`, `time`, `real`, and `realtime`. When instantiating modules, make sure that all input ports are tied to one of the supported net types.

CG244

@E: Expecting port direction input/output/inout

The direction of a port is not specified as either input, output, or inout in a module definition. In the test case below, the port direction is not specified for port [1:0] madd causing this error.

```
module counter (input clk,rst,input [1:0] radd,
    and [1:0] madd);
reg [1:0] madd;
always @(posedge clk or negedge rst)
begin
    if (!rst)
        madd<=0;
    else if (clk)
        madd<=radd+1;
    end
endmodule
```

Action

Make sure that a valid direction (input/output/inout) is specified for all ports in the port list of a module definition as shown in the corrected test case below.

```
module counter (input clk,rst,input [1:0] radd, output [1:0]
madd);
reg [1:0] madd;
always @(posedge clk or negedge rst)
begin
    if (!rst)
        madd<=0;
    else if (clk)
        madd<=radd+1;
    end
endmodule
```

CG245

@E: parameters of type real/realtimetime are not yet supported

The compiler encountered an unsupported parameter of type real, realtime, or time. In the test case below, parameter param1 is assigned a real type which causes the compiler to error out with the above message.

```
module multiplier (operand1, operand2, result,clk);
parameter real param1=3;
input operand1;
input operand2;
input clk;
output reg result;

always@(clk,operand1,operand2)
begin
    if (clk)
        result = (operand1 * operand2)*param1;
    else
        result = (operand1 * operand2)/param1;
end
endmodule
```

Action

Use only parameters of type integer as shown in the corrected test case below.

```
module multiplier (operand1, operand2, result,clk);
parameter param1=3;
input operand1;
input operand2;
input clk;
output reg result;

always@(clk,operand1,operand2)
begin
    if (clk)
        result = (operand1 * operand2)*param1;
    else
        result = (operand1 * operand2)/param1;
end
endmodule
```

CG246

@E: Expecting parameter name

A parameter name was omitted from a parameter definition. In the test case below, parameter value 4 does not have a corresponding parameter name.

```
module top#( parameter p1 = 2, 4)(  
    sel,d1,d2,d3,d4,q );  
input wire[p1:1] sel;  
input wire [7:0]d1,d2,d3,d4;  
output reg [7:0] q;  
  
always@* begin  
    case(sel)  
        0: q = d1;  
        1: q = d2;  
        2: q = d3;  
        3: q = d4;  
        default : q = 8'hFF;  
    endcase  
end  
endmodule
```

Action

Make sure all parameters have both a name and value.

```
module top#( parameter p1 = 2, p2 = 4)(  
    sel,d1,d2,d3,d4,q );  
input wire[p1:1] sel;  
input wire [7:0]d1,d2,d3,d4;  
output reg [7:0] q;  
  
always@* begin  
    case(sel)  
        0: q = d1;  
        1: q = d2;  
        2: q = d3;  
        3: q = d4;  
        default : q = 8'hFF;  
    endcase  
end  
endmodule
```

CG247

@E: Redeclaration of <param1>

An identifier is declared as a parameter more than once. In the test case below, identifier param1 is declared twice as a parameter which causes the compiler to error out with the above message.

```
module multiplier (operand1, operand2, result, clk);
parameter param1=3;
parameter param1=5;
input operand1;
input operand2;
input clk;
output reg result;

always@(clk,operand1,operand2)
begin
    if (clk)
        result = (operand1 * operand2)*param1;
    else
        result = (operand1 * operand2)/param1;
end
endmodule
```

Action

Make sure that there are no duplicate or redeclared parameters in a module. The error in the above test case remove one of the param1 parameter declarations as shown in the corrected test case below.

```
module multiplier (operand1, operand2, result, clk);
parameter param1=3;
parameter param1=5;
input operand1;
input operand2;
input clk;
output reg result;
```

```
always@(clk,operand1,operand2)
begin
    if (clk)
        result = (operand1 * operand2)*param1;
    else
        result = (operand1 * operand2)/param1;
end
endmodule
```

CG248

@E: Verilog-2001 style parameters found in Verilog-95 mode.

Verilog-2001 has added support for declaring parameters before the module port list. Currently, Verilog-2001 can be used in the compiler by turning on the Verilog-2001 compiler feature in the Implementation Options of the Project view's user interface or by enabling the `set_option -vlog_std v2001` option. If such a parameter declaration is used without the `v2001` switch enabled, the compiler errors out with the above message as the following test case illustrates.

```
module test #(parameter c =10, d=10)
(input a, b,
output out);
assign out = a+b+c+d;
endmodule
```

Project file does NOT have the following option set.

```
set_option -vlog_std v2001
```

Action

Make sure that when using Verilog 2001 features either of these is set:

- Verilog-2001 checkbox in Implementation option should be On before using Verilog-2001 constructs.
- Project file should have the following option set.
 - `set_option -vlog_std v2001`

CG250

@E: Ranges are not allowed for this type

Integer types cannot be accessed as bit vectors. In the following test case, integer temp is declared as a 3-bit value which causes the compiler to error out with the above message.

```
module test (a, b, out);
    input a, b;
    output out;
    integer [2:0] temp;
    wire #temp out = a & b;
endmodule
```

Action

This syntax error can be corrected by removing the bit range for the integer type as shown in the corrected test case below.

```
module test (a, b, out);
    input a, b;
    output out;
    integer temp;
    wire #temp out = a & b;
endmodule
```

If you need to extract a bit value from an integer, assign the integer to a reg register and then select the bits from the register as shown below.

```
Reg [31:0] breg;
Integer bint;
.
.
.
Breg = Bint;

/* At this point breg[4], breg[8] are allowed and are the same as
the bint[4] and bint[8] values */
```

CG252

@E: Expecting parameter name

The compiler errors out with the above message if a parameter name is not specified in the parameter declaration. In the following test case, a parameter name is not specified which causes the compiler to error out with the above message.

```
module test (a, b, clk, out);
    input a, b;
    input clk;
    output [9:0] out;
    reg [9:0] out;
    integer i;
    parameter = 10;

    always @(posedge clk)
    begin
        for (i=0;i<temp;i=i+1)
            out[i] <= a & b;
    end

endmodule
```

Action

To correct this syntax error, specify the parameter name in the parameter declaration as shown in the corrected test case below.

```
module test (a, b, clk, out);
    input a, b;
    input clk;
    output [9:0] out;
    reg [9:0] out;
    integer i;
    parameter temp = 10;

    always @(posedge clk)
    begin
        for (i=0;i<temp;i=i+1)
            out[i] <= a & b;
    end

endmodule
```

CG253

@E: <temp> is already declared as a different type.

A signal is redeclared within a module using a different type. In the test case below, the signal temp is first declared as type reg and then again as type wire which causes the compiler to error out with the above message.

```
module test1 (
    input[3:0]i,
    input clk,
    output[3:0] out2 );
    reg[3:0] temp;
    wire[3:0] temp;

    always@(posedge clk)
    begin
        temp = i;
    end

    assign out2 = temp;
endmodule
```

Action

Remove any redundant signal declarations from within the module as shown in the corrected test case below.

```
module test1 (
    input[3:0]i,
    input clk,
    output[3:0] out2 );
    reg[3:0] temp;
    // wire[3:0] temp;

    always@(posedge clk)
    begin
        temp = i;
    end

    assign out2 = temp;
endmodule
```

CG254

@E: Redeclaration of ports. Cannot mix styles.

The compiler finds both Verilog 2001 and Verilog 95 syntax in the same port list. In the module definition below, out is declared in the port list using the Verilog 2001 syntax and also inside the body using the Verilog 95 syntax.

```
module counter2(output [7:0] out, cout, data, load, cin, clk);
    output [7:0] out;
    output cout;
    input [7:0] data;
    input load, cin, clk;
    reg [0:7] out;
    reg cout;
    reg [7:0] preout;

    // create the 8-bit register
    always @(posedge clk)
    begin
        out = preout;
    end

endmodule
```

Action

This is a syntax error. Make sure the two styles are not used together in the same port list. To correct the error in the above test case, change the lines:

```
module counter2(output [7:0] out, cout, data, load, cin, clk);
    output [7:0] out;
    output cout;
    input [7:0] data;
    input load, cin, clk;
```

Correction with Verilog 2001 Syntax

```
module counter2(output [7:0] out, output cout, input [7:0] data,
    input load, cin, clk);
```

Correction with Verilog 1995 Syntax

```
module counter2(out, cout, data, load, cin, clk);
    output [7:0] out;
    output cout;
    input [7:0] data;
    input load, cin, clk;
```

CG255

@E: Type realtime not supported

An input or output port type is declared as `realtime` (type `realtime` is used only for simulation to represent time in real value and is not applicable during synthesis). In the test case below, the type of input port `clk` is declared as `realtime` which causes the compiler to error out with the above message.

```
module multiplier (operand1, operand2, result,clk);
parameter param1=3;
input operand1;
input operand2;

output reg result;

always@(clk,operand1,operand2)
begin
    if (clk)
        result = (operand1 * operand2)*param1;
    else
        result = (operand1 * operand2)/param1;
end
endmodule
```

Action

Make sure that no input or output is declared as type `realtime` as shown in the corrected test case below.

```

module multiplier (operand1, operand2, result,clk);
parameter param1=3;
input operand1;
input operand2;


```

CG257

@E: Expected argument name

The keyword input is not followed by an identifier in a function definition. In the test case below, the definition for function parity does not include an identifier after the keyword input which causes the compiler to error out with the above message.

```

module test_function (a, c);
input [31:0] a;
output c;
assign c = parity(a);

function parity;


```

```
endmodule
```

Action

Provide an argument name after the keyword input in the function definition. In the corrected test case below, the argument [0:31] set is specified after the keyword input for the function definition parity.

```
module test_function (a, c);
    input [31:0] a;
    output c;
    assign c = parity(a);

    function parity;
        input [0:31] set;
        reg [0:3] ret;
        integer j;
        begin
            ret = 0;
            for (j= 0; j <= 31; j= j + 1 )
                if (set [j] == 1)
                    ret = ret + 1;
            parity = ret % 2;
        end
    endfunction

endmodule
```

CG258

@E: event declaration not allowed in function

An event type declaration was found within a function (time-controlled statements are not allowed within functions). In the test case below, identifier set is declared as a named event within the function parity which causes the compiler to error out with the above message.

```
module parity_top (a, c);
    input [31:0] a;
    output c;
    assign c = parity(a);
```

```
function parity (input [0:31] set);
    event set;
    reg [0:3] ret;
    integer j;
    begin
        ret = 0;
        for (j= 0; j <= 31; j= j + 1 )
            if (set [j] == 1)
                ret = ret + 1;
        parity = ret % 2;
    end
endfunction

endmodule
```

Action

Make sure that any named event is not declared inside a function definition as shown in the corrected test case given below.

```
module parity_top (a, c);
    input [31:0] a;
    output c;
    assign c = parity(a);

    function parity ( input [0:31] set);
        reg [0:3] ret;
        integer j;
        begin
            ret = 0;
            for (j= 0; j <= 31; j= j + 1 )
                if (set [j] == 1)
                    ret = ret + 1;
            parity = ret % 2;
        end
    endfunction

endmodule
```

CG259

@E: output port illegal for function.

A function definition has an argument declared as an output port (an argument of a function definition can only be declared as an input port). In the test case below, the definition of function parity includes the argument [0:31] set declared as an output port which causes the compiler to error out with the above message.

```
module parity_top (a,c);
    input [31:0] a;
    output c;
    assign c = parity(a);

    function parity (output [0:31] set);
        reg [0:3];
        integer j;
        begin
            ret = 0;
            for (j= 0; j <= 31; j= j + 1 )
                if (set [j] == 1)
                    ret = ret + 1;
            parity = ret % 2;
        end
    endfunction

endmodule
```

Action

Make sure that no argument of the function is declared as an output port. To eliminate the error in the above test case, remove or change the output mode of argument [0:31] set to input as shown in the corrected test case below.

```
module parity_top (a,c);
    input [31:0] a;
    output c;
    assign c = parity(a);

    function parity (input [0:31] set);
        reg [0:3];
        integer j;
        begin
            ret = 0;
```

```
        for (j= 0; j <= 31; j= j + 1 )
            if (set [j] == 1)
                ret = ret + 1;
                parity = ret % 2;
        end
endfunction

endmodule
```

CG260

@E: inout port illegal for function.

A function definition has an argument declared as an inout port (an argument of a function definition can only be declared as an input port). In the test case below, the definition of function parity includes the argument [0:31] set declared as an inout port which causes the compiler to error out with the above message.

```
module parity_top (a, c);
    input [31:0] a;
    output c;
    assign c = parity(a);

    function parity (inout [0:31] set);
        reg [0:3];
        integer j;
        begin
            ret = 0;
            for (j= 0; j <= 31; j= j + 1 )
                if (set [j] == 1)
                    ret = ret + 1;
                    parity = ret % 2;
        end
    endfunction

endmodule
```

Action

Make sure that no argument of the function is declared as an inout port. To eliminate the error in the above test case, remove or change the inout mode of argument [0:31] set to input as shown in the corrected test case below.

```
module parity_top (a, c);
    input [31:0] a;
    output c;
    assign c = parity(a);

    function parity (input [0:31] set);
        reg [0:3];
        integer j;
        begin
            ret = 0;
            for (j= 0; j <= 31; j= j + 1 )
                if (set [j] == 1)
                    ret = ret + 1;
            parity = ret % 2;
        end
    endfunction

endmodule
```

CG261

@E: Expecting port direction

The direction of a port is not specified for any of the arguments in the function definition. In the test case below, the port direction of argument [0:31] set for function parity is not specified which causes the compiler to error out with the above message.

```
module parity_top (a, c);
    input [31:0] a;
    output c;
    assign c = parity(a);
```

```
function parity ([0:31] set);
reg [0:3] ret;
integer j;
begin
ret = 0;
for (j= 0; j <= 31; j= j + 1 )
if (set [j] == 1)
ret = ret + 1;
parity = ret % 2;
end
endfunction

endmodule
```

Action

Make sure that the port direction is specified for all arguments in a function definition. To eliminate the error in the above test case, specify the port for argument [0:31] set as input as shown in the corrected test case below.

```
module parity_top (a, c);
input [31:0] a;
output c;
assign c = parity(a);

function parity (input [0:31] set);
reg [0:3] ret;
integer j;
begin
ret = 0;
for (j= 0; j <= 31; j= j + 1 )
if (set [j] == 1)
ret = ret + 1;
parity = ret % 2;
end
endfunction

endmodule
```

CG262

@E: automatic functions are not yet supported.

The compiler encountered a function declared as an automatic function. The keyword automatic in a function definition declares a recursive function (recursive functions currently are not supported by the compiler).

In the following test case, function fnctn is declared as an automatic function which causes the compiler to error out with the above message.

```
module top (a, c);
    input a;
    output c;
    assign c = fnctn(a);

    function automatic fnctn;
        input a;
        fnctn = fnctn(a);
    endfunction

endmodule
```

Action

Do not use automatic functions. Whenever RTL describes logic without definite bounds such as recursive functions where all function declarations are allocated dynamically for each recursive call, the compiler cannot create hardware to emulate that behavior.

CG263

@E: functions returning realtime are not supported.

The return type of a function is declared as realtime (type realtime is used for simulation to represent time in real value and is not applicable to synthesis). In the test case below, the return type of function parity is declared as realtime which causes the compiler to error out with the above message

```
module top_parity (a, c);
    input [31:0] a;
    output c;
    assign c = parity(a);

    function realtime parity;
    input [0:31] set;
    reg [0:3] ret;
    integer j;
    begin
        ret = 0;
        for (j= 0; j <= 31; j= j + 1 )
            if (set [j] == 1)
                ret = ret + 1;
                parity = ret % 2;
    end
endfunction

endmodule
```

Action

Do not use functions with return type `realtime` as shown in the corrected test case below.

```
module top_parity (a, c);
    input [31:0] a;
    output c;
    assign c = parity(a);

    function parity;
    input [0:31] set;
    reg [0:3] ret;
    integer j;
    begin
        ret = 0;
        for (j= 0; j <= 31; j= j + 1 )
            if (set [j] == 1)
                ret = ret + 1;
                parity = ret % 2;
    end
endfunction

endmodule
```

CG264

@E: Expecting function name

Verilog requires that an identifier or a function name follow the keyword `function`. This name must be unique and cannot be one of the reserved words. In the following test case, the function name is a reserved keyword which causes the compiler to error out with the above message.

```
module error(input[3:0]i, j,
              input clk,
              output reg[3:0] out,
              output[3:0] out2 );
  reg[3:0] temp;
  function [3:0] input; // input is a reserved word
  input[1:0] a, b;
  begin
    mycat = {a, b};
  end
  endfunction

  always@(posedge clk)
  begin
    temp = j;
    out = mycat(i[1:0], i[3:2]);
  end

  assign out2 = temp;
endmodule
```

Action

To eliminate the error in the above test case, use a unique function name as shown in the corrected test case below.

```
module my_top(input[3:0]i, j,
              input clk,
              output reg[3:0] out,
              output[3:0] out2 );
  reg[3:0] temp;
  function [3:0] mycat;
```

```

input[1:0] a, b;
begin
    mycat = {a, b};
end
endfunction

always@(posedge clk)
begin
    temp = j;
    out = mycat(i[1:0], i[3:2]);
end

assign out2 = temp;
endmodule

```

CG265

@E: function name collides with variable name

Functions in Verilog are identified by a unique function name. This name cannot be reused to define a variable or signal within the module. In the following test case, the name of a signal in the design is the same as the function name which causes the compiler to error out with the above message.

```

module error(input[3:0]i, j,
             input clk,
             output reg[3:0] out,
             output[3:0] out2 );
reg[3:0] temp;
wire[3:0] mycat;
function [3:0] mycat;
input[1:0] a, b;
begin
    mycat = {a, b};
end
endfunction

always@(posedge clk)
begin
    temp = j;
    out = mycat(i[1:0], i[3:2]);
end

```

```
assign out2 = temp;
endmodule
```

Action

Assign unique names to functions and variables or signals in the design as shown in the corrected test case below.

```
module error(input[3:0]i, j,
              input clk,
              output reg[3:0] out,
              output[3:0] out2 );
  reg[3:0] temp;
  wire[3:0] temp2;
  function [3:0] mycat;
    input[1:0] a, b;
  begin
    mycat = {a, b};
  end
  endfunction

  always@(posedge clk)
  begin
    temp = j;
    out = mycat(i[1:0], i[3:2]);
  end

  assign out2 = temp;
endmodule
```

CG266

@E: automatic tasks are not yet supported.

The compiler encountered a task that is defined as an automatic task (automatic tasks currently are not supported by the compiler).

The keyword automatic declares an automatic task. In the following test case, task tsk is declared as an automatic task which causes the compiler to error out with the above message.

```
module top (clk,a,b, c,clk1,a1,b1,c1);
input a,b,a1,b1,clk,clk1;
output c,c1;
reg c,c1;

task automatic tsk;
input a,b;
output c;
c= a ^ b;
endtask

always@(posedge clk)
begin
    tsk(a,b,c);
end

always@(posedge clk1)
begin
    tsk(a1,b1,c1);
end

endmodule
```

Action

Automatic tasks are reentrant with all the task declarations allocated dynamically for each concurrent task entry and are intended to be used when there is a chance that the task might be called concurrently from two locations in the code. Because the compiler cannot create hardware to emulate this behavior, the error is eliminated by not declaring a task as automatic as shown in the corrected test case below.

```
module top (clk,a,b, c,clk1,a1,b1,c1);
input a,b,a1,b1,clk,clk1;
output c,c1;
reg c,c1;

task tsk;
input a,b;
output c;
c= a ^ b;
endtask

always@(posedge clk)
begin
    tsk(a,b,c);
end
```

```
always@(posedge clk1)
begin
    tsk(a1,b1,c1);
end

endmodule
```

CG267

@E: Expecting task name

Verilog requires that a task identifier or a task name follow the keyword task. This name must be a unique name that is not one of the reserved words. In the following test case, a reserved word is used for the task name which causes the compiler to error out with the above message.

```
module error (
    input[3:0]i, j,
    input clk,
    output reg out3,
    output[3:0] out2 );
    reg[3:0] temp;

    always@(posedge clk)
    begin
        temp = j;
        or_task(i[1],i[2],out3);
    end

    assign out2 = temp;
    task or; // or is a reserved word in Verilog
    input a, b;
    output c;
    begin
        c = a | b;
    end
    endtask
endmodule
```

Action

Make sure that the task name is unique as shown in the corrected test case below.

```
module error (
    input[3:0]i, j,
    input clk,
    output reg out3,
    output[3:0] out2 );
    reg[3:0] temp;

    always@(posedge clk)
    begin
        temp = j;
        or_task(i[1],i[2],out3);
    end

    assign out2 = temp;
    task or_task;
        input a, b;
        output c;
        begin
            c = a | b;
        end
    endtask
endmodule
```

CG268

@E: Redeclaration of task or function

A task or function is defined more than once within the same module. In the test case below, the task `or_task` is defined initially at the beginning of the module and then again near the end of the module which causes the compiler to error out with the above message.

```
module test1(input[3:0]i, j,
             input clk,
             output reg out3,
             output[3:0] out2 );
    reg[3:0] temp;
```

```
task or_task ;
  input a, b;
  output c;
  begin
    c = a | b;
  end
endtask

always@(posedge clk)
begin
  temp = j;
  or_task(i[1],i[2],out3);
end

assign out2 = temp;

task or_task;
  input a, b;
  output c;
  begin
    c = a & b;
  end
endtask

endmodule
```

Action

Make sure that there are no duplicate or redeclared definitions of tasks or functions in the module as shown in the corrected test case below.

```
module test1(input[3:0]i, j,
             input clk,
             output reg out3,
             output[3:0] out2 );
  reg[3:0] temp;

  always@(posedge clk)
  begin
    temp = j;
    or_task(i[1],i[2],out3);
  end

  assign out2 = temp;
```

```
task or_task;
  input a, b;
  output c;
begin
  c = a | b;
end
endtask

endmodule
```

CG269

@E: Could not locate legal disable target <*my_m1*>

The disable statement in Verilog can be used to terminate a named block or task. occurs if the target name specified after the disable statement is invalid. In the following test case, the instance name on the disable statement is incorrect which causes the compiler to error out with the above message.

```
module m1 (
  output c,
  input a, b );
  assign c = a & b;
endmodule

module error (
  input[3:0] i, j,
  input clk,
  output reg out3,
  output[3:0] out, out2 );
  reg[3:0] temp;

  always@(posedge clk)
  begin : block1
    disable my_m1;
    temp = j;
  end

  assign out2 = temp;
  m1 my_m1 (.c(out[3]), .a(i[1]), .b(i[0]));
endmodule
```

Action

Make sure that the target is a block or task name as shown in the corrected test case below.

```
module m1 (
  output c,
  input a, b );
  assign c = a & b;
endmodule

module error (
  input[3:0] i, j,
  input clk,
  output reg out3,
  output[3:0] out, out2 );
  reg[3:0] temp;

  always@(posedge clk)
  begin : block1
    disable block1;
    temp = j;
  end

  assign out2 = temp;
  m1 my_m1 (.c(out[3]), .a(i[1]), .b(i[0]));
endmodule
```

CG271

@E: Functions are not legal disable targets

The disable statement in Verilog can be used to terminate a named block or task. In this case, the target name specified after the disable statement is the name of a function.

In the following test case, the instance name on the disable statement (`parity`) is the name of the function which causes the compiler to error out with the above message.

```
module top_parity (a,c);
    input [31:0] a;
    output c;
    assign c = parity(a);

    function parity;
        input [0:31] set;
        reg [0:3] ret;
        integer j;
        begin
            ret = 0;
            for (j= 0; j <= 31; j= j + 1 )
                if (set [j] == 1)
                    ret = ret + 1;
                parity = ret % 2;
        disable parity;
        end
    endfunction

endmodule
```

Action

Ensure that the target of the disable statement is a block or task name as shown in the corrected test case below.

```
module top_parity (a,c);
    input [31:0] a;
    output c;
    assign c = parity(a);

    function parity;
        input [0:31] set;
        reg [0:3] ret;
        integer j;
        begin:dummy
            ret = 0;
            for (j= 0; j <= 31; j= j + 1 )
                if (set [j] == 1)
                    ret = ret + 1;
                parity = ret % 2;
            if ( j==31)
                disable dummy;
        end
    endfunction

endmodule
```

CG272

@W: syn_keep not allowed on inout ports

The `syn_keep` synthesis directive was used on an inout port. This synthesis directive is ignored if placed on the inout port of a design.

The compiler and mapper optimize out or removes logic that is unnecessary or redundant. The `syn_keep` directive preserves nets from being optimized away. This directive also preserves registers and instantiated components. The following test case uses `syn_keep` on ports.

```
module encoder1 (none_on, out, in);
    output none_on;
    inout [2:0] out /* synthesis syn_keep = 1 */;
    input [7:0] in;
    reg [2:0] out1;
    reg none_on;

    always @(in)
    begin: local
        integer i;
        out1 = 0;
        none_on = 1;
        /* returns the value of the highest bit number turned on */
        for (i = 0; i < 8; i = i +1)
        begin
            if (in[i]) begin
                out1 = i;
                none_on = 0;
            end
        end
    end
    assign out = out1;
endmodule
```

Action

Since `out` is an inout port, `syn_keep` does not add any value since the port is maintained at the top level. If a port in the lower level modules needs preserved, use `syn_hier`. Use `syn_keep` to keep nets from getting merged or removed during synthesis due to optimizations done by both the compiler and the mapper.

In the following test case, the two nets (keep1 and keep2) would merge if syn_keep was not used since they are being driven by an AND gate (in1 & in2). The example shows the proper usage of the syn_keep directive.

```
module example2 (out1, out2, clk, in1, in2);
output out1, out2;
input clk;input in1, in2;
wire and_out;wire keep1 /* synthesis syn_keep=1 */;
wire keep2 /* synthesis syn_keep=1 */;
reg out1, out2;
assign and_out=in1&in2;
assign keep1=and_out;
assign keep2=and_out;

always @(posedge clk)begin;
out1<=keep1;
out2<=keep2;
end
endmodule
```

CG275

@W: Duplicate module name <test> (using last description encountered)!

The Allow Duplicate Modules option is enabled and modules with the same name were encountered. Duplicate modules are normally not allowed in a project. You can have duplicate modules when you enable the option on the Verilog tab of the Options for implementation dialog box. In the test case below, there are two modules with the same name (test) which results in this warning when the Allow Duplicate Modules option (set_option -dup 1) is enabled.

```
/* first module test- synchronous reset flip-flop*/
module test (d,clk,rst,q);
input d,clk,rst;
output reg q;
```

```
always@(posedge clk)
  if(rst)
    q <= 1'b0;
  else
    q <= d;
endmodule

/*second module test- asynchronous reset flip-flop */
module test (d,clk,rst,q);
  input d,clk,rst;
  output reg q;

  always@(posedge clk or posedge rst)
    if(rst)
      q <= 1'b0;
    else
      q <= d;
  endmodule

module top (d,clk,rst,q);
  input d,clk,rst;
  output q;
  test u1 (d,clk,rst,q);
endmodule
```

Action

The intent of the warning is to inform you that in the case of duplicate modules, the last module definition encountered will be used in all instances. Accordingly, make sure that the module to be used is referenced last. In the above test case, module top uses the module definition of the second test module (asynchronous reset flip-flop). To use the first test module definition (synchronous reset flip-flop), change the order of the module definitions as shown in the example below.

```
/*second module test- asynchronous reset flip-flop */
module test (d,clk,rst,q);
  input d,clk,rst;
  output reg q;

  always@(posedge clk or posedge rst)
    if(rst)
      q <= 1'b0;
    else
      q <= d;
  endmodule
```

```
/* first module test- synchronous reset flip-flop*/
module test (d,clk,rst,q);
input d,clk,rst;
output reg q;

always@(posedge clk)
  if(rst)
    q <= 1'b0;
  else
    q <= d;
endmodule

module top (d,clk,rst,q);
input d,clk,rst;
output q;
test u1 (d,clk,rst,q);
endmodule
```

CG276

@E: Could not find variable <*sig1.nu1.nu*> - possible illegal field access of struct

An attempt was made to access the member of a non-existent struct. In the test case below, variable *sig1.nu1.nu* is not correctly defined by the struct data type which results in the error.

```
typedef struct packed {
  struct packed {
    struct packed {
      logic[4:1][2:1] u1;
    }nul;
    byte u2;
  }nul;
}dt;

module sub(
  input byte d1,
  output byte dout1 );
dt sig1;
assign sig1.nu1.nu.u1 = d1;
assign dout1 = sig1.nu1.nul.ul;
endmodule
```

Action

Make sure that the definition and assignment match as shown in the corrected test case below.

```
typedef struct packed {
    struct packed {
        struct packed {
            logic[4:1][2:1] u1;
        }nul;
        byte u2;
    }nul;
}dt;

module sub(
    input byte d1,
    output byte dout1 );
dt sig1;
assign sig1.nul.nul.ul = d1;
assign dout1 = sig1.nul.nul.ul;
endmodule
```

CG277

@E: Expecting event or signal name <name>

A signal name or an event expression does not follow the @ symbol in an always block. In the test case below, an underscore character follows the @ symbol in the always block which causes the compiler to error out with the above message.

```
module seq (input clk, en, input [3:0] addr, raddr, input [7:0]
data,
    output [7:0] q, q2 );
reg [7:0]mem[15:0];
parameter test_addr = 15;

always@_(posedge clk)
    if (en)
        mem[addr] <= data;
        assign q = mem[raddr];
        assign q2 = mem[test_addr];
endmodule
```

Action

This syntax error can be avoided by making sure that an event expression or signal name always follows the @ symbol in an always block. In the test case below, removing the underscore before the posedge clk event expression eliminates the error.

```
module seq (input clk, en, input [3:0] addr, raddr, input [7:0]
data,
    output [7:0] q, q2 );
reg [7:0]mem[15:0];
parameter test_addr = 15;

always@(posedge clk)
if (en)
    mem[addr] <= data;
    assign q = mem[raddr];
    assign q2 = mem[test_addr];
endmodule
```

CG278

@E: Expecting event name

The target identifier specified after the event triggering identifier (->) is not a valid named event. In the test case below, there is no identifier specified after the statement if(clk)-> which causes the compiler to error out with the above message.

```
module dff(clk,r_data,q);
input clk;
input r_data;
output reg q;
event r_data1;

always@(posedge(clk))
begin
    if(clk)->;
        q=r_data;
    end
endmodule
```

Action

Provide a declared named event after the event triggering identifier (->) in the event triggering statement as shown in the test case below.

```
module dff(clk,r_data,q);
    input clk;
    input r_data;
    output reg q;
    event r_data1;

    always@(posedge clk)
        begin
            if(clk)->r_data1;
            q=r_data;
        end
endmodule
```

Named event control and triggering currently is not supported by the synthesis tool.

CG280

@E: Expecting name of block or task

A Verilog construct such as a disable statement is not followed by a valid task or block name. In the test case below, the disable statement within the task is not followed by the task name which causes the compiler to error out with the above message.

```
module test (input[3:0]i, j,
             input clk,
             output reg out3,
             output[3:0] out2 );
    reg[3:0] temp;

    always@(posedge clk)
    begin
        temp = j;
        or_task(i[1], i[2], out3);
    end

    assign out2 = temp;
```

```
task or_task;
  input a, b;
  output c;
begin
  disable ;
  c = a | b;
end
endtask

endmodule
```

Action

Ensure that the target block or task name is specified as shown in the corrected test case below.

```
module test (input[3:0]i, j,
             input clk,
             output reg out3,
             output[3:0] out2 );
  reg[3:0] temp;

  always@(posedge clk)
  begin
    temp = j;
    or_task(i[1], i[2], out3);
  end

  assign out2 = temp;

  task or_task;
    input a, b;
    output c;
  begin
    disable or_task;
    c = a | b;
  end
endtask

endmodule
```

CG283

@E: Procedural assign is not a supported statement

The compiler encountered a procedural assign statement. This is not supported by the compiler since hardware cannot be created to completely match the RTL.

In the following test case, Pow is assigned using a procedural assign statement. In simulation, Pow is assigned in the first assign statement. When the second assign statement occurs, it first deassigns the assignment to Pow and assigns the second assignment to Pow depending on the value of out.

However, during the interval between these two assign statements, Pow is still sensitive to a and b and the first assign statement is evaluated until the second assign statement is executed. No appropriate hardware can be created to represent this.

```
module decoder(out, in, a, b, c, Pow);
    output [7:0] out;
    input [2:0] in, a, b;
    output [2:0] Pow;
    reg [2:0] Pow;
    assign out = 1'b1 ^ in;

    always @(a,b,c )
    begin
        assign Pow = a & b;
        .....
        if (out < 4 )
        .....
        assign Pow = c;
    end
endmodule
```

Action

Assign statements are usually used in test benches to create stimulus for the design under test. These statements are not supported in the compiler.

CG284

@E: Procedural deassign is not a supported statement

The compiler encountered a procedural deassign statement. This is not supported by the compiler since hardware cannot be created to completely match the RTL.

In the following test case, Pow is assigned using a procedural assign statement. In simulation, Pow is assigned in the first assign statement. When out is less than 4, Pow is deassigned (a and b have no effect on Pow). Pow is then assigned the value c. No appropriate hardware can be created to represent this logic.

```
module decoder(out, in, a, b, c, Pow);
output [7:0] out;
input [2:0] in, a, b;
output [2:0] Pow;
reg [2:0] Pow;
assign out = 1'b1 ^ in;

always @(a,b,c )
begin
assign Pow = a & b;
.....
if (out < 4 )
deassign Pow;
end
.....
assign Pow =c;
endmodule
```

Action

Deassign statements are usually used in test benches to create stimulus for the design under test. These statements are not supported in the compiler.

CG285

@E: Expecting statement

The Verilog compiler reports when it fails to identify a valid statement within a procedural block. An incorrect or illegal statement, such as the continuous assignment statement shown in the test case below, results in message.

```
module error(input[3:0] i, j,
              input clk,
              output[3:0] out, out2 );
  reg[3:0] temp;

  always@(posedge clk)
  begin
    temp := j; // incorrect assignment syntax
  end

  assign out2 = temp;
  and and_inst1 (out[0], i[3], j[0]);
endmodule
```

Action

Correct the procedural assignment statement as shown in the corrected test case below.

```
module error(input[3:0] i, j,
              input clk,
              output[3:0] out, out2 );
  reg[3:0] temp;

  always@(posedge clk)
  begin
    temp = j;
  end

  assign out2 = temp;
  and and_inst1 (out[0], i[3], j[0]);
endmodule
```

CG286

@W: Case statement has both a full_case directive and a default clause -- ignoring full_case directive.

The case statement in the Verilog code has a `full_case` directive next to it as well as a default case specified. The `full_case` directive is a synthesis directive informing the compiler not to create logic for cases specified in the case statement (i.e., all the unspecified case choices are ignored). If the default case is specified within the case statement with an associated `full_case` directive, then logic for all of the unspecified cases is implemented, and the `full_case` directive is ignored. In the following case, the warning occurs.

```
module prep1(Q, CLK, RST, S_L, S1, S0, d0, d1, d2, d3);
    output [7:0] Q;
    input CLK, RST, S_L;
    input S1, S0;
    input [7:0] d0, d1, d2, d3;
    reg [7:0] Y, q_reg, Q; // q_reg is output from 8-bit register

    always @(S1 or S0 or d0 or d1 or d2 or d3)
    begin
        case ({S1, S0}) // synthesis full_case
            2'b00: Y = d0;
            2'b01: Y = d1;
            2'b10: Y = d2;
            default : Y = d3;
        endcase
    end

    always @(posedge CLK or posedge RST)
    begin
        if (RST) begin
            q_reg = 0; // reset register
            Q = 0; // reset shift register
        end else if (S_L) begin
            Q[7:0] = {Q[6:0],Q[7]}; // shift register
            q_reg = Y;
        end else begin
            Q = q_reg; // load from register
            q_reg = Y; // load from mux
        end
    end
endmodule
```

Action

Make sure that if the case specified is complete and if no logic for the unspecified cases must be implemented, that the `full_case` directive is specified against the case statement. If full case is to be implemented, make sure that the default case is not specified in the case statement since it overrides the `full_case` directive. In the above test case, the logic for case branch 2'b11 is implemented because the `full_case` directive is overridden by the default clause. If this needs to be prevented, comment out the line

```
default : Y = d3;
```

to implement the full case logic. Note that `full_case` and `parallel_case` directives are Verilog only.

Note that if an FSM is extracted from the case statement (which results in a state-machine primitive in the RTL view), the default case is not implemented.

CG288

@E: Expecting delimiter: , or ;

A semicolon delimiter or the comma separating two identifiers or arguments is missing in the Verilog code.

In the following test case, the semicolon is missing in the assignment to `c` which causes the compiler to error out with the above message.

```
module mux(a,b,c);
  input a,b;
  output c;
  assign c= a?b:0
endmodule
```

Action

Place delimiters as defined by the language. To eliminate the error in the above test case, add a semicolon at the end of the assignment statement to `c` as shown in the corrected test case below.

```
module mux (a,b,c);
  input a,b;
  output c;
  assign c= a?b:0;
endmodule
```

CG289

@W: Specified digits overflow the number's size

The number of bits exceeds the value specified for the size of the number.

In the test case below, the sized number is given as 1'b11 where the size is 1 but the number of bits is two which causes the compiler to issue the above warning.

```
module comb (addr,op);
  input [1:0]addr;
  output reg op;
  always@(addr)
    case(addr)
      2'b00:op=1'b11;
      2'b01:op=1'b0;
      2'b10:op=1'b0;
      2'b11:op=1'b0;
    endcase
endmodule
```

Action

Make sure that the number of bits is the same as the value of the specified size. To eliminate the warning in the above test case, change the assigned value to 1'b1 as shown in the corrected test case below.

```
module comb (addr,op);
  input [1:0]addr;
  output reg op;
  always@(addr)
    case(addr)
      2'b00:op=1'b1;
```

```
2'b01:op=1'b0;
2'b10:op=1'b0;
2'b11:op=1'b0;
endcase
endmodule
```

CG290

@W: Referenced variable <a> is not in sensitivity list

This warning occurs when the sensitivity list in the process does not include all signals that are referenced within the process. This warning occurs in conjunction with the message:

```
@W: Incomplete sensitivity list - assuming completeness
```

The reference variable warning above lists all the signals required in the sensitivity list. The compiler assumes completeness; the compiler infers the logic as if all the referenced signals are in the sensitivity list. The test case below returns the above warning because signals a and b are referenced within the process, but b is not listed in the sensitivity list.

```
library ieee;
use ieee.std_logic_1164.all;

entity decoder is
    port (inp: in std_logic_vector(2 downto 0);
          ins: out std_logic_vector (7 downto 0);
          a, b: in std_logic_vector (7 downto 0);
          outp: out std_logic_vector(7 downto 0));
end decoder;

architecture behave of decoder is
begin
    process (inp)
    begin
        case inp is
            when "000" => outp <= "00000001";
            when "001" => outp <= "00000010";
            when "010" => outp <= "00000100";
            when "011" => outp <= "00001000";
            when "100" => outp <= "00010000";
            when "101" => outp <= "00100000";
        end case;
    end process;
end;
```

```

        when "110" => outp <= "01000000";
        when "111" => outp <= "10000000";
        when others => outp <= "XXXXXXXX";
    end case;
    ins <= a xor b;
end process;
end behave;

```

Action

Make sure that all the signals referenced in the process are listed in the sensitivity list to remove any possible RTL/simulation mismatches. To eliminate the warning in the above test case, list the signals in the sensitivity list as shown in the corrected test case below.

```

library ieee;
use ieee.std_logic_1164.all;

entity decoder is
    port (inp: in std_logic_vector(2 downto 0);
          ins: out std_logic_vector (7 downto 0);
          a, b: in std_logic_vector (7 downto 0);
          outp: out std_logic_vector(7 downto 0));
end decoder;

architecture behave of decoder is
begin
    process (inp, a, b)
    begin
        case inp is
            when "000" => outp <= "00000001";
            when "001" => outp <= "00000010";
            when "010" => outp <= "00000100";
            when "011" => outp <= "00001000";
            when "100" => outp <= "00010000";
            when "101" => outp <= "00100000";
            when "110" => outp <= "01000000";
            when "111" => outp <= "10000000";
            when others => outp <= "XXXXXXXX";
        end case;
        ins <= a xor b;
    end process;
end behave;

```

CG291

@W: Ignoring parameter <size> in sensitivity list

A parameter is specified in the sensitivity list. The compiler ignores the presence of the parameter in the sensitivity list.

```
module test (bus_a,bus_out);
parameter size = 5;
input [size -1 :0] bus_a;
output [size -1 :0] bus_out;
reg [size -1:0] bus_out;
integer i;

always @(size, bus_a) // synthesis loop_limit 20001
  for (i=0; i <= size; i=i+1)
    bus_out <= bus_a + i;
endmodule
```

Action

To remove possible RTL/simulation mismatches, make sure that all signals referenced in the always block are listed in the sensitivity list. Parameters should not be listed in the sensitivity list. To eliminate the warning in the above test case, remove the referenced parameter from the sensitivity list as shown in the corrected test case below.

```
module test (bus_a,bus_out);
parameter size = 5;
input [size -1 :0] bus_a;
output [size -1 :0] bus_out;
reg [size -1:0] bus_out;
integer i;

always @(bus_a) // synthesis loop_limit 20001
  for (i=0; i <= size; i=i+1)
    bus_out <= bus_a + i;
endmodule
```

Warnings occur when the referenced signals in the always block are not listed in the sensitivity list as well as when unreferenced signals are listed in the sensitivity list. The sensitivity list is assumed to be complete and synthesis is done under this assumption.

CG292

@W: Ignoring localparam <param1> in sensitivity list.

The compiler encountered a localparam in the sensitivity list. Because local parameters are constants, there is no scope for the inclusion of events. In the test case below, local parameter param1 is specified in the sensitivity list which causes the compiler to issue the above warning.

```
module multiplier (operand1, operand2, result,clk);
localparam param1=3;
input [param1-2:0]operand1;
input [param1-2:0] operand2;
input clk;
output reg [3:0] result;

always@(clk,operand1,operand2,param1)
begin
    if (clk)
        result = (operand1 * operand2)*param1;
    else
        result = (operand1 * operand2)/param1;
end
endmodule
```

Action

Make sure that there are no local parameters specified inside a sensitivity list. To eliminate the warning in the above test case, remove local parameter param1 from the sensitivity list as shown in the corrected test case below.

```
module multiplier (operand1, operand2, result,clk);
localparam param1=3;
input [param1-2:0]operand1;
input [param1-2:0] operand2;
input clk;
output reg [3:0] result;
```

```
always@(clk,operand1,operand2)
begin
    if (clk)
        result = (operand1 * operand2)*param1;
    else
        result = (operand1 * operand2)/param1;
end
endmodule
```

CG293

@W: Ignoring initial statement

An initial block was encountered. Initial blocks are used in Verilog to initialize signals to values. The compiler ignores initial blocks, and no initial values are assigned to the signals. In the following test case, the warning appears due to the existence of initial blocks.

```
module dff1(q, d, clk, set, reset);
input d, clk, set, reset;
output q;
// declare q and qb to be reg, because assigned inside always
reg q, qb;
reg qs;

always @(posedge clk or posedge set or posedge reset)
begin
    if (reset)
        q <= 0;
    else if (set)
        q <= 1;
    else
        q <= d;
end
endmodule

module dff_tb;
reg clk,set,reset,d;
dff1 inst_1 (q,d,clk,set,reset);
initial
begin
reset = 0;
set = 0;
```

```
d =0;  
clk =0;  
#50 reset =1;  
#50 reset =0;  
#50 set =1;  
#50 set =0;  
end  
  
always  
#50 clk = ~clk;  
  
always  
#110 d = ~d;  
endmodule
```

Action

Make sure that initial blocks are not used in designs for synthesis as they are ignored and can cause possible mismatches between RTL and post synthesis simulation. In the above test case, module `dff_tb` is a testbench and should not be included as one of the files in the project file. Initial blocks are usually used in test benches for creating stimulus and should not be included in designs for synthesis.

CG294

@W: always block should contain at least one event control

An always block with no event triggering it was encountered. Logic is created to indicate that the statements in this always block are always true. In the test case below, since the always block has no specific event control, out is always driven by the signal preout.

Note that in the test case below, out drives preout and preout drives out in the assignment `{cout, preout} = out + cin;`; a combinational loop is detected for signal out.

```
module counter2(out, cout, data, load, cin, clk);
    output [7:0] out;
    output cout;
    input [7:0] data;
    input load, cin, clk;
    reg [7:0] out;
    reg cout;
    reg [7:0] preout;

    // create the 8-bit register
    always
    begin
        out = preout;
    end

    // calculate the next state of the counter and the carry out
    // note that we don't want load to affect cout, for performance
    // reasons

    always @(out or data or load or cin)
    begin
        {cout, preout} = out + cin;
        if (load) preout = data;
    end

endmodule
```

Action

Make sure that the `always` blocks in Verilog designs always have an event control. To eliminate the warning in the above test case, add the event to the `always` block as shown in the corrected test case below.

```
module counter2(out, cout, data, load, cin, clk);
    output [7:0] out;
    output cout;
    input [7:0] data;
    input load, cin, clk;
    reg [7:0] out;
    reg cout;
    reg [7:0] preout;

    // create the 8-bit register
    always @ (posedge clk)
    begin
        out = preout;
    end
```

```
// calculate the next state of the counter and the carry out
// note that we don't want load to affect cout, for performance
// reasons

always @(out or data or load or cin)
begin
    {cout, preout} = out + cin;
    if (load) preout = data;
end

endmodule
```

Adding the event to the always block registers the output with the preout value and causes the combinational loop of out to disappear.

CG295

@W: Warning "always block should contain at least one event control" repeated too many times. Suppressed for rest of run

The compiler encountered four or more always blocks without an event trigger. Logic is created to indicate that the statements in these always block are always true.

In the test case below, there are five always blocks that do not have any event control which causes the compiler to issue the above warning.

```
module comb(out, in, a, b, c, Pow);
    output [7:0] out;
    input [3:0] in;
    input [4:0] a, b;
    output [4:0] Pow,c;
    reg [4:0] Pow,c;
    assign out = 1'b1 ^ in;

    always
    begin
        Pow[0] = a[0] & b[0];
        c[0] = Pow[0];
    end
```

```
always
begin
    Pow[1] = a[1] & b[1];
    c[1] = Pow[1];
end

always
begin
    Pow[2] = a[2] & b[2];
    c[2] = Pow[2];
end

always
begin
    Pow[3] = a[3] & b[3];
    c[3] = Pow[3];
end

always
begin
    Pow[4] = a[4] & b[4];
    c[4] = Pow[4];
end

endmodule
```

Action

Make sure that the `always` blocks in Verilog designs always have an event control. To eliminate the warning in the above test case, add an event to all of the `always` block as shown in the corrected test case below.

```
module comb (out, in, a, b, c, Pow);
output [7:0] out;
input [3:0] in;
input [4:0] a, b;
output [4:0] Pow,c;
reg [4:0] Pow,c;
assign out = 1'b1 ^ in;

always @(a,b,c,out)
begin
    Pow[0] = a[0] & b[0];
    c[0] = Pow[0];
end
```

```
always @(a,b,c,out )
begin
    Pow[1] = a[1] & b[1];
    c[1] = Pow[1];
end

always @(a,b,c,out )
begin
    Pow[2] = a[2] & b[2];
    c[2] = Pow[2];
end

always @(a,b,c,out )
begin
    Pow[3] = a[3] & b[3];
    c[3] = Pow[3];
end

always @(a,b,c,out )
begin
    Pow[4] = a[4] & b[4];
    c[4] = Pow[4];
end

endmodule
```

CG296

@W: Incomplete sensitivity list; assuming completeness. Make sure all referenced variables in message CG290 are included in the sensitivity list.

The sensitivity list in the process (VHDL) or always block (Verilog) does not include all signals accessed within that process or always block. Warning **CG290** occurs in conjunction with the warning:

@W: Referenced variable b is not in sensitivity list

Verilog Test Case

This warning lists all signals that must be in the sensitivity list. The compiler assumes completeness; it infers the logic as if all signals referenced in the always block are in the sensitivity list. The test case below returns this warning because although signals `a` and `b` are referenced within the always block, `b` is not listed in the sensitivity list.

```
module compare(equal, a, b);
parameter size = 1;
output equal;
input [size-1:0] a, b; // declare inputs
reg equal;
always @(a)
begin
    equal = a == b;
end
endmodule
```

VHDL Test Case

The warning lists all the signals that must be in the sensitivity list. The compiler assumes completeness; it infers the logic as if all signals referenced in the process are in the sensitivity list. The test case below returns the first warning because signals `a` and `b` are referenced within the process, although `b` is not in the sensitivity list.

```
library ieee;
use ieee.std_logic_1164.all;

entity decoder is
    port (inp: in std_logic_vector(2 downto 0);
          ins: out std_logic_vector (7 downto 0);
          a, b: in std_logic_vector (7 downto 0);
          outp: out std_logic_vector(7 downto 0));
end decoder;

architecture behave of decoder is
begin
    process (inp,a)
    begin
        case inp is
            when "000" => outp <= "00000001";
            when "001" => outp <= "00000010";
            when "010" => outp <= "00000100";
            when "011" => outp <= "00001000";
        end case;
    end process;
end;
```

```
        when "100" => outp <= "00010000";
        when "101" => outp <= "00100000";
        when "110" => outp <= "01000000";
        when "111" => outp <= "10000000";
        when others => outp <= "XXXXXXXX";
    end case;
    ins <= a xor b;
end process;
end behave;
```

Action

Make sure that all signals referenced in the process (VHDL) or always block (Verilog) are listed in the sensitivity list to remove possible RTL/simulation mismatches.

Corrected Verilog Test Case

To eliminate the warning in the Verilog test case, add the missing signal to the always block as shown in the corrected test case below.

```
module compare(equal, a, b);
parameter size = 1;
output equal;
input [size-1:0] a, b; // declare inputs
reg equal;
always @(a,b)
begin
    equal = a == b;
end
endmodule
```

Corrected VHDL Test Case

To eliminate the warning in the VHDL test case, add the missing signal to the process line as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;
```

```

entity decoder is
    port (inp: in std_logic_vector(2 downto 0);
          ins: out std_logic_vector (7 downto 0);
          a, b: in std_logic_vector (7 downto 0);
          outp: out std_logic_vector(7 downto 0));
end decoder;

architecture behave of decoder is
begin
    process (inp, a, b)
    begin
        case inp is
            when "000" => outp <= "00000001";
            when "001" => outp <= "00000010";
            when "010" => outp <= "00000100";
            when "011" => outp <= "00001000";
            when "100" => outp <= "00010000";
            when "101" => outp <= "00100000";
            when "110" => outp <= "01000000";
            when "111" => outp <= "10000000";
            when others => outp <= "XXXXXXXX";
        end case;
        ins <= a xor b;
    end process;
end behave;

```

CG300

@E: posedge and negedge of the same signal not allowed

An always block is triggered by both the positive edge and the negative edge of the same signal. In the following test case, the sensitivity list of the always block contains both the positive and negative edge of the signal clk which causes the compiler to error out with the above message.

```

module edges (
    input clk,
    input [3:0] a,
    output reg [3:0] q );

```

```
always@(posedge clk, negedge clk) // Error!
begin
    q <= a;
end
endmodule
```

Action

Make sure that the always block is triggered by only a single edge of a signal as shown in the corrected test case below.

```
module edges (
    input clk,
    input [3:0] a,
    output reg [3:0] q);

    always@(posedge clk)
    begin
        q <= a;
    end
endmodule
```

CG301

@E: Can't mix posedge/negedge use with plain signal references

The sensitivity list of an always block contains references to both:

- the edge of a signal, such as either the posedge clk or the negedge clk
- a plain signal reference such as rst

In the test case below, the always statement contains both a posedge clk and rst signal which causes the compiler to error out with the above message.

```
module error (
    input clk, rst,
    input [3:0] a,
    output reg [3:0] q );
```

```
always@(posedge clk, rst)
begin
if (rst)
    q <= 4'b0000;
else
    q <= a;
end

endmodule
```

Action

Use either all edges or all plain signals, but do not mix them in an always block. The test case below uses all posedge signals in the always block to eliminate the above error.

```
module test1 (
    input clk, rst,
    input [3:0] a,
    output reg [3:0] q);

    always@(posedge clk, posedge rst) // all posedge signals
    begin
        if (rst)
            q <= 4'b0000;
        else
            q <= a;
    end

endmodule
```

CG303

@E: Event expression is too complex

The sensitivity list of an event control statement contains a complex expression that is not a plain signal or the posedge/negedge of a signal. The following test causes the compiler to error out with the above message.

```
module error (
    input clk,
    input en,
    input [3:0] b,
    output reg [3:0] q );

    always@(posedge clk & en) // event expression is too complex
begin
    q <= b;
end

endmodule
```

Action

Use a plain signal or the posedge/negedge of a signal in the sensitivity list to avoid this error. The corrected test case below shows how the posedge of the clk signal is used to infer a register.

```
module test1 (
    input clk,
    input en,
    input [3:0] b,
    output reg [3:0] q );

    always@(posedge clk) // posedge of clk is used
begin
    if (en)
        q <= b;
end

endmodule
```

CG304

@E: Range illegal for multi-dimensional reference <a>

A range is specified as the dimension of a multi-dimensional array (dimensions must always be constant expressions). In the test case below, the dimension of single-bit multi-dimensional array a is specified as a range instead of a constant which causes the compiler to error out with the above message.

```
module comb (q,temp);
output q;
input temp;
wire a[1:0];

assign a[1]=temp;
assign    q = a[1:0];

endmodule
```

Action

Make sure that array dimensions are always constant expressions. To eliminate the error in the above test case, specify a constant dimension for the array as shown in the corrected test case below.

```
module comb (q,temp);
output q;
input temp;
wire a[1:0];

assign a[1]=temp;
assign q = a[1];

endmodule
```

CG305

@E: Illegal usage of part-select expression variable <arrayb>

A part-select expression is used, however, the array is not indexed properly. In the test case below, while assigning a value to two-dimensional array arrayb, the target is specified as arrayb[1][0:2] (an illegal part select) which causes the compiler to error out with the above message.

```
module seq (input [3:0]a1,b1,output [3:0] temp, input clk);
reg [3:0] arrayb [3:0][0:2];

always @(posedge clk)
begin
arrayb[1][0:2] = a1&b1;
end
```

```
assign temp = arrayb[1][0];
endmodule
```

Action

Make sure that the array is indexed properly when assigning a value to the array. In the corrected test case below, the target array is specified as `arrayb[1][0]`.

```
module seq (input [3:0]a1,b1,output [3:0] temp, input clk);
reg [3:0] arrayb [3:0][0:2];

always @(posedge clk)
begin
    arrayb[1][0] = a1&b1;
end

assign temp = arrayb[1][0];
endmodule
```

CG306

@E: default_nettype is set to none. Net type of <temp1> must be declared.

The `default_nettype macro, which controls the net type created for implicit net declarations, is set to none and a net type is not explicitly declared (when the `default_nettype macro is set to none, all nets must be explicitly declared). In the following test case, `default_nettype is set to none but, because the net type of variable temp1 is not explicitly declared, the compiler errors out with the above message.

```
`default_nettype none
module test01 (input a, b, c, output reg out1, d);
parameter [3:0] cycles= 8;
assign temp1 = a;
```

```
always @(a,b,c)
begin
    out1 = a & b ;
    d= a || c;
end
endmodule
```

Action

When the `default_nettype macro is set to none, make sure that all net types are explicitly declared. In the corrected test case below, the net type of variable temp1 is declared as type wire.

```
`default_nettype none
module test01 (input a, b, c, output reg out1, d);
parameter [3:0] cycles= 8;
wire temp1;
assign temp1 = a;

always @(a,b,c)
begin
    out1 = a & b ;
    d= a || c;
end
endmodule
```

CG307

@E: Gate type <pmos> not supported. Please consider using a technology-specific primitive

A pmos primitive was encountered in the source code. For example, the error occurs in the following code.

```
module pmos_primitive(input data, control, output out);
pmos(out, data, control);
endmodule
```

Verilog PMOS primitives are not supported which results in the error.

Action

Use a generic primitive or replace the pmos statement with an assign statement.

```
assign out = ~control ? data : 1'bZ;
```

In the assign statement, when control is low, PMOS transistor turns ON, and out takes the value of data. When control is high, PMOS transistor is OFF which is equivalent to high impedance.

CG308

@E: Duplicate instance name <adder1>

The compiler found multiple instances with the same instance name.

Here is an example that generates this error.

```
module fa (input a, b, cin, output sum, carryout);
  assign sum = a ^ b ^ cin;
  assign carryout = ( (a & b) | (cin & a) | (cin & b) );
endmodule

module faN #( parameter N = 4)(input [N-1 : 0] A, B, input cin,
  output [N-1 : 0] sum, output carry);
  wire [N:1] temp ;
  fa adder1(.a(A[0]), .b(B[0]), .cin(cin), .sum(sum[0]),
    .carryout(temp[1]));
  fa adder1(.a(A[1]), .b(B[1]), .cin(temp[1]), .sum(sum[1]),
    .carryout(temp[2]));
  fa adder3(.a(A[2]), .b(B[2]), .cin(temp[2]), .sum(sum[2]),
    .carryout(temp[3]));
  fa adder4(.a(A[3]), .b(B[3]), .cin(temp[3]), .sum(sum[3]),
    .carryout(temp[4]));
  assign carry = temp[4] ;
endmodule
```

In the above example, there are two instances of fa with the same instance name adder1. This causes the compiler to error out saying - Duplicate instance name adder1.

Action

This is a syntax error. Use different instance names during component instantiation. In the above code, modifying the second instance with a different name, say adder2 removes the error.

```
module fa (input a, b, cin, output sum, carryout);
  assign sum = a ^ b ^ cin;
  assign carryout = ( (a & b) | (cin & a) | (cin & b) );
endmodule
```

```

module faN #( parameter N = 4)(input [N-1 : 0] A, B, input cin,
    output [N-1 : 0] sum, output carry);
wire [N:1] temp ;
fa adder1(.a(A[0]), .b(B[0]), .cin(cin), .sum(sum[0]),
    .carryout(temp[1]));
fa adder2(.a(A[1]), .b(B[1]), .cin(temp[1]), .sum(sum[1]),
    .carryout(temp[2]));
fa adder3(.a(A[2]), .b(B[2]), .cin(temp[2]), .sum(sum[2]),
    .carryout(temp[3]));
fa adder4(.a(A[3]), .b(B[3]), .cin(temp[3]), .sum(sum[3]),
    .carryout(temp[4]));
assign carry = temp[4] ;
endmodule

```

CG309

@W: Could not find module with name <FA1>, performing a case compare ...

Applies to mixed HDL designs; a VHDL instantiation was encountered within a Verilog module, but the component could not be found. In the following test case, the VHDL entity is instantiated from a Verilog top-level module, but the compiler cannot find FA1. Because Verilog is case sensitive, the compiler attempts to match the component in lowercase and issues the above warning.

```

library ieee;
use ieee.std_logic_1164.all;

entity fa is
port (a,b,cin : in std_logic;
      sum, cout : out std_logic);
end fa;

architecture df of fa is
begin
    sum <= a xor b xor cin;
    cout <= ((a and b) or ( cin and ( a or b))) ;
end df;

module faN #(parameter N = 4) (input [N-1 : 0] a, b, input cin,
output [N-1 : 0] sum, output cout);
wire [N:0] temp;

```

```

assign temp[0] = cin;
generate
begin :b1
  genvar i;
  for ( i = 0; i < N ; i = i + 1)
begin : u
  FA u1(.a(a[i]), .b(b[i]), .cin(temp[i]), .sum(sum[i]),
  .cout(temp[i+1]));
end
end ;
endgenerate

assign cout = temp[N] ;

endmodule

```

Action

When instantiating components in mixed HDL designs, make sure that both the name and case of the component in the Verilog top-level module match the name and case of the VHDL lower level entity.

CG310

@W: Found module with name <FA**>, attempting to match ...**

Applies only to mixed HDL designs; a VHDL instantiation was encountered within a Verilog module but a component with a different case was found.

In the following test case, the VHDL entity is instantiated from a Verilog top-level module, but the compiler cannot find FA. The compiler attempts to match the component in lowercase. As a result of the comparison, the compiler finds the component, attempts to match the ports and, when confirmed, generates the above warning.

```

library ieee;
use ieee.std_logic_1164.all;

entity fa is
port (a,b,cin : in std_logic;
      sum, cout : out std_logic);
end fa;

```

```
architecture df of fa is
begin
    sum <= a xor b xor cin;
    cout <= ((a and b) or ( cin and ( a or b))) ;
end df;

module faN #(parameter N = 4) ( input [N-1 : 0] a, b, input cin,
output [N-1 : 0] sum, output cout);
wire [N:0] temp;

assign temp[0] = cin;
generate
    begin :b1
    genvar i;
    for ( i = 0; i < N ; i = i + 1)
    begin : u
        FA ul(.a(a[i]), .b(b[i]), .cin(temp[i]), .sum(sum[i]),
                .cout(temp[i+1]));
    end
    end ;
endgenerate

assign cout = temp[N] ;
endmodule
```

Action

When instantiating components in mixed HDL designs, make sure that both the name and case of the component in the Verilog top-level module match the name and case of the VHDL lower level entity.

CG311

@W: Module with name <*FA1*> could not be found even with a case compare

Applies only to mixed HDL designs; the compiler encountered a VHDL instantiation within a Verilog module and it cannot find the component. In the following test case, the VHDL entity is instantiated from a Verilog top-level

module, but the compiler cannot find FA1. The compiler attempts to match the component in lowercase. As a result of the failed comparison, the compiler generates the above warning.

```
library ieee;
use ieee.std_logic_1164.all;

entity fa is
port (a,b,cin : in std_logic;
      sum, cout : out std_logic);
end fa;

architecture df of fa is
begin
  sum <= a xor b xor cin;
  cout <= ((a and b) or ( cin and ( a or b))) ;
end df;

module faN #(parameter N = 4) ( input [N-1 : 0] a, b, input cin,
  output [N-1 : 0] sum, output cout);
wire [N:0] temp;
assign temp[0] = cin;
generate
  begin :b1
    genvar i;
    for ( i = 0; i < N ; i = i + 1)
      begin : u
        FA1 u1(.a(a[i]), .b(b[i]), .cin(temp[i]), .sum(sum[i]),
          .cout(temp[i+1]));
      end
    end ;
  endgenerate
  assign cout = temp[N] ;
endmodule
```

Action

When instantiating components in mixed HDL designs, make sure that both the name and case of the component in the Verilog top-level module match the name and case of the VHDL lower level entity.

CG313

@E: Port : direction is incompatible with declaration (can only be type wire)

An input port is declared as a reg type. This includes reg, integer, time, real, and realtime. In the test case below, input b is declared as a type reg which causes the compiler to error out with the above message.

```
module comb (input a, input reg b, output out);
  assign out = a & b;
endmodule
```

Action

Make sure that the input ports are always declared as net data types. In the test case below, input b is changed to a net data type.

```
module comb (input a, input b, output out);
  assign out = a & b;
endmodule
```


CHAPTER 14

CG Messages 315 – 478

CG315

@E: Port not found -- possible mismatch across languages.

The compiler encountered a VHDL instantiation within a Verilog module, but the VHDL instantiation did not contain the referenced port. This error applies only to mixed HDL designs. In the following test case, the top-level Verilog module instantiates lower level VHDL entity fa, but because fa does not contain a port b, the compiler reports the above error.

```
library ieee;
use ieee.std_logic_1164.all;

entity fa is
  port (a,cin : in std_logic;
        sum, cout : out std_logic);
end fa;

architecture df of fa is
  signal b : std_logic;
begin
  sum <= a xor b xor cin;
  cout <= ((a and b) or (cin and (a or b))) ;
end df;
```

```

module faN #(parameter N = 4)
  (input [N-1 : 0] a, b, input cin, output [N-1 : 0] sum, output
  cout);
  wire [N:0] temp;

  assign temp[0] = cin;
  generate
    begin :b1
      genvar i;
      for ( i = 0; i < N ; i = i + 1)
        begin : u
          fa u1(.a(a[i]), .b(b[i]), .cin(temp[i]), .sum(sum[i]),
            .cout(temp[i+1]));
        end
    end ;
  endgenerate

  assign cout = temp[N] ;
endmodule

```

Action

Make sure that the ports between the top and lower level modules correspond. Modifying lower level entity `fa` as shown in the corrected test case below eliminates the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity fa is
  port (a,b, cin : in std_logic;
        sum, cout : out std_logic);
end fa;

architecture df of fa is
begin
  sum <= a xor b xor cin;
  cout <= ((a and b) or (cin and (a or b))) ;
end df;

```

CG317

@E: Expression does not evaluate to a constant

The value assigned to a parameter in a module instantiation statement did not evaluate to a constant. In the test case below, the parameter value is specified as input a in the instantiation of module and_n which causes the error.

```
module top (out, a, b);
    output out;
    input a,b;
    and_n #(a) (out,a,b);
endmodule

module and_n (out, a, b);
    parameter size =1;
    output [size:0]out;
    input [size:0]a,b;
    assign out = a & b;
endmodule
```

Action

Use only expressions that evaluate to run-time constants as parameter expressions. In the test case below, specifying a parameter expression of 0 (a constant) corrects the error.

```
module top (out, a, b);
    output out;
    input a,b;
    and_n #(0) (out,a,b);
endmodule

module and_n (out, a, b);
    parameter size =1;
    output [size:0]out;
    input [size:0]a,b;
    assign out = a & b;
endmodule
```

CG321

@E: Divide (/) is only supported for unsigned values

A design uses the division operation for any operand other than unsigned numbers. The division operation only can be used on unsigned values in Verilog. If a division operation is performed on a signed value as shown in the following test case, the compiler errors out.

```
module test (
    input clk,
    input signed [3:0] a, // a is a signed number
    output reg [2:0] q );
    wire [2:0] temp;

    always@(posedge clk)
    begin
        q <= temp;
    end

    assign temp = a/2; // division of signed value
endmodule
```

Action

Edit the code to make sure that the division operation is performed on unsigned values as shown in the corrected test case below.

```
module test (
    input clk,
    input [3:0] a, // a is an unsigned number
    output reg [2:0] q );
    wire [2:0] temp;

    always@(posedge clk)
    begin
        q <= temp;
    end

    assign temp = a/2; // division of unsigned value
endmodule
```

Division is supported only for unsigned values. When both the divisor and dividend are constants, the division is calculated by the compiler. For fractional results, the integral part is taken as the result of the division, for example, $3/3 = 1$, $5/4 = 1$, $3/4 = 0$. If the divisor and dividend are not constants, the divisor must be a non-zero positive constant power of 2.

Make sure that all division in the RTL code is done so that the divisor is a positive constant power of 2. In such cases, the logic created is essentially a right shift operation of the dividend. The number of times the shift operation occurs depends on the value of the divisor. For example, if the divisor is 8 ($2^{**}3$), the shift to the right is done three times. In the above test case, change the divisor b to a positive constant power of 2, such as 1, 2, 4, 8, and so on.

CG322

@E: mod (%) is only supported for unsigned values

A mod operation was performed on a signed value (the mod operation can be used only on unsigned values in Verilog). In the following test case, the object of the mod operation (a) is a signed value which results in the error.

```
module test (
    input clk,
    input signed [3:0] a, // a is a signed number
    output reg [2:0] q );
    wire [2:0] temp;

    always@(posedge clk)
    begin
        q <= temp;
    end

    assign temp = a % 2; // mod of signed value
endmodule
```

Action

Recode the design so that the mod operation is performed on unsigned values as shown in the corrected test case below.

```
module test (
    input clk,
    input [3:0] a, // a is an unsigned number
    output reg [2:0] q;
    wire [2:0] temp;

    always@(posedge clk)
    begin
        q <= temp;
    end

    assign temp = a % 2; // mod of unsigned value
endmodule
```

CG323

@E: mod (%) is only supported for pure signed or unsigned values

A mod operator was used on real data types (mod operators are only supported for pure signed or unsigned values). In the test case below, left argument a of the mod operator is a signed number whereas the right operand is a real data type which causes the error.

```
module test (
    input clk,
    input signed [3:0] a, output reg [2:0] q ,
    input unsigned [3:0 ] c ;
    wire [2:0] temp;

    always@(posedge clk)
    begin
        q <= temp;
    end
    assign temp = a % c;
endmodule
```

Action

Make sure that the operands of the mod operator are either signed or unsigned values. To eliminate the error in the above test case, change the real number to an unsigned number as shown in the corrected test case below.

```
module test (
    input clk,
    input signed [3:0] a, output reg [2:0] q ,
    input signed [3:0] c );
    wire [2:0] temp;

    always@(posedge clk)
    begin
        q <= temp;
    end
    assign temp = a % c;
endmodule
```

CG326

@E: Macro default_netttype cannot be used within a module

A `default_netttype macro was found within a module definition. In the following test case, macro `default_netttype appears inside the module definition which causes the error.

```
module comb (input a, b, c, output out1, d);
`default_netttype none
wire templ;
assign templ = a;
assign out1=a&b;
assign d=a || c;
endmodule
```

Action

Make sure that a `default_netttype macro is used only outside of a module definition as shown in the corrected test case below.

```
`default_netttype none
module comb (input a, b, c, output out1, d);
wire templ;
assign templ = a;
assign out1=a&b;
assign d=a || c;
endmodule
```

CG327

@E: <non> is not a valid option for macro default_nettype

The option specified in the `default_nettype macro is not valid. In the test case below, macro `default_nettype is assigned the invalid option non which causes the error.

```
`default_nettype  non
module test01 (input a, b,c,output reg out1,d);
parameter [3:0] cycles= 8;
wire temp1;
assign temp1 = a;

always @(a,b,c)
begin
    out1 = a & b ;
    d= a || c;
end
endmodule
```

Action

Make sure that the `default_nettype macro is always assigned a valid option. In the corrected test case below, the option for the `default_nettype macro is correctly changed to none.

```
`default_nettype  none
module test01 (input a, b,c,output reg out1,d);
parameter [3:0] cycles= 8;
wire temp1;
assign temp1 = a;

always @(a,b,c)
begin
    out1 = a & b ;
    d= a || c;
end
endmodule
```

CG328

@E: Macro <MIN> has no arguments.

While using macros defined with formal arguments, the actual arguments were not specified in the instantiation. In the test case below, macro MIN includes two formal arguments but, because no actual arguments are specified, the compiler errors out.

```
`define MIN(p1, p2) (p1)<(p2)?(p1):(p2)
module comb(i1, i2, o);
  input i1, i2;
  output o;
  reg o;

  always @(i1, i2) begin
    o = `MIN;
  end
endmodule
```

Action

Make sure that each formal argument of a macro includes an actual argument in the macro instantiation. The corrected test case below shows actual arguments added for macro MIN.

```
`define MIN(p1, p2) (p1)<(p2)?(p1):(p2)
module comb(i1, i2, o);
  input i1, i2;
  output o;
  reg o;

  always @(i1, i2) begin
    o = `MIN(i1, i2);
  end
endmodule
```

CG329

@E: Recursive macro is not supported - <MIN>

A design includes a macro that calls itself (recursive macro). In the following test case, the second MIN(p1,p2) macro in the formal definition in recursive which causes the error.

```
`define MIN(p1,p2) (`MIN(p1,p2))*p2
module comb (i1, i2, o);
  input i1, i2;
  output o;
  reg o;

  always @(i1, i2) begin
    o = `MIN(i1,i2);
  end
endmodule
```

Action

Avoid using recursive macros. In the corrected test case below, removing the macro reference from within itself and adding equivalent logic to the design eliminates to error.

```
`define MIN(p1,p2) (p1*p2)
module comb (i1, i2, o);
  input i1, i2;
  output o;
  reg o;
  reg temp;

  always @(i1, i2) begin
    temp = `MIN(i1,i2);
    o = `MIN(temp,i2);
  end
endmodule
```

CG330

@E: `elsif must follow `ifdef or `ifndef

An `elsif directive was encountered before its corresponding `ifdef or `ifndef directive. In the test case below, the `endif directive terminates the initial `ifdef directive which causes the compiler to error out when the `elsif directive is subsequently encountered.

```
module adder_8(cout, sum, a, b, cin);
`ifdef SMALL
    parameter WIDTH = 8;
`else
    parameter WIDTH =32;
`endif
`elsif MEDIUM
    parameter WIDTH = 16

output cout;
output [WIDTH -1:0] sum;
input cin;
input [WIDTH-1:0] a, b;

assign {cout, sum} = a + b + cin;
endmodule
```

Action

Make sure that there are matching `ifdef (or `ifndef), `else, and `endif directives. In the corrected test case below, pushing the `elsif block inside the `endif directive eliminates the error.

```
module adder_8(cout, sum, a, b, cin);
`ifdef SMALL
    parameter WIDTH = 8;
`elsif MEDIUM
    parameter WIDTH = 16
`else
    parameter WIDTH =32;
`endif
```

```
output cout;
output [WIDTH -1:0] sum;
input cin;
input [WIDTH-1:0] a, b;

assign {cout, sum} = a + b + cin;

endmodule
```

CG331

@E: `else must follow `ifdef or `ifndef

An `else directive was encountered before a corresponding `ifdef or `ifndef directive (Verilog allows text substitution through the use of `define statements; the compiler directives `ifdef, `else, and `endif are used for conditional compilation). In the following test case, if the macro SMALL is defined with a `define statement, the width of the adder is 8 bits and if undefined, the width of the adder is 32 bits. Commenting out the `ifdef directive causes the error.

```
module adder_8(cout, sum, a, b, cin);
//`ifdef SMALL
parameter WIDTH = 8;
`else
parameter WIDTH =32;
`endif

output cout;
output [WIDTH -1:0] sum;
input cin;
input [WIDTH-1:0] a, b;

assign {cout, sum} = a + b + cin;

endmodule
```

Action

Make sure that there are matching `ifdef, `else, and `endif directives. To eliminate the error in the above test case, enable the `ifdef definition as shown in the corrected test case below.

```
module adder_8(cout, sum, a, b, cin);
`ifdef SMALL
parameter WIDTH = 8;
`else
parameter WIDTH =32;
`endif

output cout;
output [WIDTH -1:0] sum;
input cin;
input [WIDTH-1:0] a, b;

assign {cout, sum} = a + b + cin;
endmodule
```

CG332

@E: `endif must follow `ifdef or `ifndef

An `endif directive does not follow a corresponding `ifdef or `ifndef directive (Verilog allows text substitution through the use of `define statements; the compiler directives `ifdef, `else, and `endif are used for conditional compilation). In the following test case, if the macro SMALL is defined with a `define statement, the width of the adder is 8 bits and if undefined, the width of the adder is 32 bits. The second `endif directive causes the error.

```
module adder_8(cout, sum, a, b, cin);
`ifdef SMALL
parameter WIDTH = 8;
`else
parameter WIDTH =32;
`endif
`endif

output cout;
output [WIDTH -1:0] sum;
input cin;
input [WIDTH-1:0] a, b;

assign {cout, sum} = a + b + cin;
endmodule
```

Action

Make sure that there are matching `ifdef, `else, and `endif directives. To eliminate the error in the above test case, delete or comment out the second `endif definition as shown in the corrected test case below.

```
module adder_8(cout, sum, a, b, cin);
`ifdef SMALL
parameter WIDTH = 8;
`else
parameter WIDTH =32;
`endif
```endif

output cout;
output [WIDTH -1:0] sum;
input cin;
input [WIDTH-1:0] a, b;

assign {cout, sum} = a + b + cin;
endmodule
```

# CG333

### **@N: Read directive translate\_on.**

The synthesis translate\_off and translate\_on directives isolate non-synthesizable code from synthesizable code. This note occurs when a translate\_on directive is encountered in the code. The translate\_off and translate\_on directives are always used in pairs; make sure that every translate\_off directive always has a corresponding translate\_on directive present in the HDL code. These notes are produced for the following test case. Each note contains the line number where the translate\_off or translate\_on directive was read.

```
module real_time (ina, inb, out);
 input ina, inb;
 output out;
 /* synthesis translate_off */
 realtime cur_time;
 /* synthesis translate_on */
 assign out = ina & inb;
endmodule
```

## Action

Make sure that there is one `translate_on` directive associated with each `translate_off` directive.

# CG334

### **@N: Read directive `translate_off`.**

The synthesis `translate_off` and `translate_on` directives isolate non-synthesizable code from synthesizable code. This note occurs when a `translate_off` directive is encountered in the code. The `translate_off` and `translate_on` directives are always used in pairs; make sure that every `translate_off` directive always has a corresponding `translate_on` present in the HDL code. These notes are produced for the following test case. Each note contains the line number where the `translate_off` or `translate_on` directive was read.

```
module real_time (ina, inb, out);
 input ina, inb;
 output out;
 /* synthesis translate_off */
 realtime cur_time;
 /* synthesis translate_on */
 assign out = ina & inb;
endmodule
```

## Action

Make sure that there is one `translate_on` directive associated with each `translate_off` directive.

## CG336

### **@E: Port declarations on concatenated port list should be in the same direction.**

Ports of different direction types are concatenated in the port list (only ports of the same direction can be concatenated). In the test case below, input port a and inout port b are concatenated ({a,b}) which results in the error.

```
module array ({a,b}, c, cout);
 input a,c;
 inout b;
 output cout;
 assign cout=a & c & b;
endmodule
```

#### Action

Make sure that only ports of the same direction are concatenated in the port list. In the corrected test case below, redefining inout port b as an input port allows ports a and b to be concatenated in the port list.

```
module array ({a,b}, c, cout);
 input a,c;
 input b;
 output cout;
 assign cout=a & c & b;
endmodule
```

## CG337

### **@E: Missing port declarations for concatenated port list.**

The direction of concatenated ports in the port list is not declared (only ports of the same direction can be concatenated in the port list). In the test case below, the direction of the concatenated ports ({a,b}) is not declared which causes the error.

```
module comb ({a,b}, cout);
output cout;
assign cout=a & b;
endmodule
```

## Action

Make sure that the directions of all the ports in the port list are declared. In the corrected test case below, declaring ports a and b as input allows the ports to be concatenated in the port list.

```
module comb ({a,b}, cout);
input a,b;
output cout;
assign cout=a & b;
endmodule
```

# CG338

## **@E: Variable <mem> is not a function, use [ ] for indexing**

Parentheses are used in place of square brackets when accessing a memory. In the following test case, the assign statement incorrectly uses parentheses which are reserved for function calls. The compiler expects brackets for memory indexing and errors out when they are not found.

```
module block_ram(input [7:0] din, output [7:0] dout,
 input [9:0] waddr, raddr, input we, clk);
reg [7:0] mem[0:1023];
reg [9:0] raddr_reg;

always @(posedge clk)
begin
raddr_reg <= raddr;
if (we)
 mem[waddr] <= din;
end

assign dout = mem(raddr_reg);

endmodule
```

## Action

Make sure that square brackets are used when indexing memory. To eliminate the error, replace the parentheses with square brackets as shown in the corrected test case below.

```
module block_ram(input [7:0] din, output [7:0] dout,
 input [9:0] waddr, raddr, input we, clk);
reg [7:0] mem[0:1023];
reg [9:0] raddr_reg;

always @(posedge clk)
begin
raddr_reg <= raddr;
if (we)
 mem[waddr] <= din;
end

assign dout = mem[raddr_reg];
endmodule
```

## CG339

### **@E: No definition for function/task <*addition*>**

A function was called that was not defined within the module. In the test case below, the statement dout <= addition(din, d2) calls the function addition and passes it the arguments din and d2. When the function cannot be found in the module, the compiler errors out.

```
module test (output reg [7:0] dout, input [7:0]
 din, d2, input clk, rst);
always @(posedge clk or negedge rst)
begin
if (!rst)
 dout <= 0;
else
 dout <= addition(din, d2);
end
endmodule
```

## Action

Make sure that the function being called is defined within the module. To eliminate this syntax error, add the function definition to the module as shown in the corrected test case below.

```
module test (output reg [7:0] dout, input [7:0]
 din, d2, input clk, rst);
 always @(posedge clk or negedge rst)
 begin
 if(!rst)
 dout <= 0;
 else
 dout <= addition(din, d2);
 end

 function addition;
 input [7:0] d, e;
 begin
 addition = d + e;
 end
 endfunction

endmodule
```

## CG341

### **@E: <datax> is already declared in this scope.**

A literal is declared more than once. In the following test case, datax is declared as a wire twice which results in the error.

```
module rega (clk,reset,ce,data,q);
 input clk,reset,ce;
 input data;
 output q;
 reg q ;
 wire datax;
 wire datax;
 assign datax = data;
```

```
always@ (posedge clk or posedge reset)
begin
 if (reset)
 q <= 1'b0;
 else if(ce)
 q <= datax;
 end
endmodule
```

## Action

This error usually requires editing the source code to remove the redundant literal. To eliminate the error, comment out one of the declarations as shown in the corrected test case below.

```
module rega (clk,reset,ce,data,q);
 input clk,reset,ce;
 input data;
 output q;
 reg q ;
 wire datax;
 //wire datax;
 assign datax = data;

 always@ (posedge clk or posedge reset)
 begin
 if (reset)
 q <= 1'b0;
 else if(ce)
 q <= datax;
 end
 endmodule
```

## CG342

### **@E: Expecting target variable, found <temp> -- possible misspelling**

An unknown variable was assigned within a loop statement. In the following test case, variable temp is assigned within the for loop, but is not declared within the module which causes the error.

```
module test_array(input [7:0] a, b, output [7:0] y);
reg [7:0]y;
integer i;
always @ (a, b)
 for(i = 0; i < 8; i = i + 1)
 temp = a[i] + b[i];
endmodule
```

## Action

Specify the predefined target variable within the for loop as shown in the corrected test case below to eliminate the error.

```
module test_array(input [7:0] a, b, output [7:0] y);
reg [7:0]y;
integer i;
always @ (a, b)
 for(i = 0; i < 8; i = i + 1)
 y[i] = a[i] + b[i];
endmodule
```

# CG345

## @E: Expecting only one default statement

More than one default statement is specified within a case statement (a case statement can have only one default statement). In the following test case, the second default statement within the case statement causes the error.

```
// 3 to 8 decoder
module deco3to8(input [2:0]in, output reg [7:0] out);
always @ (in)
 case (in)
 3'd0 : out <= 8'h01;
 3'd1 : out <= 8'h02;
 3'd2 : out <= 8'h04;
 3'd3 : out <= 8'h08;
 3'd4 : out <= 8'h10;
 3'd5 : out <= 8'h20;
 3'd6 : out <= 8'h40;
```

```

 3'd7 : out <= 8'h80;
 default : $display(" Invalid control signal ");
default : out <= 8'h0;
endcase
endmodule

```

## Action

Make sure that there is only one default statement associated with each case statement. To eliminate the error in the above test case, delete or comment out the second default statement as shown in the corrected test case below.

```

// 3 to 8 decoder
module deco3to8(input [2:0]in, output reg [7:0] out);
always @ (in)
 case (in)
 3'd0 : out <= 8'h01;
 3'd1 : out <= 8'h02;
 3'd2 : out <= 8'h04;
 3'd3 : out <= 8'h08;
 3'd4 : out <= 8'h10;
 3'd5 : out <= 8'h20;
 3'd6 : out <= 8'h40;
 3'd7 : out <= 8'h80;
 default : $display(" Invalid control signal ");
// default : out <= 8'h0;
 endcase
endmodule

```

# CG346

## **@N: Read full\_case directive.**

A case statement was encountered that has been defined as full case using a full\_case directive as in the test case below. When used with a case, casex, or casez statement, the full\_case directive indicates that all possible values have been given and that no additional hardware is needed to preserve signal values.

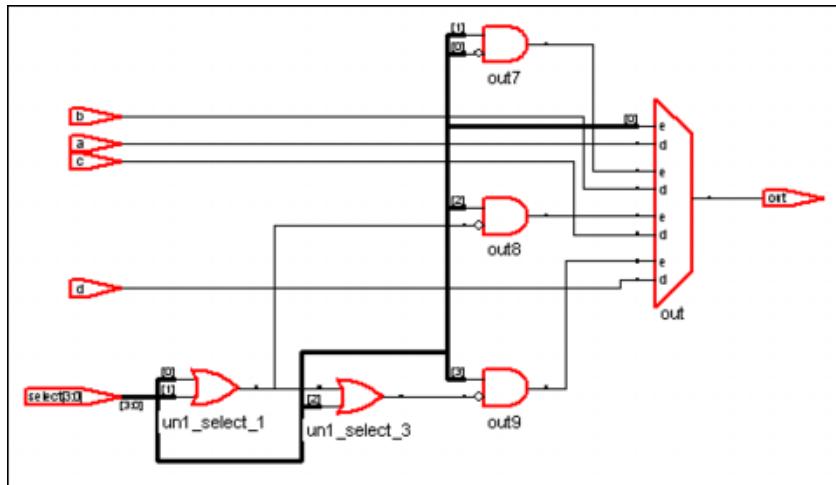
```
module muxnew3 (output reg out, input a,b,c,d, input [3:0] select);
```

```

always @(select or a or b or c or d)
begin
 casez (select) /* synthesis full_case */
 4'b???1: out = a;
 4'b??1?: out = b;
 4'b?1???: out = c;
 4'b1????: out = d;
 endcase
end
endmodule

```

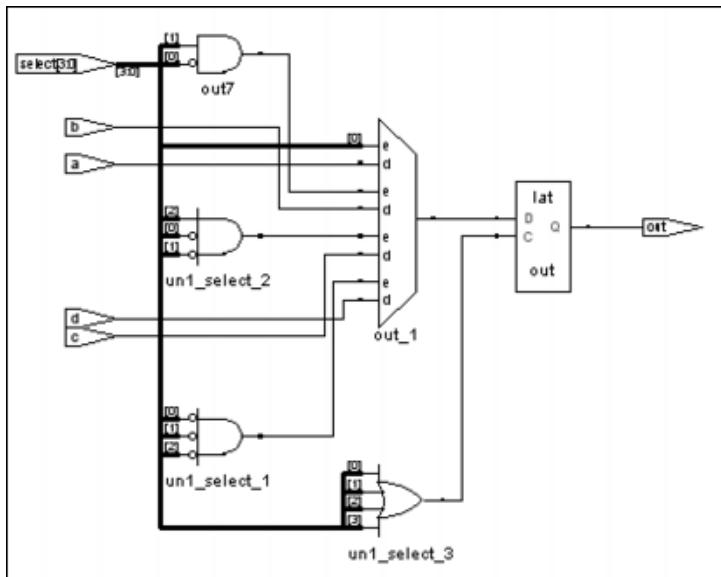
The following schematic shows the RTL view for the above module.



When the `full_case` directive is included as in the above test case, it instructs the synthesis tool not to preserve the value of `out` when all bits of `select` are zero. If the `full_case` directive is not specified, the above code does not specify what to do when the `select` bus is all zeros. If the `select` bus is being driven from outside the current module, the current module has no information about the legal values of `select`, and the synthesis tool must preserve the value of `out` when all bits of `select` are zero. Preserving the value of `out` requires the tool to add an extraneous, level-sensitive latch if `out` is not assigned elsewhere through every path of the `always` block. A warning message similar to the following is also generated:

"Latch generated from always block for signal out, probably caused by a missing assignment in an if or case statement"

The following schematic is the RTL generated by the synthesis tool when the full\_case directive is not specified. When the select bus has a value of 0000, the latch is disabled and out holds the previous value.



## Action

Make sure that if the case specified is complete (i.e., no logic for the unspecified cases must be implemented) that the full\_case directive is specified in the case statement. If full case is to be implemented, make sure that a default case directive is not specified in the case statement because this directive overrides the full\_case directive.

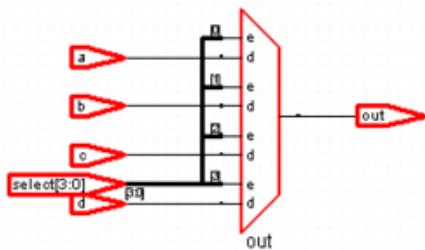
# CG347

## **@N: Read a parallel\_case directive.**

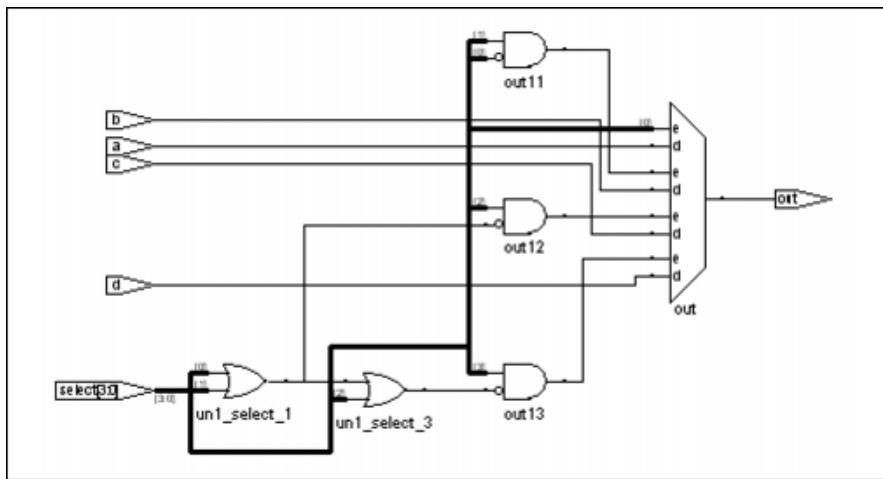
A case statement was encountered that has been defined as parallel case using a /\*synthesis parallel\_case\*/ directive as in the test case below.

```
module muxnew4 (output reg out, input a, b, c, d, input [3:0]select);
 always @ (select or a or b or c or d)
 begin
 casez (select) /* synthesis parallel_case */
 4'b????1: out <= a;
 4'b??1?: out <= b;
 4'b?1???: out <= c;
 4'b1????: out <= d;
 default: out <= 'bx;
 endcase
 end
endmodule
```

The following schematic shows the RTL view for the above module.



If parallel case is not specified explicitly with a parallel\_case directive, a priority encoder is inferred. The following schematic shows the RTL view when parallel case is not specified.



### Action

The `parallel_case` directive is a synthesis directive used only in Verilog designs. The presence of this directive forces a parallel-multiplexed structure rather than a priority-encoded structure. The parallel-multiplexed structure is useful because `case` statements are defined to work in priority order, executing (only) the first statement with a tag that matches the select value.

When the select bus is driven from outside the current module, the current module has no information about the legal values of select, and the software must create a chain of disabling logic so that a match on a statement tag disables all following statements. However, if you know the legal values of select, you can eliminate extra priority-encoding logic with the `parallel_case` directive. In the above example, if the only legal values of select are 4'b1000, 4'b0100, 4'b0010, and 4'b0001, and only one of the tags can be matched at a time, you can specify the `parallel_case` directive. Specifying the `parallel_case` directive causes the tag-matching logic to be parallel and independent, instead of chained.

## CG349

### @E: Expecting digit after '.' in real

A real number is assigned a value and the decimal point (.) is not followed by a digit. In the test case below, e is of type real and is assigned the value 3. The compiler errors out with the above message when it encounters the semicolon following the decimal point.

```
module real_test(a, b, c);
 real e, f;
 assign e = 3. ;
 assign f = 4.5 ;
endmodule
```

### Action

Make sure that a digit always follows the decimal point in a type real. Adding a 0 following the decimal point in the assignment for e as shown in the corrected test case below eliminates the error.

```
module real_test(a, b, c);
 real e, f;
 assign e = 3.0 ;
 assign f = 4.5 ;
endmodule
```

## CG351

### @E: Integer size out of range 1 to 1048576

The size constant specified in a sized number is zero (the size constant in a sized number is written only in decimal and specifies the number of bits in the number). In the test case below, the size constant in the expression is specified as 0 which causes the compiler to error out.

```
module cg351(a,b,c);
 input a, b;
 output c;
 assign c= a?b:0'b0;
endmodule
```

## Action

When using sized numbers, make sure that the size constants are always a non-zero unsigned decimal number. In the corrected test case below, the number 1 is used as a size constant.

```
module cg351(a,b,c);
 input a, b;
 output c;
 assign c= a?b:1'b0;
endmodule
```

# CG353

### **@E: Expecting digit in radix <8>**

A value is assigned that is not supported by the specified radix. In the following test case, the radix assigned to the q1 value is octal (o), but the value assigned includes non-octal digits (88) which results in message.

```
module test (input clk, reset, input [7:0] d, output reg [7:0]
q1); always @ (posedge clk or posedge reset)
 if (reset)
 q1 <= 8'o88;
 else
 q1 <= d;
endmodule
```

Similarly, in the test case below, the compiler expects decimal digits and errors out when hex (ab) characters are assigned.

```
module test (input clk, reset, input [7:0] d, output reg [7:0] q1);
 always @ (posedge clk or posedge reset)
 if (reset)
 q1 <= 8'dab;
 else
 q1 <= d;
endmodule
```

## Action

Specifying the appropriate value for the specified number system as shown below eliminates the errors in the above test cases.

```
module test (input clk, reset, input [7:0] d, output reg [7:0] q1);
 always @ (posedge clk or posedge reset)
 if (reset)
 q1 <= 8'o77;
 else
 q1 <= d;
endmodule

module test (input clk, reset, input [7:0] d, output reg [7:0] q1);
 always @ (posedge clk or posedge reset)
 if (reset)
 q1 <= 8'hab;
 else
 q1 <= d;
endmodule
```

# CG355

## @E: Unknown escape character </>

The compiler encountered an invalid escape character. In the following test case, \l is not a valid escape character which causes the error.

```
module display1;
reg [7:0] str;
initial
 str = "\1\%123456" ;
endmodule
```

## Action

Use the valid escape character for strings. If a back-slash character (\) is part of the string, precede it with another back-slash. To eliminate the error in the above test case, replace the quoted string with either \\%123456 or \n%123456.

# CG357

## **@E: Illegal reference to array <a1>**

An array without proper indexing is assigned to a variable. In the test case below, output e is assigned to array a1 without an index which causes the error.

```
module comb (input a,b,c,d, output e,f);
wire a1[1:0];
assign a1[1]=a&b;
assign a1[0]=c&d;
assign e=a1 & c;
assign f=a1[0] & a;
endmodule
```

## Action

When using arrays, make sure that the index of each dimension of the array is specified. In the corrected test case below, providing an index for the missing array dimension allows the assignment.

```
module comb (input a,b,c,d, output e,f);
 wire a1[1:0];
 assign a1[1]=a&b;
 assign a1[0]=c&d;
 assign e=a1[1] & c;
 assign f=a1[0] & a;
endmodule
```

## CG359

### @E: Multiple .\*'s illegal in instantiation

Multiple ".\*"s (dot-stars) are used in an implicit instantiation (the ".\*" connection is used to implicitly instantiate ports when the instance port name and size exactly match a connecting module's variable port name and size). In the test case below, the instantiation has multiple ".\*" connections which causes the compiler to error out with the above message.

```
module nand_1 (out, a, b);
 output [7:0]out;
 input [7:0]a,b;
 assign out=~(a & b);
endmodule

module nand_top(out, a, b);
 output [7:0]out;
 input [7:0] a, b;
 nand_1 my_nand (.*,.*);
endmodule
```

### Action

When using a ".\*" connection for instantiating components, make sure that the ".\*" is specified only once as shown in the corrected test case below.

```
module nand_1 (out, a, b);
 output [7:0]out;
 input [7:0]a,b;
 assign out=~(a & b);
endmodule
```

```
module nand_top(out, a, b);
output [7:0]out;
input [7:0] a, b;
nand_1 my_nand (.*);
endmodule
```

## CG360

### @W: Removing wire <temp>, as there is no assignment to it.

A net data type is declared and no assignment is made to that data type. In the following test case, temp is declared as a net data type without an assignment which results in the above warning.

```
module adder (cout, sum, a, b, cin);
output cout;
output [7:0] sum;
input [7:0] a, b;
input cin;
wire temp;
assign sum=a + b;
assign cout=cin;
endmodule
```

### Action

Make sure that all declared data types are assigned values in the design. In the corrected test case below, logic is added to drive temp and affect its output.

```
module adder (cout, sum, a, b, cin);
output cout;
output [7:0] sum;
input [7:0] a, b;
input cin;
wire [7:0]temp;
assign temp=a+b;
assign sum=temp;
assign cout=cin;
endmodule
```

# CG361

## @W: Case tag overlaps with a previous tag (case branches <1> and <3>)

The compiler encountered a case tag that overlaps with a previous tag. In the test case below, the parallel\_case directive indicates to the compiler that all of the case branches have equal priority. The compiler finds two branches having the same tag (2'b10).

```
module cg361(input [1:0]arg, output reg result,input a,b,c,d);
always @(arg,a,b,c,d)
 case (arg) /*synthesis parallel_case*/
 2'b00 : result = d;
 2'b10 : result = a;
 2'b01 : result = b;
2'b10 : result = c;
 2'b11 : result =c;
 endcase
endmodule
```

### Action

Make sure that unique case tags are used as shown in the corrected test case below.

```
module cg361(input [1:0]arg, output reg result,input a,b,c,d);
always @(arg,a,b,c,d)
 case (arg) /*synthesis parallel_case*/
 2'b00 : result = d;
 2'b10 : result = a;
 2'b01 : result = b;
 2'b11 : result =c;
 endcase
endmodule
```

# CG364

## @N: Synthesizing module <*module\_name*>.

The compiler synthesized a module. In the example below, four modules (mux, reg8, rotate, and top1) are synthesized with this note reported for each module.

```
module mux(out, a, b, sel); // mux
 output [7:0] out;
 input [7:0] a, b;
 input sel;
 assign out = sel ? a : b;
endmodule

module reg8(q, data, clk, rst); // eight bit register
 output [7:0] q;
 input [7:0] data;
 input clk, rst;
 reg [7:0] q;

 always @(posedge clk or posedge rst)
 begin
 if (rst)
 q = 0;
 else
 q = data;
 end
endmodule

module rotate(q, data, clk, r_l, rst); // rotates bits or loads
 output [7:0] q;
 input [7:0] data;
 input clk, r_l, rst;
 reg [7:0] q;

 // when r_l is high, it rotates; if low, it loads data
 always @(posedge clk or posedge rst)
 begin
 if (rst)
 q = 8'b0;
 else if (r_l)
 q = {q[6:0], q[7]};
 else
 q = data;
 end
endmodule
```

```
// -----
// Top level design, example #1. The port connections are
// listed in order:
// -----

module top1(q, a, b, sel, r_l, clk, rst);
output [7:0] q;
input [7:0] a, b;
input sel, r_l, clk, rst;
wire [7:0] mux_out, reg_out;
mux mux_1 (mux_out, a, b, sel);
reg8 reg8_1 (reg_out, mux_out, clk, rst);
rotate rotate_1 (q, reg_out, clk, r_l, rst);
endmodule
```

### Action

Note that the top-level module is the last module analyzed. You can change the top-level module from the Verilog/VHDL tab in the implementation options. The corresponding Tcl command in VHDL or Verilog is:

```
set_option -top_module "top2"
```

# CG366

### @E: Second argument for <\$readmemb> must be a memory.

The second argument of a \$readmemb or \$readmemh system task is not a memory variable. In the test case below, the second argument of task \$readmemb is specified as an input port (in) which causes the compiler to error out.

```
module top(input [7:0]in, input clk, input [2:0] raddr,
 waddr, output [7:0] out3);
reg [7:0] mem [7:0];
initial
begin
 $readmemb("data.dat",in);
end
```

```
always @(posedge clk)
mem[waddr] <= in;
assign out3 = mem[raddr];
endmodule
```

## Action

When using system task \$readmemb or \$readmemh, make sure that the second argument is a memory variable. In the corrected test case below, changing the second argument to memory variable mem eliminates the error.

```
module top(input [7:0]in, input clk, input [2:0] raddr,
waddr, output [7:0] out3);
reg [7:0] mem [7:0];
initial
begin
$readmemb("data.dat" ,mem);
end

always @(posedge clk)
mem[waddr] <= in;
assign out3 = mem[raddr];
endmodule
```

# CG367

## **@E: EOF within block comment in data file for task \$readmemb**

When using system tasks \$readmemb and \$readmemh, the file to be read included block comments that did not end with a \*/.

```
module top(input in, clk, input [7:0] in1, input [2:0] raddr,
waddr, output[7:0] out3);
parameter x = 1;
reg [7:0] mem [7:0];
initial
begin
$readmemb("data.dat" , mem);
end

always @(posedge clk)
mem[waddr] <= in1;
```

```
assign out3 = mem[raddr];
endmodule
/* File data.dat >>
100000
1000_00
0000_00
0000_00
/* this is to check block comment
```

## Action

Make sure that the file to be read by the \$readmemb and \$readmemh system tasks contains only binary or hex numbers, spaces, single-line comments, and block comments with proper start and end punctuation.

# CG370

### @W: Expecting two arguments, found <3>, ignoring.

The number of arguments specified for a task is more than the number expected. In the test case below, system task \$readmemb, which requires two arguments, includes a third argument (1) which is ignored as indicated in the message.

```
module top(input [7:0]in, input clk, input [2:0] raddr,
 waddr, output [7:0] out3);
reg [7:0] mem [7:0];
initial
begin
 $readmemb ("data.dat", mem, 1);
end

always @(posedge clk)
mem[waddr] <= in;
assign out3 = mem[raddr];
endmodule
```

## Action

Make sure that only the required number of arguments are passed to a task. In the corrected test case below, two arguments are correctly passed to task \$readmemb.

```
module top(input [7:0]in, input clk, input [2:0] raddr,
 waddr, output [7:0] out3);
 reg [7:0] mem [7:0];
 initial
 begin
 $readmemb("data.dat", mem);
 end

 always @(posedge clk)
 mem[waddr] <= in;
 assign out3 = mem[raddr];
endmodule
```

## CG371

### **@W: Cannot find data file <*data1.dat*> for task <\$readmemh>??**

A file required by a function or task cannot be found. In the test case below, the data1.dat file specified in system task \$readmemh is not found in the current location which results in the above message.

```
module top(input [7:0]in, input clk, input [2:0] raddr,
 waddr, output [7:0] out3);
 reg [7:0] mem [7:0];
 initial
 begin
 $readmemb("data1.dat", mem);
 end

 always @(posedge clk)
 mem[waddr] <= in;
 assign out3 = mem[raddr];
endmodule
```

### Action

Make sure that the data file exists either in the current location or in the location specified in the task or function.

## CG373

### **.\* and .name port connections are illegal on undefined modules xxxx**

An undefined module is instantiated using implicit instantiation such as the .\* or .name construct. If an undefined module is instantiated using named or positional instantiation, the Verilog compiler creates a black box for that module. In the test case below, implicit instantiation construct .\* is used to instantiate undefined module adder which causes the compiler to error out.

```
module adder2(cout, sum, a, b, cin);
parameter t=2;
output [t-1:0]cout;
output [t-1:0]sum;
input [t-1:0]a, b;
input cin;
adder #(.w(t)) my_adder (.*);
endmodule
```

### User Action

When using implicit instantiation, make sure that the module being instantiated is defined in the scope of the project as shown in the corrected test case below.

```
module adder2(cout, sum, a, b, cin);
parameter t=2;
output [t-1:0]cout;
output [t-1:0]sum;
input [t-1:0]a, b;
input cin;
adder #(.w(t)) my_adder (.*);
endmodule
```

```
module adder(cout, sum, a, b, cin);
parameter w=16;
output [w-1:0]cout;
output [w-1:0]sum;
input [w-1:0]a, b;
input cin;
assign {cout,sum}=a+b+cin;
endmodule
```

## CG374

### **@E: Variable <c> does not exist in the module <top>**

A ".\*" (dot-star) connection is used to instantiate a module that contains either different port names or an extra port. In the test case below, module top instantiates module comb using a ".\*" connection; the extra port (c) in module comb causes the compiler to error out.

```
module comb (input a, b, c, output out1, d);
assign d= c || b;
assign out1 = a & b ;
endmodule

module top (input a, b, output out1, d);
comb u (.');
endmodule
```

### Action

When using a ".\*" connection, make sure that the instance port names and sizes exactly match the variable port names and sizes of the connecting module as shown in the corrected test case below.

```
module comb (input a, b, c, output out1, d);
assign d= c || b;
assign out1 = a & b ;
endmodule

module top (input a, b, c, output out1, d);
comb u (.');
endmodule
```

# CG375

## @E: Expecting file name as first argument to <\$readmemb>

The first argument of a \$readmemb or \$readmemh system task was not a file name. In the test case below, the first argument of system task \$readmemb is specified as input port in which causes the compiler to error out.

```
module top(input [7:0]in, input clk, input [2:0] raddr,
 waddr, output [7:0] out3);
 reg [7:0] mem [7:0];
 initial
 begin
 $readmemb(in, mem);
 end

 always @(posedge clk)
 mem[waddr] <= in;
 assign out3 = mem[raddr];
 endmodule
```

### Action

While using the system task \$readmemb or \$readmemh, make sure that the first argument is always a file name. In the corrected test case below, the first argument of the \$readmemb task is changed to the name of the associated memory file (data.dat for the test case).

```
module top(input [7:0]in, input clk, input [2:0] raddr,
 waddr, output [7:0] out3);
 reg [7:0] mem [7:0];
 initial
 begin
 $readmemb("data.dat", mem);
 end

 always @(posedge clk)
 mem[waddr] <= in;
 assign out3 = mem[raddr];
 endmodule
```

# CG376

## @E: Expecting variable name

A reserved word was encountered when a net, register, or variable name was expected.

### Case 1

In the following Verilog test case, a time type declaration is associated with output (a reserved Verilog keyword) as the register name which results in the error.

```
module test (a, b, out);
 input a, b;
 output out;
 time output;
 assign out = a & b;
endmodule
```

### Action

Use non-reserve words to declare net and register names. In the corrected test case below, the name of the register has been changed to out1.

```
module test (a, b, out);
 input a, b;
 output out;
 time out1;
 assign out = a & b;
endmodule
```

### Case 2

In the following test case, the SystemVerilog reserved keyword do is incorrectly used as variable name which results in the error.

```
module src (din1,,out);
 input [7:0] din1;
 output [7:0] out;
 logic [7:0] do;
 assign do = ~din1;
 assign out = do ;
endmodule
```

## Action

Use non-reserve words to declare variable names. In the corrected test case below, the name of the variable has been changed to do1.

```
module src (din1,,out);
 input [7:0] din1;
 output [7:0] out;
 logic [7:0] do1;
 assign do1 = ~din1;
 assign out = do1 ;
endmodule
```

# CG377

## **@W: Newline within a string in a comment must be preceded with a '\', ignoring property ...**

The compiler is ignoring an attribute or directive because a portion of the string value extends to a second line without being preceded by a line-continuation character (\). In the test case below, the value for the `syn_ramstyle` attribute (`select_ram`) extends to a second line without the required line-continuation character which results in the above message.

```
module cg377 (input clk, we, input [9:0] rdaddr, wraddr,
 input [15:0] din,
 output [15:0] dout);
 reg [9: 0] reg_rdaddr ;
 reg [15:0] mem[0 : 1023] /* synthesis syn_ramstyle = "select_
 ram" */;
```

```
always @ (posedge clk)
begin
 if (we)
 mem[wraddr] <= din;
 reg_rdaddr <= rdaddr ;
 end

 assign dout = mem[reg_rdaddr] ;

endmodule
```

## Action

This warning can be corrected either by using a line-continuation character if the string continues on a second line or by splitting the line so that the complete attribute or directive is on the same line. In the corrected test case below, a line-continuation character is added to join the two lines of the select\_ram value string.

```
module cg377 (input clk, we, input [9:0] rdaddr, wraddr,
 input [15:0] din,
 output [15:0] dout);
 reg [9: 0] reg_rdaddr ;
 reg [15:0] mem[0 : 1023] /* synthesis syn_ramstyle = "select_\
 ram" */;

 always @ (posedge clk)
 begin
 if (we)
 mem[wraddr] <= din;
 reg_rdaddr <= rdaddr ;
 end

 assign dout = mem[reg_rdaddr] ;

endmodule
```

# CG380

## @E: Multiple event control statements illegal within always\_ff.

The SystemVerilog `always_ff` process models sequential logic that is triggered by clocks. In this case, there are multiple event-control statements in the `always_ff` block. In the test case below, the two event control statements (@) in the `always_ff` block cause the compiler to error out.

```
module seq (in,clk,rst,treg);
 input in,clk,rst;
 output treg;
 reg treg;

 always_ff
 begin
 @(posedge rst)treg<=0;
 @ (posedge clk)treg<=in;
 end
endmodule
```

### Action

When using an `always_ff` block, make sure that the block contains only one event control (and no blocking timing controls) as shown in the corrected test case below.

```
module seq (in,clk,rst,treg);
 input in,clk,rst;
 output treg;
 reg treg;

 always_ff @(posedge clk or posedge rst)
 begin
 if (rst)
 treg<=0;
 else
 treg<=in;
 end
endmodule
```

# CG381

## @E: Event statement illegal within <always\_comb>

An event statement was encountered in an always\_comb block. SystemVerilog provides a special always\_comb process for modeling combinational logic (the logic inferred from an always\_comb process is combinational). In the test case below, event statement @ (a, b) in the always\_comb block causes the compiler to error out.

```
module comb (a,b,out1);
 input a,b;
 output out1;
 reg out1;

 always_comb@(a,b)
 out1<=a & b;

endmodule
```

### Action

Make sure that there are no event statements appearing in an always\_comb block as shown in the corrected test case below.

```
module comb (a,b,out1);
 input a,b;
 output out1;
 reg out1;

 always_comb
 out1<=a & b;

endmodule
```

# CG382

## @E: Event trigger statement illegal within <always\_comb>

An event trigger statement was encountered in an always\_comb block. SystemVerilog provides a special always\_comb process for modeling combinational logic (the logic inferred from an always\_comb process is combinational). In the test case below, event trigger statement if(a)->r\_data in the always\_comb block causes the compiler to error out.

```
module comb (a, b,out1);
 input a,b;
 output out1;
 reg out1;
 event r_data;

 always_comb
 begin
 if (a) ->r_data;
 out1 = a & b;
 end
 endmodule
```

### Action

Avoid event triggering statements within an always\_comb block as shown in the corrected test case below.

```
module comb (a, b,out1);
 input a,b;
 output out1;
 reg out1;

 always_comb
 begin
 out1 = a & b;
 end
 endmodule
```

# CG383

## @E: Wait statement illegal within <always\_comb>

A wait statement was encountered in an always\_comb block. SystemVerilog provides a special always\_comb process for modeling combinational logic (the logic inferred from an always\_comb process is combinational). In the test case below, wait statement wait(a) in the always\_comb block causes the compiler to error out.

```
module comb (a, b,out1);
 input a,b;
 output out1;
 reg out1;

 always_comb
 begin
 wait(a);
 out1 = a & b;
 end
 endmodule
```

### Action

Do not use wait statements within an always\_comb block as shown in the corrected test case below.

```
module comb (a, b,out1);
 input a,b;
 output out1;
 reg out1;

 always_comb
 begin
 out1 = a & b;
 end
 endmodule
```

## CG384

### @E: fork/join illegal within <always\_comb>

A fork/join statement was encountered in an always\_comb block. SystemVerilog provides a special always\_comb process for modeling combinational logic (the logic inferred from an always\_comb process is combinational). In the test case below, the fork/join statement in the always\_comb block causes the compiler to error out.

```
module comb (input [3:0] a, output reg [2:0] q);
 always_comb
 begin
 fork
 q = a[3:1] & a[2:0];
 join
 end
 endmodule
```

### Action

Do not use fork/join statements within an always\_comb block as shown in the corrected test case below.

```
module comb (input [3:0] a, output reg [2:0] q);
 always_comb
 begin
 q = a[3:1] & a[2:0];
 end
 endmodule
```

## CG385

### @E: Disable statement illegal within <always\_comb>

A disable statement was encountered in an always\_comb block. SystemVerilog provides a special always\_comb process for modeling combinational logic (the logic inferred from an always\_comb process is combinational). In the test case below, the disable statement in the always\_comb block causes the compiler to error out.

```
module comb (a, b, c);
 input a, b;
 output c;
 reg c;

 always_comb
 begin :BLK2
 c = a & b;
 disable BLK2;
 end
endmodule
```

## Action

Avoid disable statements within an always\_comb block as shown in the corrected test case below.

```
module comb (a, b, c);
 input a, b;
 output c;
 reg c;

 always_comb
 begin :BLK2
 c = a & b;
 end
endmodule
```

# CG386

## **@E: `endif count mismatch at end of source**

An `endif statement was missing from the end of a module. In the test case below, there is an `ifdef statement, but no subsequent `endif statement prior to the endmodule to indicate the end of the `ifdef block.

```
module cg386 (a, b, c);
output a;
input b, c;
`ifdef behavioral
 wire a = b & c;
`else
 and a1 (a,b,c);
`endmodule
```

## Action

Make sure that there is a matching `endif statement for each `ifdef or `ifndef statement. In the corrected test case below, a matching `endif statement is added to mark the end of the `ifdef block.

```
module cg386 (a, b, c);
output a;
input b, c;
`ifdef behavioral
 wire a = b & c;
`else
 and a1 (a,b,c);
`endif
endmodule
```

## CG390

### **@E: Repeat multiplier in concatenation evaluates to <0>**

A negative value was used in a concatenation operation as shown in the following test case.

```
module test_8(test, a, b);
output [7:0] test;
input [7:0] a, b;
wire [7:0] test;
assign test = {-8{a}};
endmodule
```

## Action

Edit the code that has a negative value for the concatenation. To correct the error in the above test case, change the assign statement as shown in the corrected test case below.

```
module test_8(test, a, b);
 output [7:0] test;
 input [7:0] a, b;
 wire [7:0] test;
 assign test = {8{a}};
endmodule
```

The result assigns test[7:0] to { a,a,a,a,a,a,a,a }.

## CG393

### **@E: Expecting block name for For-generate statement**

The compiler detected a for loop generate statement in which a generate-block identifier inside the loop was not provided. In the test case below, the for-generate statement in top-level module adder\_4 has no block name which causes the compiler to error out.

```
module adder (sum,c0,a,b,cin);
 input a,b;
 output sum;
 output c0;
 input cin;
 assign {c0,sum} = a + b + cin;
endmodule

module adder_4(a,b,cout,sum,cin);
 input [3:0]a,b;
 output [3:0]sum;
 input cin;
 output cout;
 wire [4:0]c0;
 assign c0[0] = cin;
 assign cout = c0[4];
```

```
generate
genvar i;
begin
 for (i=0; i<=3; i = i + 1)
 begin
 adder add (sum [i], c0[i+1], a [i],b[i], c0[i]);
 end
 end
endgenerate
endmodule
```

## Action

Include a block name in the for-generate statement as shown in the corrected test case below.

```
module adder (sum,c0,a,b,cin);
input a,b;
output sum;
output c0;
input cin;
assign {c0,sum} = a + b + cin;
endmodule

module adder_4(a,b,cout,sum,cin);
input [3:0]a,b;
output [3:0]sum;
input cin;
output cout;
wire [4:0]c0;
assign c0[0] = cin;
assign cout = c0[4];

generate
genvar i;
begin
 for (i=0; i<=3; i = i + 1)
 begin : ul
 adder add (sum [i], c0[i+1], a [i],b[i], c0[i]);
 end
 end
endgenerate
endmodule
```

## CG396

### **@E: All unreferenced modules are black boxes - cannot set top level module**

The project includes a non-existent file. A common cause for this error is moving a project file that contains relative paths to the source files. As an example, if the project file includes the line

```
add_file -verilog ".../srr/test.v"
```

and there is no such file relative to the current project location, the compiler errors out.

#### Action

Make sure that any source file specified in the project is at the corresponding location relative to the project file.

## CG397

### **@E: Can't synthesize UDP primitives yet**

The Verilog language allows you to specify User-Defined Primitives (UDPs). However, the compiler does not support such primitives currently. In this case, the compiler encountered a UDP as shown in the test case below.

```
primitive D_EDGE_FF(Q, CLK, DATA);
 output Q;
 reg Q;
 //reg Q;
 input DATA, CLK;
 initial Q = 0;
 table
 // CLK DATA Q (state) Q (next)
 (01) 0 : ? : 0;
 (01) 1 : ? : 1;
 (0x) 1 : 1 : 1;
 (0x) 0 : 0 : 0;
 //Ignore negative edge of clock
```

```
(?0) ? : ? : -;
//Ignore data changes on steady clock :
? (??) : ? : -;
endtable
endprimitive

module REG4 (CLK, DIN, DOUT);
input CLK;
input [0:3] DIN;
output [0:3] DOUT;
D_EDGE_FF DLAB0 (DOUT[0], CLK, DIN[0]);
D_EDGE_FF DLAB1 (DOUT[1], CLK, DIN[1]);
D_EDGE_FF DLAB2 (DOUT[2], CLK, DIN[2]);
D_EDGE_FF DLAB3 (DOUT[3], CLK, DIN[3]);
endmodule
```

## Action

The compiler currently does not support UDPs. If a UDP is to be instantiated in the final design, make sure that a synthesizable definition of the module exists to perform the function of the UDP.

This error can occur while reading files with UDPs that are not instantiated in the design. Prior to synthesis, the tool checks syntax and also determines if the HDL code can be synthesized before synthesis begins.

# CG402

## @E: Multiple writes on bit <1> of <equal>

SystemVerilog; the compiler encountered more than one write for a bit of a variable declared using the var keyword. In the test case below, output port equal is declared as a variable using the var keyword, but because there are multiple write operations for bit 1 of the variable, the compiler errors out.

```
module cg402(equal, a, b);
output var [1:0]equal;
input a,b;
```

```
always @(a,b)
begin
 equal[1] =b;
 equal[0] =!a;
end

always @(
equal[1] = !b;
endmodule
```

## Action

Avoid multiple writes on a variable. In the test case below, there are no multiple writes on any of the bits of variable equal.

```
module cg402(equal, a, b);
output var [1:0]equal;
input a,b;

always @(a,b)
begin
 equal[1] =b;
 equal[0] =!a;
end
endmodule
```

# CG404

## **@E: Assignment to constant type <c> is illegal**

SystemVerilog; a constant is assigned a value after it is declared. In the test case below, constant c is declared as equal to 10, but is subsequently assigned a value of 11 which causes the compiler to error out.

```
module cg404 (equal, a, b);
parameter size = 1;
output equal;
input [size-1:0] a, b;
const shortint c = 10;
```

```
assign c = 11;
assign equal = a == b;
endmodule
```

## Action

Do not reassign a value to an already declared constant. In the test case below, constant c is assigned a value of 10 only when it is declared and nowhere else.

```
module cg404 (equal, a, b);
parameter size = 1;
output equal;
input [size-1:0] a, b;
const shortint c = 10;

assign equal = a == b;
endmodule
```

# CG410

## **@E: Only one assignment is legal on <equal>**

SystemVerilog; the compiler identified multiple assignments on a variable declared using the var keyword. In the test case below, output port equal is declared as a variable using the var keyword but because of the multiple assignments on this variable, the compiler errors out.

```
module cg410(equal, a, b);
output var equal;
input a,b;

always @(a)
equal =a;

always @(\ b)
if(b)
equal = !b;

endmodule
```

## Action

Avoid multiple assignments on a variable. In the test case below, there are no multiple assignments on the variable equal.

```
module cg410(equal, a, b);
 output var equal;
 input a,b;

 always @(a,b)
 if(b)
 equal =!b;
 else
 equal =a;
endmodule
```

# CG412

### **@A: Treating ==? and !=? as == and != - possible simulation mismatch**

The compiler encountered the equality operators === or !== in Verilog. These are case equality operators. Equality operators == and != are logical equality operators. In case comparisons, the values x and z are strictly compared as values, not interpretations as unknown or high impedance, hence the result is never an unknown. In logical comparisons, if any operand contains an x or a z, the result is an unknown value (x). For example, if data1 = 4'b1xx0; and data2 = 4'b1xx0; then data1==data2 returns x while data== data2 returns value 1. In terms of synthesis, case equality operators are converted into logic equality operators. The following test case uses case equality operators which results in the above warning.

```
module compare(equal, a, b);
 parameter size = 1;
 output equal;
 input [size-1:0] a, b;
 wire equal;
 wire [size-1:0] c;
assign equal = a === b;
endmodule
```

## Action

Make sure that logic equality operators (`==`, `!=`) are used in designs for synthesis. Use of case equality operators can cause mismatches in RTL and post-synthesis simulations. To eliminate the warning in the above test case and to prevent possible RTL/post-synthesis simulation mismatches, replace the case equality operator with the equality operator as shown in the corrected test case below.

```
module compare(equal, a, b);
parameter size = 1;
output equal;
input [size-1:0] a, b;
wire equal;
wire [size-1:0] c;
assign equal = a == b;
endmodule
```

# CG413

## **@E: Assignment expressions are valid within procedural statements**

SystemVerilog; an assignment expression was encountered outside a procedural block. In the test case below, the assignment expression `out1++` is outside the procedural block which causes the compiler to error out.

```
module test (in,out1,out2,out3);
input in;
output out3;
output reg out1,out2;
var logic r1 ;

always @(*)
begin
 r1 =in;
 out1 = r1--;
 out2 = r1++;
end

assign out3 =out1++;
endmodule
```

## Action

Use assignment expressions only inside a procedural block as shown in the corrected test case below.

```
module test (in,out1,out2,out3);
 input in;
 output reg out1,out2,out3;
 var logic r1 ;

 always @(*)
 begin
 r1 =in;
 out1 = r1--;
 out2 = r1++;
 out3 =out1++;
 end
endmodule
```

# CG414

## @W: Ignoring macro: `line

The compiler encountered a `line macro (`line macros are not supported). In the test case below, a `line macro is used which results in the warning.

```
`line 3 d:/error_documentation/compile_error/cg180/cg180.v
module cg414(a,b,c);
 input wire a, b;
 output c;
 assign c= a & b;
endmodule
```

## Action

To avoid the warning during synthesis, comment out the macro as shown in the test case below (the macro later can be uncommented for simulation).

```
//`line 3 d:/error_documentation/compile_error/cg180/cg180.v
module cg414(a,b,c);
input wire a, b;
output c;
assign c= a & b;
endmodule
```

## CG417

### **@E: An inout port should be of type net**

A register type was used with an inout port. A port, by default, is a net. Only output ports can be optionally redeclared as registers. In the following test case, a reg declaration is associated with inout port c.

```
module comb (a,b,c);
input a,b;
inout reg c;
assign c= a & b;
endmodule
```

#### Action

Declare all inout ports as net-type. In the corrected test case below, the register type is removed from inout port c.

```
module comb(a,b,c);
input a,b;
inout c;
assign c= a & b;
endmodule
```

## CG418

### **@E: Range is illegal for type <int>**

SystemVerilog; a range is declared for a shortint, int, longint, or byte data type. In the test case below, output port equal is declared as type int, but because a range is provided, the compiler errors out.

```
module cg418(equal, a, b);
 output int [1:0] equal;
 input a,b;
 assign equal[1]= a;
 assign equal[0]= b;
endmodule
```

### Action

Either do not specify a range on for data types int, shortint, longint, or byte or use the bit or logic data type if the object requires a range. In the corrected test case below, output port equal is declared as bit.

```
module cg418(equal, a, b);
 output bit [1:0] equal;
 input a,b;
 assign equal[1]= a;
 assign equal[0]= b;
endmodule
```

## CG419

### **@E: port assignments allowed for only variable types.**

A port assignment on a non-variable type was encountered (only ports of a variable type can have a port assignment). In the two test cases below, the incorrect output port assignment causes the compiler to error out.

## Verilog Test Case

```
module test(out, in1, in2);
 output out=1'b0;
 input in1, in2;

 always@(in1,in2)
 out=in1 & in2;
endmodule
```

## SystemVerilog Test Case

```
module test (in,out);
 input [1:0]in;
 output out=2'b0;
 nand u1[1:0] (out,in[0],in[1]);
endmodule
```

## Action

Make port assignments only on variable types as shown in the following two corrected test cases.

## Verilog Test Case

```
module test(out, in1, in2);
 output reg out=1'b0;
 input in1, in2;

 always@(in1,in2)
 out=in1 & in2;
endmodule
```

## SystemVerilog Test Case

```
module test (in,out);
 input [7:0]in[1:0];
 output var logic [7:0] out =8'b0;
 nand u1[7:0] (out,in[0],in[1]);
endmodule
```

## CG421

### **@E: Expecting initial value for constant <c>**

SystemVerilog; a constant was not initialized during its declaration. In the test case below, constant c is declared, but because it is not initialized, the compiler to error out.

```
module cg421 (equal, a, b);
parameter size = 1;
output equal;
input [size-1:0] a, b;
const c ;
assign equal = a == b;
endmodule
```

#### Action

Initialize the constant when it is declared as shown in the corrected test case below.

```
module cg421 (equal, a, b);
parameter size = 1;
output equal;
input [size-1:0] a, b;
const c =10;
assign equal = a == b;
endmodule
```

## CG424

### **@W: Initial values on multi-dimensional variables not yet implemented.**

The compiler encountered the initialization of a multi-dimensional variable (initialization of multi-dimensional variables currently is not supported). In the test case below, 2-dimensional array [7:0]mem[15:0] is initialized which results in the warning.

```
module data_buffer (input clk, en, input [3:0] addr, raddr,
 input [7:0] data, output [7:0] q, q2);
 reg [7:0]mem[15:0]=0;
 parameter test_addr = 15;

 always@(posedge clk)
 if (en)
 mem[addr] <= data;
 assign q = mem[raddr];
 assign q2 = mem[test_addr];
endmodule
```

## Action

Do not initialize multi-dimensional variables. To avoid the warning, remove the initialization from 2-dimensional array [7:0]mem[15:0] as shown in the corrected test case below.

```
module data_buffer (input clk, en, input [3:0] addr, raddr,
 input [7:0] data, output [7:0] q, q2);
 reg [7:0]mem[15:0];
 parameter test_addr = 15;

 always@(posedge clk)
 if (en)
 mem[addr] <= data;
 assign q = mem[raddr];
 assign q2 = mem[test_addr];
endmodule
```

## CG425

### **@E: Assignment target <buffer> must be of type reg, genvar, or logic**

SystemVerilog; the target used in a procedural assignment statement is of a type other than reg, genvar, or logic. In the test case below, the wire buffer inside the always block is being used as a target which causes the compiler to error out.

```
module data_buffer(d_s,data,clk,buffer);
 input d_s;
 input [15:0] data;
 input clk;
 output [15:0] buffer ;

 always@(posedge(clk))
 begin
 if (d_s)
 begin
 buffer=data;
 end
 end
 endmodule
```

## Action

Use only targets of type `reg`, `genvar`, or `logic` in procedural assignment statements. In the test case below, the output `buffer` is declared as type `reg` which is a valid target inside the `always` block.

```
module data_buffer(d_s,data,clk,buffer);
 input d_s;
 input [15:0] data;
 input clk;
 output reg [15:0] buffer ;

 always@(posedge(clk))
 begin
 if (d_s)
 begin
 buffer=data;
 end
 end
 endmodule
```

## CG426

### @E: Assignment target <b> must be of type reg or genvar

The target used in a procedural assignment statement is of a type other than `reg` or `genvar`. In the test case below, wire `b` is being used inside the `always` block as a target which causes the compiler to error out.

```
module ccg426(a,b,clk);
 input a;
 input clk;
 output b;
 wire temp;
 assign temp= a;

 always@(posedge clk)
 b=temp;
endmodule
```

### Action

Use only targets of type `reg` or `genvar` in procedural assignment statements. In the test case below, output `b` is declared as type `reg` which is a valid target inside the `always` block.

```
module ccg426(a,b,clk);
 input a;
 input clk;
 output reg b;
 wire temp;
 assign temp= a;

 always@(posedge clk)
 b=temp;
endmodule
```

# CG428

## @E: Expecting =, <= or arithmetic assignment operator

The Verilog language requires assignments to be done using = (blocking) or <= (non-blocking) operators. In the following test case, the incomplete mem assignment in the always block causes the compiler to error out.

```
module singleportram(q, a2, d, we, clk, en);
 output [7:0] q;
 input [7:0] d;
 input[6:0] a2;
 input clk, we, en;
 reg [6:0] read_addr;
 reg[7:0] mem [127:0];
 wire temp;
 assign q = mem[read_addr];

 always @(posedge clk) begin
 if (we)
 mem[a2] d;
 read_addr <= a2;
 end
endmodule
```

## Action

This is a syntax error. To eliminate the error in the above test case, add the non-blocking operator to the mem assignment as shown in the corrected test case below.

```
module singleportram(q, a2, d, we, clk, en);
 output [7:0] q;
 input [7:0] d;
 input[6:0] a2;
 input clk, we, en;
 reg [6:0] read_addr;
 reg[7:0] mem [127:0];
 wire temp;
 assign q = mem[read_addr];
```

```
always @(posedge clk) begin
 if (we)
 mem[a2] <= d;
 read_addr <= a2;
 end
endmodule
```

## CG430

### **@E: Expecting one of (b, h, o or d). Found: <1>**

SystemVerilog; a 1, 0, x, or z was encountered following an apostrophe in a sized number which prevented the compiler from evaluating the number (the compiler expects a valid base-format character of a b, h, o, or d following the apostrophe). In the test case below, a 1 appears after the apostrophe in the sized number (1'1) which causes the error.

```
module cg430(a,b,c);
 input a, b;
 output c;
 assign c= a?b:1'1;
endmodule
```

### Action

Specify a valid base format when using sized numbers. In the corrected test case below, specifying a sized number of 1'b1 (base format b) corrects the error.

```
module cg430(a,b,c);
 input a, b;
 output c;
 assign c= a?b:1'b1;
endmodule
```

# CG431

## **@E: Expecting radix character (one of b, h, o or d) or unsized single bit literal (one of '1,'0,'x,'z)**

In a number specification, following an apostrophe, either:

- There was no character present
- There was a character other than a valid base format character (b, h, o, or d) or, in SystemVerilog, an unsized single-bit literal (1, 0, x, or z).

In the test case below, the character following the apostrophe in the number specification (1'k1) is not a valid base format (or single-bit literal character) which causes the error.

```
module cg431(a,b,c);
 input a, b;
 output c;
 assign c= a?b:1'k1;
endmodule
```

### Action

This is a syntax error in both Verilog and SystemVerilog. The valid format for specifying a sized number is:

*size' baseFormat number*

For unsized numbers:

- with no base format specified, defaults to decimal number (i.e., 2345)
- with a base format specified, the size is machine dependent (e.g., 'ha, 'b1, or 'o1)
- an unsized single-bit literal (1, 0, x, or z) can be used (SystemVerilog only)

In the corrected test case below, the sized number is specified properly as 1'b1 which eliminates the error.

```
module cg431(a,b,c);
 input a, b;
 output c;
 assign c= a?b:1'b1; // In both verilog and system verilog
//assign c= a?b:'b1; // In both verilog and system verilog
//assign c= a?b:'1; // In system verilog only
endmodule
```

## CG432

### **@E: Expecting port name or expression in gate instantiation**

A gate was instantiated without port mapping. In the code segment below, the and primitive does not contain port names within the instantiation which results in the error.

```
module tst(input a, b, output c);
 and u0();
endmodule
```

#### Action

Be sure to specify the port name or expression in the gate instantiation as shown in the code segment below.

```
module tst(input a, b, output c);
 and u0(c, a, b);
endmodule
```

## CG433

### **@E: Named connections are illegal for gates**

Named association found in gate instantiation. The Verilog language has multiple input built-in gates such as AND, NAND, NOR, OR, XOR, and XNOR. These gates can be instantiated by using the following syntax:

*multiple\_input\_gate\_type [instanceName] (output,input1,input2,input3...,inputN)*

In the test case below, named associations are used in association with Verilog built-in gates which causes the error.

```
module and_gate (a, b, c, test);
 output test;
 input c;
 input a, b;
 and al (.out1(test), .in1(a), .in2(b), .in3(c));
endmodule
```

### Action

This is a syntax error. Make sure that instantiations of Verilog built-in gates use only positional association. To eliminate the error in the above test case, remove the named associations from the and instantiation as shown in the corrected test case below.

```
module and_gate (a, b, c, test);
 output test;
 input c;
 input a, b;
 and al (test, a, b, c);
endmodule
```

## CG434

### @E: Unexpected port name <portName>

The above error occurs when instantiating a VHDL entity in Verilog using named association and the referenced VHDL entity includes duplicate port names.

### Action

Make sure that there are no duplicate port names in the referenced VHDL entity and that the corresponding VHDL file compiles without error.

## CG435

### **@E: Unexpected port map for port <portName>**

The above error occurs when instantiating a VHDL entity in Verilog using named association and the referenced VHDL entity includes duplicate port names.

#### Action

Make sure that there are no duplicate port names in the referenced VHDL entity and that the corresponding VHDL file compiles without error.

## CG440

### **@N: Allowing writes in multiple processes for multi-port RAM**

Code for a multi-port RAM was encountered. Generally, the compiler does not allow a write to the same variable in a multiple process. However, because multi-port RAM requires this type of code, the write operation is permitted. In the following test case, the compiler allows the assignment of variable mem to two different procedural blocks.

```
module ram(data0,data1,waddr0,waddr1,we0,we1,clk0,clk1,q0,q1);
parameter d_width = 8;
parameter addr_width = 8;
parameter mem_depth = 256;
input [d_width-1:0] data0, data1;
input [addr_width-1:0] waddr0, waddr1;
input we0, we1, clk0, clk1;
output [d_width-1:0] q0, q1;
reg [addr_width-1:0] reg_addr0, reg_addr1;
reg [d_width-1:0] mem [mem_depth-1:0]
/* synthesis syn_ramstyle="no_rw_check" */;
assign q0 = mem[reg_addr0];
assign q1 = mem[reg_addr1];
```

```
always @(posedge clk0)
begin
 reg_addr0 <= waddr0;
 if (we0)
 mem[waddr0] <= data0;
end

always @(posedge clk1)
begin
 reg_addr1 <= waddr1;
 if (we1)
 mem[waddr1] <= data1;
end

endmodule
```

## Action

Informative message; no user action required.

# CG445

## **@E: Parameter expression did not evaluate to a constant**

The parameter expression in a generated module instantiation did not evaluate to a constant when the generate loop was elaborated (evaluation to a constant is required to elaborate the generate loop into individual statements). In the test case below, the parameter expression is  $2^*i + y$ . However, because the value of integer  $y$  is not a run-time constant, the entire parameter expression does not evaluate to a constant which causes the error.

```
module adder8 (sum, c0, a, b, cin);
parameter k =5;
input [7:0]a, b;
output [7:0]sum;
output [0:0] c0;
input [0:0] cin;
assign {c0, sum} = a + b + cin;
endmodule
```

```

module adder_32bit (a, b, sum, cin_in, c0_out);
 input [31:0]a, b;
 output [31:0]sum;
 input cin_in;
 output c0_out;
 wire [4:0]c0;
 integer y=0;
 assign c0[0] = cin_in;

 generate
 genvar i;
 begin:u1
 for (i=0; i<=3; i = i + 1)
 begin :u
 adder8 #(.k(2*i+ y))add (sum [8*i+7:8*i], c0[i+1],
 a [8*i+7:8*i], b[8*i+7:8*i], c0[i]);
 end
 end
 endgenerate

 assign c0_out = c0[4];

endmodule

```

## Action

Use only expressions that evaluate to constants in parameter expressions within a generated module instantiation statement. In the corrected test case below, identifier y is declared as a parameter and elaborated as a run-time constant.

```

module adder8 (sum, c0, a, b, cin);
 parameter k =5;
 input [7:0]a, b;
 output [7:0]sum;
 output [0:0] c0;
 input [0:0] cin;
 assign {c0, sum} = a + b + cin;
endmodule

module adder_32bit (a, b, sum, cin_in, c0_out);
 parameter y=0;
 input [31:0]a, b;
 output [31:0]sum;
 input cin_in;
 output c0_out;
 wire [4:0]c0;

```

```

assign c0[0] = cin_in;
generate
genvar i;
begin:ul
 for (i=0; i<=3; i = i + 1)
 begin :u
 adder8 #(.k(2*i + y))add (sum [8*i+7:8*i], c0[i+1],
 a [8*i+7:8*i], b[8*i+7:8*i], c0[i]);
 end
 end
endgenerate

assign c0_out = c0[4];

endmodule

```

## CG446

**@E: enum variable <temp> can only be assigned to one of its members in the declared set**

SystemVerilog; the value assigned to an enum variable does not belong to the declared set of enum variables. In the following test case, enum variable temp can accept only the values set, reset, or himp. Accordingly, when enum variable temp is assigned the value a2 (which does not belong to the declared set of temp), the compiler errors out.

```

module test (a,b,c,q);
 input a,b,c;
 output q;
 enum logic {set=1'b1, reset=1'b0, himp=1'bz} temp;
 enum {a[3]=4} templ;

 always @(a or b)
 if (a)
 temp=a2;
 else if (b)
 temp= reset;
 else
 temp=himp;

 assign q= c ? temp : 1'bz;
endmodule

```

## Action

Make sure that all enum variables are assigned using only elements of their declared sets. In the corrected test case below, enum variable temp is assigned the values set and reset which belong to its declared set.

```
module test (a,b,c,q);
 input a,b,c;
 output q;
 enum logic {set=1'b1, reset=1'b0, himp=1'bz} temp;
 enum {a[3]=4} temp1;

 always @(a or b)
 if (a)
 temp=set;
 else if (b)
 temp= reset;
 else
 temp=himp;

 assign q= c ? temp : 1'bz;
endmodule
```

## CG447

### **@E: enum variable <temp1> can only be assigned to an identical type enum**

SystemVerilog; an enum variable was assigned an enum variable that was not of the identical type (i.e., declared together separated by comma). In the test case below, enum variable temp1 is assigned enum variable temp which, because it is not the identical type, causes error.

```
module test (a,b,c,q);
 input a,b,c;
 output q;
 enum logic {set=1'b1, reset=1'b0, himp=1'bz} temp;
 enum bit {a[2]} temp1;
 assign temp1= temp;
```

```
always @(a or b)
if (a)
 temp=set;
else if (b)
 temp= reset;
else
 temp=himp;

assign q= c ? temp : temp1;
endmodule
```

## Action

Assign an enum variable of similar type or use only elements of its declared set as shown below where the enum variable temp1 is assigned one of its elements a1.

```
module test (a,b,c,q);
input a,b,c;
output q;
enum logic {set=1'b1, reset=1'b0, himp=1'bz} temp;
enum bit {a[2]} temp1;
assign temp1=a1;

always @(a or b)
if (a)
 temp=set;
else if (b)
 temp= reset;
else
 temp=himp;

assign q= c ? temp : temp1;
endmodule
```

# CG448

## @E: Invalid assignment to enum variable <my>

SystemVerilog; an enum variable was assigned to another variable that was not another enum variable of identical type or any of the elements of its declared set. In the test case below, enum variable temp is assigned the variable k which is neither one of the elements of the declared set (set, reset, or himp) nor another enum variable of identical type.

```
module test (a,b,c,q);
 input a,b,c;
 output q;
 parameter k=1;
 enum logic {set=1'b1, reset=1'b0, himp=1'bz} temp,temp1;

 always @(a or b)
 if (a)
 temp=k ;
 else if (b)
 temp= reset;
 else
 temp=himp;

 assign q= c ? temp : temp1;
endmodule
```

### Action

Assign an enum variable with either one of the elements of the declared set or another enum variable of identical type. In the given context, the enum variable can take any of the three elements set, reset, or himp or the other identical type enum variable temp1 as shown in the corrected test case below.

```
module test (a,b,c,q);
 input a,b,c;
 output q;
 parameter k=1;
 enum logic {set=1'b1, reset=1'b0, himp=1'bz} temp,temp1;
```

```
always @(a or b)
if (a)
 temp=set; // or temp=temp1;
else if (b)
 temp= reset;
else
 temp=himp;

assign q= c ? temp : temp1;
endmodule
```

## CG449

### **@E: enum variable assignment <temp> did not evaluate to a valid value**

SystemVerilog; an enum variable was assigned a value that was not another enum variable of identical type or any of the elements of its declared set. In the test case below, enum variable temp is assigned the value 1'b1 which is neither one of the elements of the declared set (set, reset, or himp) nor another enum variable of identical type.

```
module test (a,b,c,q);
input a,b,c;
output q;
parameter k=1;
enum logic {set=1'b1, reset=1'b0, himp=1'bz} temp,temp1;

always @(a or b)
if (a)
 temp=1'b1 ;
else if (b)
 temp= reset;
else
 temp=himp;

assign q= c ? temp : temp1;
endmodule
```

## Action

Assign an enum variable with either one of the elements of the declared set or another enum variable of identical type. In the given context, the enum variable can take any of the three elements set, reset, or himp or the other identical type enum variable temp1 as shown in the corrected test case below.

```
module test (a,b,c,q);
 input a,b,c;
 output q;
 parameter k=1;
 enum logic {set=1'b1, reset=1'b0, himp=1'bz} temp,temp1;

 always @(a or b)
 if (a)
 temp=set; // or temp=temp1;
 else if (b)
 temp= reset;
 else
 temp=himp;

 assign q= c ? temp : temp1;
endmodule
```

## CG450

### **@W: Size is out of range for this type, truncating to 32 bits**

SystemVerilog; a variable of type int is assigned a value of size that is greater than 32 bits (the compiler truncates any value greater than 32 bits). In the test case below, variable out is of type int and, because it is assigned a value of 64'b0, the value is truncated to 32 bits as indicated by the warning.

```
module test(out,sel,in);
 output int out;
 input sel;
 input int in;
 assign out= sel ? in : 64'b0;
endmodule
```

## Action

Make sure that variables of type int are not assigned a value that is greater than 32 bits in size as shown in the corrected test case below.

```
module test(out,sel,in);
 output int out;
 input sel;
 input int in;
 assign out= sel ? in : 32'b0;
endmodule
```

# CG451

### @E: Need positive integral number

SystemVerilog; the elements of an enumeration type were declared in one of the following four formats: *name[K]*, *name[K]=c*, *name[N:M]*, or *name[N:M]=c* and either *N* or *M* was a negative integral number or *K* was not a positive integral number. In the test case below, the index -2 for enumeration element a is a negative number which results in the error.

```
module test (a,b,c,q);
 input a,b,c;
 output q;
 enum logic {a[-2],himp=1'bz} temp;

 always @(a or b)
 if (a)
 temp=a1;
 else if (b)
 temp= a0;
 else
 temp=himp;

 assign q= c ? temp : 1'bz;
endmodule
```

## Action

Use only non-negative integral numbers for the  $[N:M]$  type of indexing and use only positive integral numbers for the  $[K]$  type of indexing as shown in the corrected test case below where the index used is the positive integer 2.

```
module test (a,b,c,q);
 input a,b,c;
 output q;
 enum logic {a[2],himp=1'bz} temp;

 always @(a or b)
 if (a)
 temp=a1;
 else if (b)
 temp= a0;
 else
 temp=himp;

 assign q= c ? temp : 1'bz;
endmodule
```

## CG454

### **@E: Type mismatch between value assignment and enum name declaration**

SystemVerilog; the type of the value assigned to any of the elements of an enumerated type differs from its base type. In the test case below, the base type is bit which is a 2-state, SystemVerilog data type. However, because enumeration element himp is assigned the value 1'bz which is not a 2-state type, the compiler errors out.

```
module test (a,b,c,q);
 input a,b,c;
 output q;
 enum bit {a[2],himp=1'bz} temp;
```

```
always @(a or b)
if (a)
 temp=a1;
else if (b)
 temp= a0;
else
 temp=himp;

assign q= c ? temp : 1'bz;
endmodule
```

## Action

Make sure that the value assigned to any element of an enumerated type has the same type as that of its base type. In the corrected test case below, enumerated element himp is assigned the value 1'bz to make the base type logic.

```
module test (a,b,c,q);
input a,b,c;
output q;
enum logic {a[2],himp=1'bz} temp;

always @(a or b)
if (a)
 temp=a1;
else if (b)
 temp= a0;
else
 temp=himp;

assign q= c ? temp : 1'bz;
endmodule
```

## CG456

### @E: Enum value on <himp> collides with previously assigned value on enum <set>

SystemVerilog; the value assigned to any two elements of an enumerated type is the same (enumerated types require all elements to be unique). In the test case below, element himp is assigned the value 1'b1. However, because the value 1'b1 is already assigned to element set, the compiler errors out.

```
module test (a,b,c,q);
 input a,b,c;
 output q;
 enum logic {set=1'b1,reset=1'b0,himp=1'b1} temp;

 always @(a or b)
 if (a)
 temp=set;
 else if (b)
 temp= reset;
 else
 temp=himp;

 assign q= c ? temp : 1'bz;
endmodule
```

### Action

Make sure that the values assigned to all elements of an enumerated type are unique as shown in the corrected test case below.

```
module test (a,b,c,q);
 input a,b,c;
 output q;
 enum logic {set=1'b1,reset=1'b0,himp=1'b2} temp;

 always @(a or b)
 if (a)
 temp=set;
 else if (b)
 temp= reset;
 else
 temp=himp;

 assign q= c ? temp : 1'bz;
endmodule
```

# CG457

## @E: Cannot synthesize expression

The expression cannot be synthesized. In the following test case, the indexing expression is specified as a real number which results in the error because the expression required to index the vector must evaluate to an integer.

```
module test (
 input[3:0] i, j,
 input clk,
 output[3:0] out, out2);
reg[3:0] temp;

always@(posedge clk)
begin
 temp = j;
end

assign out2 = temp;
and (out[0], i[3.0], j[0]); // 3.0 is not an integer type
endmodule
```

## Action

Make sure that all expressions evaluate to their expected types. In the test case below, signal i is correctly indexed to an integer value (3).

```
module test (
 input[3:0] i, j,
 input clk,
 output[3:0] out, out2);
reg[3:0] temp;

always@(posedge clk)
begin
 temp = j;
end

assign out2 = temp;
and (out[0], i[3], j[0]);
endmodule
```

# CG461

## @E: Expecting constant greater than 0 to generate sequence enum names

SystemVerilog; elements of an enumeration type were declared in format *name[N]* or *name[N]=c* and *N* was not a positive integral number.

The compiler generates *N* named constants in the sequence *name0*, *name1*, ..., *nameN-1*; if *N* were 0, *N-1* would be negative. In the test case below, an index of 0 is used in enumeration element *a* which causes the compiler to error out.

```
module test (a,b,c,q);
 input a,b,c;
 output q;
 enum logic {a[0],himp=1'bz} temp;

 always @(~(a|b))
 if (a)
 temp=a1;
 else if (b)
 temp= a0;
 else
 temp=himp;

 assign q= c ? temp : 1'bz;
endmodule
```

### Action

Use only positive integral numbers as indices of enumeration elements as shown in the corrected test case below where a positive integer of 2 is used as the index.

```
module test (a,b,c,q);
 input a,b,c;
 output q;
 enum logic {a[2],himp=1'bz} temp;
```

```
always @(a or b)
if (a)
 temp=a1;
else if (b)
 temp= a0;
else
 temp=himp;

assign q= c ? temp : 1'bz;
endmodule
```

## CG462

### @E: Assignment target <a0> cannot be a enum constant

SystemVerilog; an enum constant (element of an enumerated type) was encountered as an assignment target. In the test case below, enum constant a0 is used as an assignment target which causes the compiler to error out.

```
module test (a,b,c,q);
input a,b,c;
output q;
enum logic {a[2],himp=1'bz} temp;
assign a0= 1'b0;

always @(a or b)
if (a)
 temp=a1;
else if (b)
 temp= a0;
else
 temp=himp;

assign q= c ? temp : 1'bz;
endmodule
```

Use any of the enum constants (elements of enumerated type) as an assignment target as shown in the corrected test case below.

```
module test (a,b,c,q);
input a,b,c;
output q;
enum logic {a[2],himp=1'bz} temp;
```

```
always @(a or b)
if (a)
 temp=a1;
else if (b)
 temp= a0;
else
 temp=himp;

assign q= c ? temp : 1'bz;
endmodule
```

## CG465

### @E: Expecting { for enum type

SystemVerilog; an enumerated list did not begin with a curly brace ( { ). The compiler identifies the presence of a base type (if declared) and then expects an opening curly brace to identify the start of the enumerated list. In the test case below, the opening curly brace is missing immediately following the base type keyword logic which causes the error.

```
module test (a,b,c,q);
input a,b,c;
output q;
enum logic a[2],himp=1'bz} temp;

always @(a or b)
if (a)
 temp=a1;
else if (b)
 temp= a0;
else
 temp=himp;

assign q= c ? temp : 1'bz;
endmodule
```

### Action

Make sure that enumerated lists always begin with a curly brace as shown in the corrected test case below.

```
module test (a,b,c,q);
 input a,b,c;
 output q;
 enum logic {a[2],himp=1'bz} temp;

 always @(a or b)
 if (a)
 temp=a1;
 else if (b)
 temp= a0;
 else
 temp=himp;

 assign q= c ? temp : 1'bz;
endmodule
```

## CG466

### @E: Expecting enum member name

SystemVerilog; an enum member name was not in the enclosed enumerated list or the enum number name in the list was not a proper variable. In the test case below, because the enclosed enum member name 1'b1 is not a proper variable, the compiler errors out.

```
module test (a,b,c,q);
 input a,b,c;
 output q;
 enum logic {1'b1,himp=1'bz} temp;

 always @(a or b)
 if (a)
 temp=a1;
 else if (b)
 temp= a0;
 else
 temp=himp;

 assign q= c ? temp : 1'bz;
endmodule
```

## Action

Make sure that the enum member names in the enclosed enumerated list are properly specified as shown in the corrected test case below.

```
module test (a,b,c,q);
 input a,b,c;
 output q;
 enum logic {a[2],himp=1'bz} temp;

 always @(a or b)
 if (a)
 temp=a1;
 else if (b)
 temp= a0;
 else
 temp=himp;

 assign q= c ? temp : 1'bz;
endmodule
```

## CG467

### @E: Expecting enum name

SystemVerilog; an enum variable name was missing from the enum declaration. In the following test case, there is no enum variable specified in the second enum declaration which causes the error.

```
module test (a,b,c,q);
 input a,b,c;
 output q;
 enum logic {set=1'b1, reset=1'b0, himp=1'bz} temp;
 enum {a[3]=4} ;

 always @(a or b)
 if (a)
 temp=set;
 else if (b)
 temp= reset;
 else
 temp=himp;
```

```
assign q= c ? temp : 1'bz;
endmodule
```

## Action

Make sure that an enum variable is always specified in the enum declaration. In the corrected test case below, enum variable name temp1 has been added to the second enum declaration.

```
module test (a,b,c,q);
 input a,b,c;
 output q;
 enum logic {set=1'b1, reset=1'b0, himp=1'bz} temp;
 enum {a[3]=4} temp1 ;

 always @(* a or b)
 if (a)
 temp=set;
 else if (b)
 temp= reset;
 else
 temp=himp;

 assign q= c ? temp: 1'bz;
endmodule
```

# CG469

## **@E: ++/-- not allowed on enum type <temp>**

SystemVerilog; an increment or decrement operator was encountered on an enumerated type (enum variables only can take values from the declared set of elements). In the following test case, the increment statement temp++ causes the error.

```
module test (a,b,c,q);
 input a,b,c;
 output q;
 enum logic {set=1'b1, reset=1'b0, himp=1'bz} temp;
 enum {a[3]=4} temp1 ;
```

```
always @(a or b)
if (a)
 temp=set;
else if (b)
 temp= reset ;
else
 temp++;
assign q= c ? temp : 1'bz;
endmodule
```

## Action

Avoid using increment and decrement operators on enumerated type variables. When making assignments to enum type variables, the value can only belong to the declared set of values as shown in the corrected test case below (enum variable temp is assigned only values from the declared set of set, reset, or himp).

```
module test (a,b,c,q);
input a,b,c;
output q;
enum logic {set=1'b1, reset=1'b0, himp=1'bz} temp;
enum {a[3]=4} templ1 ;

always @(a or b)
if (a)
 temp=set;
else if (b)
 temp= reset ;
else
 temp=himp;

assign q= c ? temp : 1'bz;
endmodule
```

# CG471

## @W: Ignoring enum <temp> in sensitivity list.

SystemVerilog; an enum variable was encountered in the sensitivity list. In the test case below, enum variable temp is incorrectly included in the sensitivity list of the always block and is subsequently ignored by the compiler as indicated by the above message.

```
module test (a,b,c,q);
 input a,b,c;
 output reg q;
 enum logic {set=1'b1, reset=1'b0, himp=1'bz} temp;

 always @(a or b)
 if (a)
 temp=set;
 else if (b)
 temp= reset;
 else
 temp=himp;

 always @(`c or temp)
 if (c)
 q= temp;
 else
 q =1'bz;

endmodule
```

## Action

Make sure that enum variables are not included in the sensitivity list.

```
module test (a,b,c,q);
 input a,b,c;
 output reg q;
 enum logic {set=1'b1, reset=1'b0, himp=1'bz} temp;
```

```

always @(a or b)
if (a)
 temp=set;
else if (b)
 temp= reset;
else
 temp=himp;

always @(c)
if (c)
 q= temp;
else
 q =1'bz;

endmodule

```

## CG476

### @E: Multiple procedural block writes on <temp>

SystemVerilog; an enum variable was assigned conflicting values from different always blocks. In the test case below, enum variable temp is assigned the value set from the first always block and is then assigned the value reset from the second always block. The conflicting assignments cause the error.

```

module test (a,b,c,q);
input a,b,c;
output q;
enum logic {set=1'b1, reset=1'b0, himp=1'bz} temp;

always@(a)
 temp= set;

always @ (b)
 temp= reset;

assign q= c ? temp : 1'bz;
endmodule

```

### Action

Do not use an enum variable as an assignment target in more than one block.

```
module test (a,b,c,q);
 input a,b,c;
 output q;
 enum logic {set=1'b1, reset=1'b0, himp=1'bz} temp;

 always @(a or b)
 if (a)
 temp=set;
 else if (b)
 temp= reset;
 else
 temp=himp;

 assign q= c ? temp : 1'bz;
endmodule
```

## CG478

### **@E: Expression size is either 0 or could not be determined**

The compiler could not successfully evaluate an expression. This is a serious error that generally indicates a problem within the compiler itself and should be reported to Synopsys with all possible subordinating information such as the project and data files and software version.

#### Action

Beyond reporting this error to Synopsys, use the source location information included with the message to isolate the general area of the problematic code and try experimenting with equivalent code structures.

## CHAPTER 15

# CG Messages 479 – 1327

---

## CG479

**@E: <a> is already declared in this scope.**

SystemVerilog; the name used for an enum variable has been previously declared.

### Test Case 1

In this test, enum variable name a is already declared as an input port causing the compiler to error out.

```
module test (a,b,c,q);
 input a,b,c;
 output q;
 enum logic {set=1'b1, reset=1'b0, himp=1'bz} a;

 always @(~(a or b))
 if (a)
 temp=set;
 else if (b)
 temp= reset;
 else
 temp=himp;
```

```
assign q= c ? temp : 1'bz;
endmodule
```

## Action

Use only undeclared variable names as enum variable names. In the test case below, undeclared variable name temp is used for the enum variable name.

```
module test (a,b,c,q);
 input a,b,c;
 output q;
 enum logic {set=1'b1, reset=1'b0, himp=1'bz} temp ;

 always @(a or b)
 if (a)
 temp=set;
 else if (b)
 temp= reset;
 else
 temp=himp;

 assign q= c ? temp : 1'bz;
endmodule
```

## Test Case 2

In this test, enum elements RD and WR are defined in both data types rd\_wr\_t and wr\_rd\_t causing the compiler to error out.

```
//File1.v pkg
typedef enum logic {
 RD = 1'b0,
 WR = 1'b1
} rd_wr_t;

//File2.v pkg
typedef enum logic {
 RD = 1'b0,
 WR = 1'b1
} wr_rd_t;
```

```
//File3.v top
module top (
 input clk,
 input rst,
 input rd_wr_t d1,
 input wr_rd_t d2,
 output rd_wr_t q1,
 output wr_rd_t q2);

always_ff@(posedge clk)
begin
 q1 <= d1;
 q2 <= d2;
end
endmodule
```

## Action

Use unique names for the enum elements in each data type. In the corrected test case, elements of enum wr\_rd\_t are made unique from elements of enum rd\_wr\_t.

```
//File1.v pkg
typedef enum logic {
 RD1 = 1'b0,
 WR1 = 1'b1
} rd_wr_t;

//File2.v pkg
typedef enum logic {
 RD2 = 1'b0,
 WR2 = 1'b1
} wr_rd_t;

//File3.v top
module top (
 input clk,
 input rst,
 input rd_wr_t d1,
 input wr_rd_t d2,
 output rd_wr_t q1,
 output wr_rd_t q2);
```

```
always_ff@(posedge clk)
begin
 q1 <= d1;
 q2 <= d2;
end
endmodule
```

## CG483

### @E: expecting identifier after .

SystemVerilog; an identifier is missing following a dot separator (for example, a struct member is identified as *structName.memberName*). In the test case below, the missing identifier following foo in the always block causes the compiler to error out.

```
module test (in1,in2,clk, rst ,outa, outb);
input clk,rst;
input [7:0] in1,in2 ;
output [7:0] outa,outb;

struct {
 byte a,b;
} foo;

always@(posedge clk or posedge rst)
begin
if (rst)
begin
 foo.a = 0;
 foo.b = 0;
end
else
begin
 foo. = in1;
 foo.b = in2;
end
end

assign outa = foo.a;
assign outb = foo.b;
endmodule
```

## Action

Provide a proper identifier after the period separator. In the corrected test case below, identifier a is added to the end of the foo. string to correctly reference struct member foo.a.

```
module test (in1,in2,clk, rst ,outa, outb);
 input clk,rst;
 input [7:0] in1,in2 ;
 output [7:0] outa,outb;

 struct {
 byte a,b;
 } foo;

 always@(posedge clk or posedge rst)
 begin
 if (rst)
 begin
 foo.a = 0;
 foo.b = 0;
 end
 else
 begin
 foo.a = in1;
 foo.b = in2;
 end
 end

 assign outa = foo.a;
 assign outb = foo.b;
endmodule
```

## CG488

### @E: bad net type

SystemVerilog; a struct explicitly declared as type wire was encountered and one of its members was of a type that is not compatible with type net. In the test case below, struct c is declared as type wire which, when member g is declared as type bit (an invalid net type), causes the compiler to error out.

```
module test (in1,in2,clk, rst ,outa,outb,outc);
input clk,rst;
input [7:0] in1,in2 ;
output reg [7:0] outa,outb;
output [7:0] outc;

struct {
 byte a,b;
} foo;

wire struct {
 bit [7:0] g ;
} c;

assign c.g = foo.a & foo.b;

always@(posedge clk or posedge rst)
begin
if (rst)
begin
 foo.a = 0;
 foo.b = 0;
end
else
begin
 foo.a= in1;
 foo.b = in2;
end
end

assign outc = c.g;

always@(posedge clk)
begin
 outa = foo.a;
 outb = foo.b;
end

endmodule
```

## Action

When a struct is declared to be of type `wire`, all members of that struct must be declared with a type that is compatible with type `net` to avoid the above error. In the corrected test case below, the struct of type `wire` is declared with the compatible 4-state data type `logic`.

```
module test (in1,in2,clk, rst ,outa,outb,outc);
 input clk,rst;
 input [7:0] in1,in2 ;
 output reg [7:0] outa,outb;
 output [7:0] outc;

 struct {
 byte a,b;
 } foo;

 wire struct {
 logic [7:0] g ;
 } c;

 assign c.g = foo.a & foo.b;

 always@(posedge clk or posedge rst)
 begin
 if (rst)
 begin
 foo.a = 0;
 foo.b = 0;
 end
 else
 begin
 foo.a= in1;
 foo.b = in2;
 end
 end

 assign outc = c.g;

 always@(posedge clk)
 begin
 outa = foo.a;
 outb = foo.b;
 end

endmodule
```

# CG494

## @E: Expecting struct name

SystemVerilog; the name of the declared struct cannot be located. In the test case below, the struct name in the struct declaration is missing which causes the compiler to error out.

```
module test (in1,in2,clk, rst, outa, outb);
 input clk,rst;
 input [7:0] in1,in2 ;
 output [7:0] outa,outb;

 struct {
 byte a,b;
 } ;

 always@(posedge clk or posedge rst)
 begin
 if (rst)
 begin
 foo.a = 0;
 foo.b = 0;
 end
 else
 begin
 foo.a = in1;
 foo.b = in2;
 end
 end

 assign outa = foo.a;
 assign outb = foo.b;
endmodule
```

## Action

Provide a proper struct name as shown in the corrected test case below where the name of the struct is foo.

```
module test (in1,in2,clk, rst, outa, outb);
 input clk,rst;
 input [7:0] in1,in2 ;
 output [7:0] outa,outb;
```

```

struct {
 byte a,b;
} foo;

always@(posedge clk or posedge rst)
begin
 if (rst)
 begin
 foo.a = 0;
 foo.b = 0;
 end
 else
 begin
 foo.a = in1;
 foo.b = in2;
 end
end

assign outa = foo.a;
assign outb = foo.b;
endmodule

```

## CG495

### **@E: datatype reg is illegal on nets**

A lexical restriction with net and port declarations prohibits the Verilog net type keyword `wire` from being followed by `reg`. In the test case below, the `wire` declaration is followed by `reg` which causes the compiler to error out.

```

module my_and (c,a,b);
 input a,b;
 output c;
 wire reg temp;
 assign temp = !b;
 assign c= a & temp;
endmodule

```

### Action

Avoid using the `reg` keyword in `wire` declarations.

```
module my_and (c,a,b);
 input a,b;
 output c;
 wire temp;
 assign temp =!b;
 assign c= a & temp;
endmodule
```

## CG498

### @E: Nested structs not yet implemented

SystemVerilog; the compiler encountered a nested struct (nested structs currently are not supported). In the test case below, the nested struct causes the compiler to error out.

```
module test (in1,in2,clk, rst ,outa, outb);
 input clk,rst;
 input [7:0] in1,in2 ;
 output [7:0] outa,outb;

 struct {
 byte a,b;
 struct {
 byte c,d;
 } fool;
 } foo;
}

always@(posedge clk or posedge rst)
begin
 if (rst)
 begin
 foo.a = 0;
 foo.b = 0;
 end
 else
 begin
 foo.a = in1;
 foo.b = in2;
 end
end
```

```
assign outa = foo.a;
assign outb = foo.b;
endmodule
```

## Action

Avoid the use of nested structs as shown in the corrected test case below.

```
module test (in1,in2,clk, rst ,outa, outb);
input clk,rst;
input [7:0] in1,in2 ;
output [7:0] outa,outb;

struct {
 byte a,b;
} foo;

always@(posedge clk or posedge rst)
begin
 if (rst)
 begin
 foo.a = 0;
 foo.b = 0;
 end
 else
 begin
 foo.a = in1;
 foo.b = in2;
 end
end

assign outa = foo.a;
assign outb = foo.b;
endmodule
```

# CG499

### **@E: signed/unsigned is not allowed on unpacked struct.**

SystemVerilog; An unpacked struct declared as signed or unsigned was encountered. In the following test case, unpacked struct foo is declared as signed which causes the compiler to error out.

```
module test (in1,in2,clk, rst ,outa,outb);
 input clk,rst;
 input [7:0] in1,in2 ;
 output reg [7:0] outa,outb;

 struct signed {
 byte a,b;
 } foo;

 always@(posedge clk or posedge rst)
begin
 if (rst)
 begin
 foo.a = 0;
 foo.b = 0;
 end
 else
 begin
 foo.a= in1;
 foo.b = in2;
 end
end

always@(posedge clk)
begin
 outa = foo.a;
 outb = foo.b;
end

endmodule
```

## Action

Either declare the struct as packed (to allow it to be declared signed/unsigned) or do not declare an unpacked struct as signed/unsigned. In the corrected test case below, the struct is defined as packed which allows it also to be signed.

```
module test (in1,in2,clk, rst ,outa,outb);
 input clk,rst;
 input [7:0] in1,in2 ;
 output reg [7:0] outa,outb;

 struct packed signed {
 byte a,b;
 } foo;
```

```
always@(posedge clk or posedge rst)
begin
 if (rst)
 begin
 foo.a = 0;
 foo.b = 0;
 end
 else
 begin
 foo.a= in1;
 foo.b = in2;
 end
end

always@(posedge clk)
begin
 outa = foo.a;
 outb = foo.b;
end
endmodule
```

## CG500

### **@E: unknown variable declaration type**

SystemVerilog; the compiler encountered an unknown variable declaration type within a struct. In the test case below, variables a and b are declared wire data types which results in the error.

```
module test (in1,in2,clk, rst ,outa,outb);
input clk,rst;
input [7:0] in1,in2 ;
output reg [7:0] outa,outb;

struct {
 wire a,b;
} foo;
```

```

always@(posedge clk or posedge rst)
begin
 if (rst)
 begin
 foo.a = 0;
 foo.b = 0;
 end
 else
 begin
 foo.a= in1;
 foo.b = in2;
 end
 end
end

always@(posedge clk)
begin
 outa = foo.a;
 outb = foo.b;
end

endmodule

```

## Action

Use a proper data type for the declaration of any variable. In the corrected test case below, variables a and b are declared of type byte which is an acceptable type within a struct.

```

module test (in1,in2,clk, rst ,outa,outb);
 input clk,rst;
 input [7:0] in1,in2 ;
 output reg [7:0] outa,outb;

 struct {
 byte a,b;
 } foo;

 always@(posedge clk or posedge rst)
 begin
 if (rst)
 begin
 foo.a = 0;
 foo.b = 0;
 end
 else

```

```
begin
 foo.a= in1;
 foo.b = in2;
end
end

always@(posedge clk)
begin
 outa = foo.a;
 outb = foo.b;
end
endmodule
```

## CG501

### @E: Expecting delimiter: , or ; or )

A delimiter was missing from a variable declaration. In the following test case, the missing semicolon (;) delimiter at the end of wire declaration t1 causes the compiler to error out.

```
module my_and (c,a,b);
input a,b;
output c;
wire t1
assign t1 = !b;
assign c= a & t1;
endmodule
```

### Action

Place appropriate delimiters where required.

```
module my_and (c,a,b);
input a,b;
output c;
wire t1;
assign t1 = !b;
assign c= a & t1;
endmodule
```

# CG503

## @W: priority if not yet implemented - treating as regular if

SystemVerilog; a priority if statement was encountered (priority if statements currently are not supported). In the test case below, the presence of the priority if statement results in the above warning.

```
module test (sel,a,b,c,d,out);
 input a,b,c,d;
 input [1:0] sel;
 output reg out;

 always@(a,b,c,d,sel)
 begin
 priority if(sel==2'b00) out = a;
 else if (sel==2'b01) out=b;
 else if (sel ==2'b10) out =c;
 else if (sel==2'b11) out=d;
 end
 endmodule
```

## Action

Either avoid using the priority if statement or comment out the priority string as shown in the corrected test case below.

```
module test (sel,a,b,c,d,out);
 input a,b,c,d;
 input [1:0] sel;
 output reg out;

 always@(a,b,c,d,sel)
 begin
 /* priority */ if(sel==2'b00) out = a;
 else if (sel==2'b01) out=b;
 else if (sel ==2'b10) out =c;
 else if (sel==2'b11) out=d;
 end
 endmodule
```

## CG504

### @W: unique if not yet implemented - treating as regular if

SystemVerilog; a unique if statement was encountered (unique if statements currently are not supported). In the test case below, the presence of a unique if statement results in the above warning.

```
module test (sel,a,b,c,d,out);
 input a,b,c,d;
 input [1:0] sel;
 output reg out;

 always@(a,b,c,d,sel)
 begin
 unique if(sel==2'b00) out = a;
 else if (sel==2'b01) out=b;
 else if (sel ==2'b10) out =c;
 else if (sel==2'b11) out=d;
 end
 endmodule
```

### Action

Either avoid using the unique if statement or comment out the unique string as shown in the corrected test case below.

```
module test (sel,a,b,c,d,out);
 input a,b,c,d;
 input [1:0] sel;
 output reg out;

 always@(a,b,c,d,sel)
 begin
 /* unique */ if(sel==2'b00) out = a;
 else if (sel==2'b01) out=b;
 else if (sel ==2'b10) out =c;
 else if (sel==2'b11) out=d;
 end
 endmodule
```

# CG507

## @E: Target of break is not an enclosing loop

SystemVerilog; a break statement was encountered outside of a loop (break statements work only within loops such as for, while, and do while). In the test case below, the break statement occurs outside of the loop which results in the error.

```
module test(in1,in2,out);
 input [7:0] in1,in2 ;
 output reg [7:0] out ;
 integer i;

 always @ (in1,in2)
 begin
 if (in1 == 0) break;
 i = -1;
 do
 begin
 i = i+1;
 out[i] = in1[i] & in2[i];
 end
 end

 while (i < 8);
 out[6] = in1[6] | in2[6];
 out[7] = in1[7] ^ in2[6];
 end
endmodule
```

### Action

Make sure that all break statements are used only within loops. In the corrected test case below, the break statement appears only within a loop.

```
module test(in1,in2,out);
 input [7:0] in1,in2 ;
 output reg [7:0] out ;
 integer i;

 always @ (in1,in2)
 begin
 i = -1;
 do
 begin
```

```
i = i+1;
if (i == 6) break;
out[i] = in1[i] & in2[i];
end
while (i < 8);
 out[6] = in1[6] | in2[6];
 out[7] = in1[7] ^ in2[7];
end
endmodule
```

## CG508

### @E: Target of continue is not an enclosing loop

SystemVerilog; a continue statement appears outside of a loop (the SystemVerilog continue construct works only within loops such as for, while, and do while). In the test case below, the continue statement is outside of the loop which causes the error.

```
module test(in1,in2,out);
input [7:0] in1,in2 ;
output reg [7:0] out ;
integer i;

always @ (in1,in2)
begin
if (in1 == 0) continue;
i = -1;
do
begin
 i = i+1;
 out[i] = in1[i] & in2[i];
end
while (i < 8);
 out[6] = in1[6] | in2[6];
end
endmodule
```

## Action

Make sure that all continue statements are used only within loops. In the corrected test case below, the continue statement appears within the loop.

```
module test(in1,in2,out);
 input [7:0] in1,in2 ;
 output reg [7:0] out ;
 integer i;

 always @ (in1,in2)
 begin
 i = -1;
 do
 begin
 i = i+1;
 if (i == 6) continue;
 out[i] = in1[i] & in2[i];
 end
 while (i < 8);
 out[6] = in1[6] | in2[6];
 end
endmodule
```

# CG513

## **@E: No definition for function lvar**

This SystemVerilog assertion error occurs when the LET construct is used inside the ASSERT block. In the test case below, this scenario is not ignored, so an error message is generated.

```
module top;
 reg a;
 let lvar(arg) = arg;

 always_comb AN: assert(lvar(a));
 AF: assert final(a);
 assert final(a)
```

```
 $display("%m passed at %0d",$time);
else
 $display("%m failed at %0d",$time);

endmodule
```

## Action

Please contact Synopsys support.

# CG514

### **@E: Expecting block name**

The optional block identifier specified after the colon in a block statement is invalid. In the test case below, there is no block name identifier following the colon in the `always` block statement which results in the error.

```
module select (output reg c,input a, b);
always @(a or b)
begin :
if (a==1)
c = b;
else
c= 0;
end
endmodule
```

## Action

Provide a valid block name after the colon in a block statement as shown in the corrected test case below.

```
module select (output reg c,input a, b);
 always @(a or b)
 begin : b1
 if (a==1)
 c = b;
 else
 c= 0;
 end
 endmodule
```

## CG515

### **@W: defparam is supported only for instances of the current module**

defparam is used with hierachial path names that are used to redefine the parameters in the lower level. The compiler currently only supports defparam specification for the current level and not for any levels below it. The following test case gives the warning as the defparam is being used for levels deeper than one (i.e., 2 indicated in specification).

```
defparam test3.test2.operand1_width =10;

module multiplier (operand1, operand2, result);
 parameter operand1_width = 4, operand2_width =2;
 input [operand1_width -1:0]operand1;
 input [operand2_width -1:0] operand2;
 output [operand1_width +operand2_width -2:0] result;
 assign result = operand1 * operand2;
endmodule

module level2(op1,op2,result);
 defparam test2.operand1_width =16;
 defparam test2.operand2_width =8;
 input [15:0] op1;
 input [7:0] op2;
 output [22:0]result;
 multiplier test2 (op1, op2, result);
endmodule
```

```
module level3(op1,op2,result);
defparam test3.test2.operand1_width =10;
defparam test3.test2.operand2_width =4;
input [9:0] op1;
input [3:0] op2;
output [12:0]result;
level2 test3 (op1, op2, result);
endmodule
```

## Action

Make sure that `defparams` are used only on the current level of instances. To change the parameter value across more than one level of hierarchy, use in-line parameter assignment as shown below in the corrected test case below.

```
module multiplier (operand1, operand2, result);
parameter operand1_width = 4, operand2_width =2;
input [operand1_width -1:0]operand1;
input [operand2_width -1:0] operand2;
output [operand1_width +operand2_width -2:0] result;
assign result = operand1 * operand2;
endmodule

module level2(op1,op2,result);
parameter operand1_width =16;
parameter operand2_width =8;
input [15:0] op1;
input [7:0] op2;
output [22:0]result;
multiplier
#(.operand1_width(operand1_width),.operand2_width(operand2_width))
 test2 (op1, op2, result);
endmodule

module level3(op1,op2,result);
parameter operand1_width =10;
parameter operand2_width =4;
input [9:0] op1;
input [3:0] op2;
output [12:0]result;
level2
#(.operand1_width(operand1_width),.operand2_width(operand2_width))
 test3 (op1, op2, result);
endmodule
```

## CG519

### **@E: initial values for multi-dimensional ports not supported yet**

An initial value was found on a multi-dimensional port (initial values on multi-dimensional ports are not supported). In the test case below, the initial value on port `in` causes the error.

```
module test (in,out);
 input [7:0]in[1:0]= 16'h0;
 output [7:0] out;
 nand u1[7:0] (out,in[0],in[1]);
endmodule
```

### Action

Avoid assigning initial values to multi-dimensional ports as shown in the corrected test case below.

```
module test (in,out);
 input [7:0]in[1:0];
 output [7:0] out;
 nand u1[7:0] (out,in[0],in[1]);
endmodule
```

## CG523

### **@E: endmodule label should match module name <test>**

SystemVerilog; the `endmodule` label did not match the name of the module. Using an `endmodule` label allows a block name to be defined after the `end` keyword when the name matches the one defined on the corresponding `begin` keyword. In the test case below, module name `test` does not match the `endmodule` statement label `test1` which causes the error.

```
module test (in1,in2,out1,out2);
 input in1,in2 ;
 output reg out1,out2;
 reg a,b;
```

```
always@(in1,in2)
begin : foo_in
 a = in1 & in2;
 b = in2 | in1;
end : foo_in

always@(a,b)
begin: foo_value
 out1 = a;
 out2 = b;
end : foo_value

endmodule : test1
```

## Action

Make sure that the label following the `endmodule` statement is same as the corresponding module name. In the corrected test case below, the label after the `endmodule` statement is `test` which matches the module name.

```
module test (in1,in2,out1,out2);
input in1,in2 ;
output reg out1,out2;
reg a,b;

always@(in1,in2)
begin : foo_in
 a = in1 & in2;
 b = in2 | in1;
end : foo_in

always@(a,b)
begin: foo_value
 out1 = a;
 out2 = b;
end : foo_value

endmodule : test
```

# CG525

## @E: unique or priority can only precede case or if statements

SystemVerilog; a unique or priority modifier was used for a statement other than a case or if statement. In the test case below, the presence of a priority modifier on the casez statement causes the error.

```
module test (out, a, b, c, d, select);
 output out;
 input a, b, c, d;
 input [3:0] select;
 reg out;

 always @ (select or a or b or c or d)
 begin
 priority casez (select)
 4'b???1: out = a;
 4'b??1?: out = b;
 4'b?1???: out = c;
 4'b1????: out = d;
 endcase
 end
 endmodule
```

### Action

Apply the unique and priority modifiers only to case and if statements as shown in the corrected test case below.

```
module test (out, a, b, c, d, sel);
 output out;
 input a, b, c, d;
 input [3:0] sel;
 reg out;

 always @ (a,b,c,d,sel)
 begin
 priority case (sel)
 4'b0001: out = c;
 4'b0010: out = b;
```

```
 4'b0100: out = d;
 4'b1000: out = a;
 endcase
end
endmodule
```

## CG526

### @E: Expecting identifier or initial assignment stmt(s)

An identifier or initial statement could not be found. In the test case below, the missing initial statement in the for loop results in the error.

```
module test(in1,out1);
input [2:0] in1;
output [2:0] out1;
wire [2:0] count;
reg [2:0] temp ;
assign count = in1;

always @ (count)
begin
integer i;
 for (; i < 3; i = i+1)
begin : foo
 temp = count + 1;
end
end

assign out1 = temp;
endmodule
```

### Action

Make sure that the expected identifier or initial statement is included. In the corrected test case below, adding initial statement `i=0` in the for loop eliminates the error.

```
module test(in1,out1);
 input [2:0] in1;
 output [2:0] out1;
 wire [2:0] count;
 reg [2:0] temp ;
 assign count = in1;

 always @ (count)
 begin
 integer i;
 for (i=0; i < 3; i = i+1)
 begin : foo
 temp = count + 1;
 end
 end

 assign out1 = temp;
endmodule
```

## CG527

### **@E: End block label should match block label <foo\_value>**

SystemVerilog; the optional block name defined by the end keyword did not match the block name at the corresponding begin keyword. In the test case below, the begin statement label for the second always block (foo\_value) does not match the corresponding end statement label (foo\_value) which results in the error.

```
module test (in1,in2,out1,out2);
 input in1,in2 ;
 output reg out1,out2;
 reg a,b;

 always@(in1,in2)
 begin : foo_in
 a = in1 & in2;
 b = in2 | in1;
 end : foo_in
```

```
always@(a,b)
begin: foo_value
 out1 = a;
 out2 = b;
end : foo_valu
endmodule
```

## Action

Make sure that the block name defined by the end keyword matches the block name defined on the corresponding begin keyword. In the corrected test case below, the labels at both the begin and end statements of the second always block match (foo\_value).

```
module test (in1,in2,out1,out2);
input in1,in2 ;
output reg out1,out2;
reg a,b;

always@(in1,in2)
begin : foo_in
 a = in1 & in2;
 b = in2 | in1;
end : foo_in

always@(a,b)
begin: foo_value
 out1 = a;
 out2 = b;
end : foo_value

endmodule
```

# CG528

## @W: Ignoring full\_case directive on priority case statement

SystemVerilog; a case statement has both a priority and a full\_case implementation. SystemVerilog adds unique and priority modifiers to case statements; the priority case implementation in SystemVerilog is equivalent to adding a full\_case directive to the case statement in Verilog. The compiler ignores the full\_case directive as indicated by the warning.

```
module test (out, a, b, c, d, sel);
 output out;
 input a, b, c, d;
 input [3:0] sel;
 reg out;

 always @ (a,b,c,d,sel)
 begin
 priority case (sel) /* synthesis full_case*/
 4'b0001: out = c;
 4'b0010: out = b;
 4'b0100: out = d;
 4'b1000: out = a;
 endcase
 end
endmodule
```

### Action

Do not use the full\_case directive when a priority modifier is used on the case statement as shown in the corrected test case below.

```
module test (out, a, b, c, d, sel);
 output out;
 input a, b, c, d;
 input [3:0] sel;
 reg out;
```

```
always @ (a,b,c,d,sel)
begin
 priority case (sel)
 4'b0001: out = c;
 4'b0010: out = b;
 4'b0100: out = d;
 4'b1000: out = a;
 endcase
end
endmodule
```

## CG529

### @W: Ignoring full\_case directive on unique case statement

SystemVerilog; a unique case statement has a parallel\_case and/or a full\_case directive. SystemVerilog adds unique and priority modifiers to case statements; the unique case implementation in SystemVerilog is equivalent to adding a parallel\_case directive to the case statement in Verilog and, without a default case implementation, is equivalent to adding full\_case parallel\_case directives. The compiler ignores the full\_case directive as indicated by the warning.

```
module test (out, a, b, c, d, sel);
output out;
input a, b, c, d;
input [3:0] sel;
reg out;

always @ (a,b,c,d,sel)
begin
 unique case (1'b1) /*synthesis full_case parallel_case*/
 sel[0]: out = a;
 sel[1]: out = b;
 sel[2]: out = c;
 sel[3]: out = d;
 endcase
end
endmodule
```

## Action

Do not use the `full_case` `parallel_case` directives when a `unique` modifier is used on the `case` statement as shown in the corrected test case below.

```
module test (out, a, b, c, d, sel);
 output out;
 input a, b, c, d;
 input [3:0] sel;
 reg out;

 always @ (a,b,c,d,sel)
 begin
 unique case (1'b1)
 sel[0]: out = a;
 sel[1]: out = b;
 sel[2]: out = c;
 sel[3]: out = d;
 default: out = 'bx;
 endcase
 end
endmodule
```

## CG530

### **@W: Ignoring parallel\_case directive on priority case statement**

SystemVerilog; a priority case statement includes a redundant `parallel_case` directive. SystemVerilog adds `unique` and `priority` modifiers to `case` statements; a `priority case` implementation in SystemVerilog is equivalent to adding a `parallel_case` directive to a `case` statement in Verilog. The compiler ignores the `full_case` directive as indicated by the warning.

```
module test (out, a, b, c, d, sel);
 output out;
 input a, b, c, d;
 input [3:0] sel;
 reg out;
```

```
always @ (a,b,c,d,sel)
begin
 priority case (sel) /* synthesis parallel_case */
 4'b0001: out = c;
 4'b0010: out = b;
 4'b0100: out = d;
 4'b1000: out = a;
 endcase
end
endmodule
```

## Action

Do not use a `parallel_case` directive when a `priority` modifier is used within the `case` statement as shown in the corrected test case below.

```
module test (out, a, b, c, d, sel);
output out;
input a, b, c, d;
input [3:0] sel;
reg out;

always @ (a,b,c,d,sel)
begin
 priority case (sel)
 4'b0001: out = c;
 4'b0010: out = b;
 4'b0100: out = d;
 4'b1000: out = a;
 endcase
end
endmodule
```

# CG531

## @W: Ignoring parallel\_case directive on unique case statement

SystemVerilog; a unique case statement has a `parallel_case` and/or a `full_case` directive. SystemVerilog adds unique and priority modifiers to case statements; the unique case with a default implementation in SystemVerilog is equivalent to adding a `parallel_case` directive to the `case` statement in Verilog and, without a

default case implementation, is equivalent to adding full\_case parallel\_case directives. The compiler ignores the parallel\_case directive as indicated by the warning.

```
module test (out, a, b, c, d, sel);
 output out;
 input a, b, c, d;
 input [3:0] sel;
 reg out;

 always @ (a,b,c,d,sel)
 begin
 unique case (1'b1) /*synthesis full_case parallel_case*/
 sel[0]: out = a;
 sel[1]: out = b;
 sel[2]: out = c;
 sel[3]: out = d;
 endcase
 end
endmodule
```

## Action

Do not use the parallel\_case full\_case directives when a unique modifier is used on the case statement as shown in the corrected test case below.

```
module test (out, a, b, c, d, sel);
 output out;
 input a, b, c, d;
 input [3:0] sel;
 reg out;

 always @ (a,b,c,d,sel)
 begin
 unique case (1'b1)
 sel[0]: out = a;
 sel[1]: out = b;
 sel[2]: out = c;
 sel[3]: out = d;
 default: out = 'bx;
 endcase
 end
endmodule
```

## CG532

**@W: Within an initial block, only Verilog force statements and memory \$readmemh/\$readmemb initialization statements are recognized, and all other content is ignored.**

Indicates that the compiler has encountered an initial block and that the contents of the block, with the exception of any \$readmemh/\$readmemb memory initialization or Verilog force statements, are being ignored.

```
module top(input clk,we, input in1, input [3:0] addr, output out);
reg mem [15:0];
initial
begin
 mem[0]=0;
end

always @(posedge clk)
begin
if (we)
 mem[addr] <= in1;
end

assign out = mem[addr];
endmodule
```

### Action

Check the initial block. If the block does not include any force, \$readmemh, or \$readmemb statements, comment out the entire block using synthesis translate\_off and synthesis translate\_on directives as shown in the following code segment and recompile to eliminate the warning.

```
//synthesis translate_off
initial
begin
 mem[0]=0;
end
//synthesis translate_on
```

If the block includes a valid force, \$readmemh, or \$readmemb statement as shown in the segment below, simply ignore the warning message and keep the indicated operation (initializing RAM with hex data).

```

initial
begin
 $readmemh("data.dat" , mem);
end

```

## CG533

### **@E: Cannot mix positional and named arguments**

SystemVerilog; a function call has both positional and named arguments. In the test case below, function call parity has both a positional argument (a) and a named argument (.b1(b)) which causes the error.

```

module test (a, b, c);
 input [3:0] a, b;
 output c;
 assign c = parity(a,.b1(b));
 function parity(input [3:0] a1,b1);
 reg [0:3] ret;
 integer j;
 begin
 ret = 0;
 for (j= 0; j <= 3; j= j + 1)
 if (a1[j] & b1[j] == 1)
 ret = ret + 1;
 parity = ret % 2;
 end
 endfunction
endmodule

```

### Action

Make sure that all function calls have either positional arguments or named arguments, but not both. In the corrected test case below, both argument types are listed with the positional argument type commented out.

```

module test (a, b, c);
 input [3:0] a, b;
 output c;
 //assign c = parity(a,b); /* Positional Argument-based
 //Function call */

```

```
assign c = parity(.a1(a),.b1(b)); /* Named Argument-based
Function call */
function parity(input [3:0] a1,b1);
reg [0:3] ret;
integer j;
begin
 ret = 0;
 for (j= 0; j <= 3; j= j + 1)
 if (a1[j] & b1[j] == 1)
 ret = ret + 1;
 parity = ret % 2;
 end
endfunction
endmodule
```

## CG534

### @E: Expecting function argument name

SystemVerilog; a function was called without proper arguments. In the function call in the test case below, the first argument is not properly named which causes the error.

```
module test (a, b, c);
input [3:0] a, b;
output c;
assign c = parity(.(a),.b1(b));
function parity(input [3:0] a1,b1);
reg [0:3] ret;
integer j;
begin
 ret = 0;
 for (j= 0; j <= 3; j= j + 1)
 if (a1[j] & b1[j] == 1)
 ret = ret + 1;
 parity = ret % 2;
 end
endfunction
endmodule
```

## Action

Make sure that the function call has proper arguments as shown in the corrected test case below.

```
module test (a, b, c);
 input [3:0] a, b;
 output c;
 assign c = parity(.a1(a),.b1(b));
 function parity(input [3:0] a1,b1);
 reg [0:3] ret;
 integer j;
 begin
 ret = 0;
 for (j= 0; j <= 3; j= j + 1)
 if (a1[j] & b1[j] == 1)
 ret = ret + 1;
 parity = ret % 2;
 end
 endfunction
endmodule
```

# CG535

## **@E: Only one dimension is allowed for an array of instances**

A multidimensional instantiation label was encountered in an array of instances. In the test case below, 2-dimensional instantiation label u1[7:0][2:0] in the array of instances causes the error.

```
module test (in1,in2,out);
 input [7:0] in1,in2;
 output [7:0] out;
 nand u1[7:0][2:0] (out,in1,in2);
endmodule
```

## Action

Place only a one dimension instantiation label in an array of instances as shown in the corrected test case given below.

```
module test (in1,in2,out);
 input [7:0]in1;
 input [7:0]in2;
 output [7:0] out;
 nand u1[7:0] (out,in1,in2);
endmodule
```

## CG540

### **@E: module <sub> is undefined, hence parameters cannot be found**

The compiler encountered a defparam statement for an instantiation of an undefined module. In the test case below, module sub is not defined and there is a defparam statement (defparam u2.param) for its instantiation which causes the error.

```
module top (q,a,b,c,d);
 parameter top_param=4;
 output [top_param: 0] q;
 input [top_param:0] a,b,c,d;
 sub u2 (q,a,b,c,d);
defparam u2.param=4;
endmodule
```

### Action

Make sure that there is a corresponding module definition for every defparam statement. In the corrected test case below, module definition sub has been added to the RTL code.

```
module top (q,a,b,c,d);
 parameter top_param=4;
 output [top_param: 0] q;
 input [top_param:0] a,b,c,d;
 sub u2 (q,a,b,c,d);
 defparam u2.param=4;
endmodule
```

```
module sub (q,a,b,c,d);
parameter param=4;
output [param: 0] q;
input [param:0] a,b,c,d;
assign q= a & b & c & d;
endmodule
```

## CG543

### **@E: Could not find argument <b> in function**

SystemVerilog; the argument specified in a function based on a named argument is not present in the actual functional definition. In the test case below, function parity includes arguments a1 and b1. When argument b in the function call is encountered, the compiler errors out.

```
module test (a, b, c);
input [3:0] a, b;
output c;
assign c = parity(.a1(a),.b(b));

function parity(input [3:0] a1,b1);
reg [0:3] ret;
integer j;
begin
ret = 0;
for (j= 0; j <= 3; j= j + 1)
if (a1[j] & b1[j] == 1)
ret = ret + 1;
parity = ret % 2;
end
endfunction
endmodule
```

### Action

Make sure that the arguments in the function call are the same as the arguments defined in the function definition. In the corrected test case below, the function call includes arguments a1 and b1 which match the arguments specified in the function definition.

```

module test (a, b, c);
 input [3:0] a, b;
 output c;
 assign c = parity(.a1(a),.b1(b));

 function parity(input [3:0] a1,b1);
 reg [0:3] ret;
 integer j;
 begin
 ret = 0;
 for (j= 0; j <= 3; j= j + 1)
 if (a1[j] & b1[j] == 1)
 ret = ret + 1;
 parity = ret % 2;
 end
 endfunction
endmodule

```

## CG548

**@E: No connection specified for <a> of subprogram parity - must specify at least a default argument value**

SystemVerilog; a subprogram includes arguments without default values (the subprogram call must pass values for each of the arguments). In the test case below, function parity includes two arguments (a1 and b1) that do not have default values (the function call parity() has no arguments) which results in the error.

```

module test (a, b, c);
 input [3:0] a, b;
 output c;
 assign c = parity();

 function parity(input [3:0] a1,b1);
 reg [0:3] ret;
 integer j;
 begin
 ret = 0;
 for (j= 0; j <= 3; j= j + 1)
 if (a1[j] & b1[j] == 1)

```

```

 ret = ret + 1;
 parity = ret % 2;
 end
endfunction
endmodule

```

## Action

Do either of the following:

- Assign default values for arguments in the subprogram definition so that the function call does not require any arguments. In the test case below, function parity defines a default value for b1 so that the corresponding argument in the function call can be omitted.

```

module test (a, b, c);
input [3:0] a, b;
output c;
assign c = parity(a);

function parity(input [3:0] a1,b1=4'b1);
reg [0:3] ret;
integer j;
begin
ret = 0;
for (j= 0; j <= 3; j= j + 1)
if (a1[j] & b1[j] == 1)
ret = ret + 1;
parity = ret % 2;
end
endfunction
endmodule

```

- Specify all arguments in the function call as shown in the test case below.

```

module test (a, b, c);
input [3:0] a, b;
output c;
assign c = parity(a, b);

function parity(input [3:0] a1,b1);
reg [0:3] ret;
integer j;
begin
ret = 0;
for (j= 0; j <= 3; j= j + 1)

```

```
 if (a1[j] & b1[j] == 1)
 ret = ret + 1;
 parity = ret % 2;
 end
endfunction
endmodule
```

## CG553

### **@E: Packed dimension must specify a range**

SystemVerilog; the packed dimension preceding the object name did not include a valid range. In the example below, the code segment results in an error as the range for bit is invalid.

```
bit [5] memory1[99:0]; // not a range
```

#### Action

Make sure that the range dimension declared before an object is complete when using packed arrays. The segment below shows a valid range dimension for bit.

```
bit [5:0] memory1[99:0]; // includes a range
```

## CG556

### **@E: <endfunction> label should match <function> <parity>**

SystemVerilog; an endfunction label did not match the corresponding begin function keyword (SystemVerilog allows a block name to be defined after an end keyword). In the test case below, the function name parity does not match the endfunction label parity1 which results in the error.

```
module test (a, b, c);
 input [3:0] a, b;
 output c;
 assign c = parity(a,b);

 function parity(input [3:0] a1,b1);
 reg [0:3] ret;
 integer j;
 begin
 ret = 0;
 for (j= 0; j <= 3; j= j + 1)
 if (a1[j] & b1[j] == 1)
 ret = ret + 1;
 parity = ret % 2;
 end
 endfunction : parity1
endmodule
```

## Action

Make sure that the label after an `end` statement matches the corresponding block name. In the corrected test case below, the `endfunction` label `parity` matches the function name.

```
module test (a, b, c);
 input [3:0] a, b;
 output c;
 assign c = parity(a,b);

 function parity(input [3:0] a1,b1);
 reg [0:3] ret;
 integer j;
 begin
 ret = 0;
 for (j= 0; j <= 3; j= j + 1)
 if (a1[j] & b1[j] == 1)
 ret = ret + 1;
 parity = ret % 2;
 end
 endfunction : parity
endmodule
```

## CG559

### @W: Priority case statement has a default clause that is being implemented as a regular case statement.

SystemVerilog; a priority case statement includes a default clause (priority case in SystemVerilog is equivalent to full\_case in Verilog which does not require a default clause). In the test case below, the priority case statement includes a default clause as indicated by the warning.

```
module test (out, a, b, c, d, sel);
 output out;
 input a, b, c, d;
 input [3:0] sel;
 reg out;

 always @ (a,b,c,d,sel)
 begin
 priority case (sel)
 4'b0001: out = c;
 4'b0010: out = b;
 4'b0100: out = d;
 4'b1000: out = a;
 default:out = 'bx;
 endcase
 end
endmodule
```

### Action

Remove the default clause in the priority case statement as shown in the corrected test case below.

```
module test (out, a, b, c, d, sel);
 output out;
 input a, b, c, d;
 input [3:0] sel;
 reg out;

 always @ (a,b,c,d,sel)
 begin
 priority case (sel)
 4'b0001: out = c;
 4'b0010: out = b;
```

```
4'b0100: out = d;
4'b1000: out = a;
endcase
end
endmodule
```

## CG600

### **@E: Port '*portName*' of instance '*instanceName*' must be connected to interface instance or interface port.**

An interface port of a module in SystemVerilog is incorrectly mapped to a non-interface signal or port. In the following test case, port din is not connected to an interface port which results in the error.

```
interface ifc ();
 logic tmp;
endinterface

module tst(input din, output dout);
 ifc I0();
 assign I0.tmp = !din;
 sub S0(din, dout);
endmodule

module sub(interface myif, output ddout);
 assign ddout = myif.tmp;
endmodule
```

This same error can occur if an instance of a module contains a specific port type, but the port definition originates from an include file that is not referenced in the module.

### Action

Make sure that any interface ports are connected to interface instances or interface ports.

```
interface ifc ();
 logic tmp;
endinterface
```

```
module tst(input din, output dout);
 ifc I0();
 assign I0.tmp = !din;
 sub S0(I0, dout);
endmodule

module sub(interface myif, output ddout);
 assign ddout = myif.tmp;
endmodule
```

## CG612

### @E: Interface 'intf' doesn't define method 'p2' invoked via interface instance 'I'.

This SystemVerilog assertion error occurs when the hierarchy for the property defined by the interface cannot be accessed. In the test case below, this scenario is not ignored, so an error message is generated.

```
module top;
 logic clk;
 intf I(clk);

 initial clk = 0;
 always #1 clk = !clk;
 a2: assume property (I.p2(I.a |=> I.c));
endmodule

interface intf (input clk);
 reg a, b, c;

 default clocking dc @ (posedge clk);
 property p1(arg);
 disable iff (c) arg;
 endproperty
 endclocking
endinterface
```

```
property p2(arg1);
 @(dc) if(1) arg1;
endproperty

endinterface
```

## Action

Please contact Synopsys support.

# CG624

### **@E: Too many instance parameter assignments**

A parameter is assigned incorrectly. In the SystemVerilog test case below, there is nothing to receive the parameter values 4 and 2 specified in the first line of the instantiation of pmod into top-level module pmod\_wrapper which results in the error.

```
module pmod (
 input logic clk,
 input logic rst,
 input logic [3:0] d,
 output logic [3:0] q);

 always @ (posedge clk or negedge rst)
 if (!rst) q <= 0;
 else q <= d;
 endmodule

module pmod_wrapper (
 input logic p_clk,
 input logic p_RST,
 input logic [3:0] p_in,
 output logic [3:0] p_out);

 pmod #(4,2) i_pmod (
 .clk(p_clk),
 .rst(p_RST),
 .d(p_in),
 .q(p_out));
 endmodule
```

## Action

Make sure that there are parameters in the instantiated module to receive the corresponding values, as shown in the corrected test case below.

```
module pmod (
 input logic clk,
 input logic rst,
 input logic [3:0] d,
 output logic [3:0] q);
parameter msb = 4;
parameter lsb = 2;

always @(posedge clk or negedge rst)
 if (!rst) q <= 0;
 else q <= d + msb - lsb;
endmodule

module pmod_wrapper (
 input logic p_clk,
 input logic p_RST,
 input logic [3:0] p_in,
 output logic [3:0] p_out);

pmod #(4,2) i_pmod (
 .clk(p_clk),
 .rst(p_RST),
 .d(p_in),
 .q(p_out));
endmodule
```

# CG649

## **@E: Macro default\_nettpe cannot be used within an interface**

A `default\_nettpe macro was found within an interface definition. In the following test case, macro `default\_nettpe appears inside the interface definition which causes the error.

```
interface intf;
`default_nettpe none;
 logic sig;
endinterface
```

```
module top (intf MyIntf, output out);
 assign MyIntf1.sig = MyIntf.sig;
 assign out = MyIntf1.sig;
endmodule
```

## Action

Make sure that a `default\_nettype macro is used only outside of an interface definition.

# CG672

## @E: Expecting package name

The compiler parsed a Verilog module and did not encounter the package name. In the following test case, the package name is not mentioned in the import statement which results in the error.

```
//In test_pkg.sv- Package definition
package test_pkg;
 typedef enum {a, b, c} SOP;
 typedef enum {x, y, z} SOL;
 typedef enum {p, q} NOP;
endpackage

//In test.sv
import;
module xyz (input signal1, signal2,
 input SOP SOP_in,
 input SOL SOL_in,
 input NOP NOP_in,
 output abc_out);
 assign abc_out = signal1 ? SOP_in : signal2 ? SOL_in : NOP_in;
endmodule
```

## Action

To eliminate the error in the test case above, provide the package name in the import statement as follows:

```
import test_pkg::*;
module xyz (input signal1, signal2,
 input SOP SOP_in,
 input SOL SOL_in,
 input NOP NOP_in,
 output abc_out);
 assign abc_out = signal1 ? SOP_in : signal2 ? SOL_in : NOP_in;
endmodule
```

## CG679

### @W: Previous <module> with same name <moduleName>

An earlier module declaration of the same name has been encountered in the design. This warning occurs only when duplicate modules are not allowed (Allow Duplicate Modules unchecked on the Verilog tab) and is accompanied by error message [CG855](#). The source location in the Messages window points to the first occurrence of the duplicate module name.

#### Action

If two modules with the same name are intended to be used, enable Allow Duplicate Modules on the Verilog tab or include a `set_option -allow_duplicate_modules` command with a value of 1 in the project file. Note that when you enable duplicate modules, the last module description encountered is used.

## CG702

### @E: Cannot store expression type to output

The output of an instance is connected to an expression, which is not allowed in Verilog. In the example below, the output of module simpleAssign is assigned the value `~result` (an expression) which causes the error.

```
module simpleAssign (D, Q);
 input D;
 output Q;
 assign Q = D;
endmodule

module Count4(inVal, result);
 input inVal;
 output result;
 simpleAssign inst1(inVal, ~result);
endmodule
```

## Action

Avoid assignment of an expression as an output.

# CG729

## @E: Expecting port name, got keyword <do>

A reserved keyword was incorrectly used to define a port or variable name in a SystemVerilog design.

In the following test case, the SystemVerilog construct do is used as a port name which results in the error.

```
module top (din1,din2,out);
 input [7:0] din1,din2;
 output [7:0] out;
 src u1 (.do(din1),.do2(din2),.out(out));
endmodule

module src (do,do1,out);
 input [7:0] do,do1;
 output [7:0] out;
 assign out = do^ do1;
endmodule
```

## Action

Make sure that a reserved keyword is not used as a port name.

```
module top (din1,din2,out);
 input [7:0] din1,din2;
 output [7:0] out;
 src u1 (.do1(din1),.do2(din2),.out(out));
endmodule

module src (do1,do2,out);
 input [7:0] do1,do2;
 output [7:0] out;
 assign out = do1^ do2;
endmodule
```

## CG731

### **@E: Can't synthesize UDP primitives**

While reading source HDL files containing UDPs (user-defined primitives), a UDP was encountered that had not been instantiated in the design (the Verilog compiler does not support UDPs directly).

#### Action

When a UDP is to be instantiated in the final design, make sure that a synthesizable definition of the module exists to perform the function of the UDP. The synthesis tool checks the syntax to determine if the HDL code can be synthesized before synthesis begins including the syntax of any user-defined primitives.

## CG737

### **@E: <inst\_mux> already exists in this scope - a variable by this name exists**

The same signal name and instance name are used in a module.

## Test Case 1

In the following test case, variable `inst_mux`, which is declared as a wire/reg, is also being used as an instance name which results in the error.

```
module CG737 (sel, in1, in2, out);
 input sel, in1, in2;
 output out;
 wire inst_mux;

 //reg inst_mux;
 mux inst_mux (.sel(sel), .in1(in1), .in2(in2), .out(inst_mux));
 assign out = inst_mux;
endmodule
```

## Action

In the same module, the names for signals and instances must be unique. Changing the instance name from `inst_mux` to `inst1_mux` eliminates the error in the test case.

```
module CG737 (sel, in1, in2, out);
 input sel, in1, in2;
 output out;
 wire inst_mux;

 //reg inst_mux;
 mux inst1_mux (.sel(sel), .in1(in1), .in2(in2), .out(inst_mux));
 assign out = inst_mux;
endmodule
```

## Test case 2

In the following test case, `MyIntf`, which is used as an interface port, is also being used as the instance name of the interface which results in the error.

```
interface intf;
 logic sig;
endinterface

module top (intf MyIntf, output out);
 intf MyIntf();
 assign MyIntf1.sig = MyIntf.sig;
 assign out = MyIntf1.sig;
endmodule
```

## Action

Make sure that the names of any interface ports are not the same as any instances names. Changing the interface port name from MyIntf() to MyIntf1() eliminates the error in the test case.

```
interface intf;
 logic sig;
endinterface

module top (intf MyIntf, output out);
 intf MyIntf1();
 assign MyIntf1.sig = MyIntf.sig;
 assign out = MyIntf1.sig;
endmodule
```

# CG744

## @E: Need simple constant expression

The compiler could not find a simple constant expression required by an identifier. Simple constant expressions include positive, negative, or zero values. Variables that have been assigned a constant expression also can be used as simple constant expressions. In the test case below, reg variable temp is assigned the non-constant value (a==b) which causes the error.

```
module test(a,b,c);
 input a, b;
 output c;
 reg temp = (a==b);
 assign c= temp?1:0;
endmodule
```

## Action

There are several specific identifiers that require simple constant expressions including reg and integer variables and array indices. In the corrected test case below, reg variable temp is assigned the constant expression 1. This error can also be eliminated by using a variable that has been assigned a constant expression (for example, declaring wire t=1 and then assigning reg temp=t).

```
module test(a,b,c);
 input a, b;
 output c;
 reg temp = 1;
 assign c= temp?1:0;
endmodule
```

## CG768

### @E: Expecting escaped identifier after \

A backslash character (\) is immediately followed by a space character instead of the escaped identifier (a space is used to terminate an escaped identifier and cannot be a valid escape identifier). In the test case below, the space after the backslash in the wire declaration causes the error.

```
module cg856 (a,b,c,out);
 input a, b, c;
 output out;
 wire \ temp$n ;
 assign \temp$n = a & b;
 assign out = !\temp$n | c;
endmodule
```

### Action

Make sure that a space character does not immediately follow the backslash in an escaped identifier. In the corrected test case below, identifier temp\$n immediately follows the backslash and a space terminates the identifier string making it a valid escaped identifier.

```
module cg856 (a,b,c,out);
 input a, b, c;
 output out;
 wire \temp$n ;
 assign \temp$n = a & b;
 assign out = !\temp$n | c;
endmodule
```

## CG781

**@W: Input <d1> on instance <u1> is undriven; assigning to 0.  
Simulation mismatch possible. Either assign the input or remove  
the declaration.**

An input port has been left unconnected for a given instance. In the test case below, input d1 has been left unconnected as indicated in the warning.

```
module sub(
 input d1,
 output dout1,dout2);
assign dout1 = d1;
assign dout2 = 1'b1;
endmodule

module top(
 input d1,
 output dout1,dout2,dout3);
sub u1 (,,dout1);
sub u2 (d1,dout2,dout3);
endmodule
```

### Action

Make sure that all ports of a module instance are connected and that no port remains unconnected.

```
module sub(
 input d1,
 output dout1,dout2);
assign dout1 = d1;
assign dout2 = 1'b1;
endmodule

module top(
 input d1,
 output dout1,dout2,dout3);
sub u1 (d1,,dout1);
sub u2 (d1,dout2,dout3);
endmodule
```

## CG807

### **@E: Illegal use of streaming concatenation - stream operands require explicit type cast to be used with other operators.**

When the stream operator is used with another operator (such as equality), a cast is required so that the type of the stream expression is known. In the test case below, the type of stream is unknown which results in the error.

```
module streaming();
int a, b, c;
bit [2:0] d;

assign d = b == {>>{ a, b, c }};
endmodule
```

#### Action

Adding a cast as shown in the corrected example below resolves the error.

```
module streaming();
int a, b, c;
bit [2:0] d;

assign d = b == (shortint'({>>{ a, b, c }}));
endmodule
```

## CG808

### **@E: Size mismatch between value assignment and enum name declaration**

SystemVerilog; the size of the value assigned to an element of an enumerated type differs from the size of its base type. In the test case below, the base type is logic which is of size 1 (i.e., single bit, no vector defined). However, because the enumeration element `himp` is assigned the value `2'bz` which is of size 2, the compiler errors out.

```
module test (a,b,c,q);
 input a,b,c;
 output q;
 enum logic {a[2],himp=2'bz} temp;

 always @(a or b)
 if (a)
 temp=a1;
 else if (b)
 temp= a0;
 else
 temp=himp;

 assign q= c ? temp : 1'bz;
endmodule
```

## Action

Make sure that the value assignment to any element of an enumerated type has the same size as that of its base type as shown in the corrected test case where the enumerated element himp is assigned a value 1'bz.

```
module test (a,b,c,q);
 input a,b,c;
 output q;
 enum logic {a[2],himp=1'bz} temp;

 always @(a or b)
 if (a)
 temp=a1;
 else if (b)
 temp= a0;
 else
 temp=himp;

 assign q= c ? temp : 1'bz;
endmodule
```

# CG855

## @E: Duplicate <module/interface> name <sub>

Duplicate module names were found in the design.

### Test Case 1

In the test case below, two sub-modules have the same name (sub).

```
module top (q,a,b,c,d);
parameter top_param=4;
output [top_param: 0] q;
input [top_param:0] a,b,c,d;
sub u2 (q,a,b,c,d);
defparam u2.param=4;
endmodule

module sub (q,a,b,c,d);
parameter param=4;
output [param: 0] q;
input [param:0] a,b,c,d;
assign q= a & b & c & d;
endmodule

module sub (q,a,b);
parameter param=4;
output [param: 0] q;
input [param:0] a,b;
assign q= a - b ;
endmodule
```

### Action

Usually, modules with the same name is an oversight. Change one of the names to be unique. To use two modules with the same name, enable Allow Duplicate Modules on the Verilog tab or use the `set_option -dup 1` command. Note that when you enable duplicate modules, the last module description encountered is used.

## Test Case 2

In test case 2, the compiler finds two definitions for the same module when a file containing the module description is explicitly added to a project and an `include statement also references a file containing the same module description as the added file or when different `define files are added to the project multiple times. This message appears with the warning message “Previous module with same name” for that specific module.

In the test case, the project file indicates that the adder.v file, which contains the module description of module adder, is included in the project. The adder16.v file, which is also included in the project file, has an `include adder.v command. The compiler finds a duplicate description for the same module.

```
#Project File Excerpt:

#add_file options
add_file -verilog "adder.v"
add_file -verilog "adder16.v"

#adder.v File

module adder (cout, sum, a, b, cin);
parameter size = 1; /* declare a parameter. Default required */
output cout;
output [size-1:0] sum; // sum uses the size parameter
input cin;
input [size-1:0] a, b; // 'a' and 'b' use the size parameter
assign {cout, sum} = a + b + cin;
endmodule

#adder16.v File

`include "adder.v"
module adder16 (cout, sum, a, b, cin);
output cout;
/* We are also using a parameter at this level of hierarchy */
parameter my_size = 16; // We want a 16-bit adder
output [my_size - 1: 0] sum;
input [my_size - 1: 0] a, b;
input cin;
/* my_size overwrites size inside instance my_adder of adder */
adder #(my_size) my_adder (cout, sum, a, b, cin);
endmodule
```

## Action

Make sure that all modules to be instantiated are defined in your RTL code and add them to the project file. Alternatively, use `include *filename* in the HDL code to add the lower-level module files. To eliminate the error in test case 2, either:

- Add the module definition of adder into the project file with the add file option and comment out the `include adder.v in the adder16.v file.
- Use `include adder.v to add the lower-level module files, but remove the following line from the project file:

```
add_file -verilog "adder.v"
```

You can also eliminate the warning by setting the Allow Duplicate Module option in the user interface or by setting the `set_option -dup` Tcl command to 1.

# CG879

## **@W: Treating non-constant declarative assignment variable <name> as a regular assign based on type - value could be overwritten in later writes.**

A variable assignment specified as a non-constant value can be overwritten during write operations that occur later in the process. In the test case below, the variable temp is assigned a non-constant value of a, which is the input to the module test. In the always block, however, the variable temp is updated with the value of en. Therefore, the synthesis tool generates this warning message.

```
module test(input a, en, output c);
 reg temp = a;
 always@(en)
 temp = en;
 assign c = temp;
endmodule
```

## Action

Make sure that the non-constant assignment for the variable is the expected behavior in the RTL. Otherwise, you must correct the RTL.

# CG884

### **@E: For stream operator target, the right side must be of equal or greater size than the target expression**

When using a streaming operator, the width of the stream operator expression cannot be greater than its assigned target. In the test case below, the 2-bit value (2'd15) is less than the width of the 8-bit stream expression which results in the error.

```
module streaming(o1, o2);
 output bit [3:0] d, o1, o2;
 assign {>>{ o1, o2 }} = 2'd15;
endmodule
```

## Action

Make sure that the operator value width is equal to or greater than its stream expression. In the corrected example below, the 10-bit value (10'd15) is greater than the width of the stream expression.

```
module streaming(o1, o2);
 output bit [3:0] d, o1, o2;
 assign {>>{ o1, o2 }} = 10'd15;
endmodule
```

## CG888

### **@E: Streaming operator: size of destination must be greater than or equal to the target**

The size of a stream operator expression cannot be greater than its assignment. When using a streaming operator, the target of the stream expression must be greater than or equal to its destination. In the test case below, the LHS target (d) is less than the required stream width (96 bits) which results in the error.

```
module streaming (d, e, a, b, c);
 input int a, b, c;
 output bit [63:0] d;
 output bit [95:0] e;
 assign d = {>>{ a, b, c }};
endmodule
```

### Action

Make sure that the size of the destination equals or is greater than its target. In the corrected example below, the LHS target (e) matches the stream width.

```
module streaming (d, e, a, b, c);
 input int a, b, c;
 output bit [63:0] d;
 output bit [95:0] e;
 assign e = {>>{ a, b, c }};
endmodule
```

## CG889

### **@E: Unresolved hierarchical reference <u1.sig1>**

#### Case 1

An undefined variable is being accessed hierarchically. In the test case below, variable sig1 in module top is not defined in module sub.

```
module sub (input byte d1,d2, output byte q1);
byte sig;
assign q1 = d1 ^ d2;
endmodule

module top (input byte d1,d2, output byte q1,q2);
sub u1 (.*);
assign q2 = u1.sig1;
endmodule
```

## Action

Make sure that the correct variable is defined in the module.

```
module sub (input byte d1,d2, output byte q1);
byte sig;
assign q1 = d1 ^ d2;
endmodule

module top (input byte d1,d2, output byte q1,q2);
sub u1 (.*);
assign q2 = u1.sig;
endmodule
```

## Case 2

An attempt was made to access an undefined member of a structure/union. In the test case below, variable nst in module top is not a member of the structure.

```
typedef struct packed
{
 struct packed
 {
 byte st1;
 } nst1;
} structdt;

module top(
 input structdt d1,
 output byte q1);

assign q1 = d1.nst.st1;
endmodule
```

## Action

Make sure that all members of a structure/union are referenced correctly.

```
typedef struct packed
{
 struct packed
 {
 byte st1;
 } nst1;
} structdt;

module top(
 input structdt d1,
 output byte q1);
 assign q1 = d1.nst1.st1;
endmodule
```

## CG893

### **@E: Could not resolve cross-module reference for ‘<moduleName>’**

A cross-module reference to an EDIF black-box could not be resolved (EDIF black-box modules are not supported in cross-module referencing). In the test case below, top module (top.v) includes two instances; rtl (xor\_rtl.v) and edif (logic\_edf.v). Cross-module referencing of rtl (xor\_logic) works correctly, but fails for the edif object (logics).

```
// top.v)
module top (
 input top_a,
 input top_b,
 output top_c,
 output top_d,
 output top_xored,
 output rtl,
 output edif);
logics EDIF_INST (
 //edif
 .a (top_a),
 .b (top_b),
 .c (top_c),
 .d (top_d));
```

```
xor_logic RTL_INST (
 //rtl
 .a (top_a),
 .b (top_b),
 .c (top_xored));

assign rtl = ~top.RTL_INST.c; //Cross module reference
// or rtl; working fine
assign edif = top.EDIF_INST.c; //Cross module reference
// or edif; tool failure
endmodule

// logic_edf.v
module logics (
 input a,
 input b,
 output c,
 output d) /*synthesis syn_black_box */;
endmodule

// xor_rtl.v
module xor_logic (
 input a,
 input b,
 output c);
 assign c = a ^ b;
endmodule
```

## CG906

### @E: Reference to unknown variable tmp\_acomb

This SystemVerilog assertion error occurs when the LET construct is used during continuous assignment. In the test case below, this scenario is not ignored, so an error message is generated.

```
module gray2bin1 (bin, gray);
parameter SIZE = 8; // this module is parameterizable
output [SIZE-1:0] bin;
input [SIZE-1:0] gray;

bit b;
```

```
let tmp_acomb = b;

assign bin = tmp_acomb;

endmodule
```

## Action

Please contact Synopsys support.

# CG915

### **@E: Could not find type for parameter type**

The above error is SystemVerilog related and occurs when an attempt is made to override a parameter with a parameter value that is not defined in the module.

```
module top (
 input clk, rst,
 input [MyParam-1:0] din,
 output [MyParam-1:0] dout);
leaf #(.(WIDTH(MyParam)) (.*);
endmodule

module leaf
#(parameter WIDTH = 0)
 (input clk, rst,
 input [WIDTH-1:0] din,
 output reg[WIDTH-1:0] dout);

 always@(posedge clk, posedge rst) begin
 if(rst)
 dout <= 0;
 else
 dout <= din;
 end
endmodule
```

## Action

Make sure to define the parameter to be overridden. In the corrected example below, MyParam is defined in module top.

```
module top #(parameter MyParam = 1) (
 input clk, rst,
 input [MyParam-1:0] din,
 output [MyParam-1:0] dout);
leaf #(.WIDTH(MyParam)) (.*);
endmodule

module leaf
#(parameter WIDTH = 0)
 (input clk, rst,
 input [WIDTH-1:0] din,
 output reg[WIDTH-1:0] dout);
always@(posedge clk, posedge rst) begin
 if(rst)
 dout <= 0;
 else
 dout <= din;
end
endmodule
```

## CG918

### @E: Ranges are not allowed for type (<int>)

SystemVerilog; a range is declared for a shortint, int, longint, or byte data type. In the test case below, output port equal is declared as type int, but because a range is provided, the compiler errors out.

```
module cg418(equal, a, b);
 output int [1:0] equal;
 input a,b;
 assign equal[1]= a;
 assign equal[0]= b;
endmodule
```

## Action

Either do not specify a range on for data types int, shortint, longint, or byte or use the bit or logic data type if the object requires a range. In the corrected test case below, output port equal is declared as bit.

```
module cg418(equal, a, b);
 output bit [1:0] equal;
 input a,b;
 assign equal[1]= a;
 assign equal[0]= b;
endmodule
```

## CG964

### **@E: Could not resolve hierarchical reference to variable <variableName>**

SystemVerilog; a hierarchical reference to the named variable could not be resolved. In the following test case, the index (2) in the first assignment statement is out of range for variable tmp.

```
module tst(input din, output [1:0] dout);
 genvar jj;
 assign dout[1] = BK.LP[2].tmp;
 assign dout[0] = BK.LP[0].tmp;

 generate begin: BK
 for (jj=0; jj<2; jj=jj+1) begin: LP
 logic tmp;
 assign tmp = !din;
 end
 end endgenerate
endmodule
```

## Action

Make sure that hierarchical references are correctly indexed. For the above test case, changing the assign statement to BK.LP[1].tmp eliminates the error.

## CG978

### **@W:An instance name was not specified - using generated name <II\_O>**

An instance name is missing from an instantiation in the Verilog file. The compiler assumes an instance name and compiles the HDL code. This action causes debugging problems (using find capability in HDL analyst) and problems with the post-synthesis simulation. This warning appears in the test case below because the instantiation of adder is missing an instance name.

```
module adder (cout, sum, a, b, cin);
parameter size = 8; /* declare a parameter. Default required */
output cout;
output [size-1:0] sum; // sum uses the size parameter
input cin;
input [size-1:0] a, b; // 'a' and 'b' use the size parameter
assign {cout, sum} = a + b + cin;
endmodule

module adder8 (cout, sum, a, b, cin);
output cout;
output [7: 0] sum;
input [7: 0] a, b;
input cin;
adder (cout, sum, a, b, cin);
endmodule
```

### Action

Make sure that all instantiations follow the syntax:

*module\_name instance\_name (port\_list);*

To avoid this warning and the naming of the instance in the above test case by the compiler, add the instance name to the instantiation as shown in the corrected test case below.

```

module adder (cout, sum, a, b, cin);
parameter size = 8; /* declare a parameter. Default required */
output cout;
output [size-1:0] sum; // sum uses the size parameter
input cin;
input [size-1:0] a, b; // 'a' and 'b' use the size parameter
assign {cout, sum} = a + b + cin;
endmodule

module adder8 (cout, sum, a, b, cin);
output cout;
output [7: 0] sum;
input [7: 0] a, b;
input cin;
adder adder_inst (cout, sum, a, b, cin);
endmodule

```

## CG980

**@E: Found more than one configuration. Designs with more than one configuration are not supported.**

More than one Verilog configuration is defined for the design. Currently, Verilog configuration support is limited to only one configuration.

```

//Sub Module
module leaf (
 input [7:0] d1,
 input [7:0] d2,
 output reg [15:0] dout);

 always@*
 dout = d1 * d2;
 endmodule

//Top Module
module top (
 input [7:0] d1,
 input [7:0] d2,
 input [7:0] d3,
 input [7:0] d4,
 output [15:0] dout1,
 output [15:0] dout2);

```

```
leaf u1 (
 .d1(d1),
 .d2(d2),
 .dout(dout1));
leaf u2 (
 .d1(d3),
 .d2(d4),
 .dout(dout2));
endmodule

//Configuration Definition
config cfg;
design work.top;
//default liblist xorlib multlib TOPLIB;
cell leaf use work.leaf;
endconfig

//Configuration Definition
config cfg1;
design work.top;
//default liblist xorlib multlib TOPLIB;
cell leaf use work.leaf;
endconfig
```

## Action

Make sure that only one configuration definition is defined as shown in the following example:

```
//Sub Module
module leaf (
 input [7:0] d1,
 input [7:0] d2,
 output reg [15:0] dout);

 always@*
 dout = d1 * d2;
 endmodule

//Top Module
module top (
 input [7:0] d1,
 input [7:0] d2,
 input [7:0] d3,
 input [7:0] d4,
 output [15:0] dout1,
 output [15:0] dout2);
```

```
leaf u1 (
 .d1(d1),
 .d2(d2),
 .dout(dout1));
leaf u2 (
 .d1(d3),
 .d2(d4),
 .dout(dout2));
endmodule

//Configuration Definition
config cfg;
design work.top;
//default liblist xorlib multlib TOPLIB;
cell leaf use work.leaf;
endconfig
```

## CG1023

### **@E: Unresolved expression has size 0.**

The expression has a size of 0 and cannot be resolved.

#### Action

The above error can occur when using the \$bits system function as in the case of \$bits(intf1.m) where intf1 is an interface instance and m is an object in the interface (\$bits cannot be used with an interface member). To avoid the error, use the actual value as the argument instead of the interface member.

# CG1072

## @E: Nested block comments.

Verilog language does not allow comments to be nested. When comments span multiple lines, use the conventions shown in the User Action section. The following test case includes a nested comment which causes the compiler to error out with the above message.

```
/* Simple flip-flop example/* without set
 or reset */ */
module dff(q, data, clk);
 output q;
 input data, clk;
 reg q;

 always @(posedge clk)
 begin
 q = data;
 end

endmodule
```

## Action

Make sure that comments are not nested. To eliminate the error in the above test case, either change the statement to single-line comments using // as shown in the test case below

```
// Simple flip-flop example without set
// or reset
module dff(q, data, clk);
 output q;
 input data, clk;
 reg q;

 always @(posedge clk)
 begin
 q = data;
 end

endmodule
```

or use a single multiple-line comment as shown below.

```
/* Simple flip-flop example without set
 or reset */
module dff(q, data, clk);
 output q;
 input data, clk;
 reg q;

 always @(posedge clk)
 begin
 q = data;
 end

endmodule
```

## CG1076

### @E: EOF found in translate off section

The synthesis translate\_off and translate\_on directives isolate non-synthesizable code from synthesizable code. In this case, the logic cannot be created due to the way translate\_on and translate\_off directives have been applied. In the following test case, no synthesis can be performed because the assign statement that creates the logic follows a translate\_off directive.

```
module real_time (ina, inb, out);
 input ina, inb;
 output out;
 /* synthesis translate_on */
 realtime cur_time;
 /* synthesis translate_off*/
 assign out = ina & inb;
endmodule
```

### Action

Make sure that the translate\_on and translate\_off synthesis directives are used correctly. In the above test case, the translate\_on and translate\_off directives are reversed. To eliminate the error in the above test case, change the order of the directives as shown in the corrected example below.

```
module real_time (ina, inb, out);
 input ina, inb;
 output out;
 /* synthesis translate_off */
 realtime cur_time;
 /* synthesis translate_on*/
 assign out = ina & inb;
endmodule
```

## CG1079

### **@E: translate\_on should follow translate\_off**

The synthesis translate\_off and translate\_on directives are used to isolate non-synthesizable code from synthesizable code. The compiler reports when a translate\_on directive precedes a translate\_off directive. In the following test case, the two translate\_on directives without an intervening translate\_off directive cause the compiler to error out with the above message.

```
module real_time (ina, inb, out);
 input ina, inb;
 output out;
 /* synthesis translate_on */
 realtime cur_time;
 /* synthesis translate_on */
 assign out = ina & inb;
endmodule
```

### Action

The synthesis translate\_off and translate\_on directives must be used in pairs with the translate\_on directive always following the translate\_off directive. To eliminate the error in the above test case, change the first translate\_on directive to a translate\_off directive as shown in the corrected test case below.

```
module real_time (ina, inb, out);
 input ina, inb;
 output out;
 /* synthesis translate_off */
 realtime cur_time;
 /* synthesis translate_on */
 assign out = ina & inb;
endmodule
```

## CG1086

### **@E: Expecting valid net type or name, found SystemVerilog keyword "*<int>*"**

SystemVerilog; either the keyword wire is not followed by a valid net type or the combination of the wire keyword and a valid net type is not followed by a variable name (a valid net type is any 4-state data type such as logic). In the following test case, the keyword wire is followed by int (a two-state data type) which causes the compiler to error out.

```
module my_and (c,a,b);
 input a,b;
 output c;
 wire int t1;
 assign t1 = !b;
 assign c= a & t1;
endmodule
```

### Action

Make sure that a valid net name is used and, when a net data type is specified, that it is a valid, 4-state data type. In the corrected test case below, the above two conditions are met.

```
module my_and (c,a,b);
 input a,b;
 output c;
 wire logic t1;
 assign t1 = !b;
 assign c= a & t1;
endmodule
```

## CG1119

### @E: Expecting exponent

An exponent in a scientific notation is not specified for a real number. In the case below, the missing exponent following the ‘e’ causes the compiler to error out with the above message.

```
module test_real();
real r;
initial
 r =2.2e;
endmodule
```

### Action

Make sure that when using scientific notation that an exponent is specified. To eliminate the syntax error in the above test case, include an exponent as shown in the corrected test case below.

```
module test_real();
real r;
initial
 r =2.2e6;
endmodule
```

## CG1120

### @E: Found unsized single bit literal in Verilog-2001 mode.

Verilog-2001 mode; an unsized single-bit literal was encountered without a base specifier (specifying an unsized literal as a single-bit value with a preceding apostrophe is only supported in SystemVerilog). In the test case below, the compiler encounters unsized literal ‘1 (in Verilog-2001 mode) which causes the compiler to error out with this message.

```
module cg1120(a,b,c);
 input a, b;
 output c;
 assign c= a?b:'1;
endmodule
```

## Action

Unsized numbers in Verilog-2001 are interpreted as follows:

- Numbers without a base format specified default to a decimal number (for example, 1234)
- The number of bits for a number with a base format specified, but without a size specified is machine dependent (for example, 'hc3, 'b11, or 'o56)

To specify an unsized number in Verilog-2001, you must use one of the above two methods. In the corrected test case below, a base format is added to the unsized literal ('b1) to eliminate the error.

```
module cg1120(a,b,c);
 input a, b;
 output c;
 assign c= a?b:'b1;
endmodule
```

## CG1137

### **@E: Black box encountered in resolving XMR. Could not resolve '<bb\_signal>'**

An attempt was made to hierarchically access a variable from an undefined or black-boxed module. In the test case below, the error is the result of the type referenced during the instantiation of module sub.

```
module sub (input a, output b) ;
 logic bb_signal = a;
 assign b = a;
endmodule
```

```
module top (input a, output b1, b2);
 assign b2 = UUT.bb_signal;
 sub1 UUT(a, b1);
endmodule
```

## Action

Avoid hierarchically accessing variables in undefined or black-box modules.

```
module sub (input a, output b) ;
 logic bb_signal = a;
 assign b = a;
endmodule

module top (input a, output b1, b2);
 assign b2 = UUT.bb_signal;
 sub UUT(a, b1);
endmodule
```

# CG1144

## **@E: Encountered SystemVerilog keyword "<end>" when expecting a parameter name.**

A SystemVerilog keyword is missing. In the test case below, a corresponding endcase statement was omitted following the default case entry which results in the error.

```
module top(
 input logic[2:1] sel,
 input byte d1[4:1],
 output byte q);

 always_comb begin
 case(sel)
 0: q = d1[1];
 1: q = d1[2];
 2: q = d1[3];
```

```
 3: q = d1[4];
 default : q = '1;
 end
end
endmodule
```

## Action

Make sure that all constructs are syntactically correct and complete.

```
module top(
 input logic[2:1] sel,
 input byte d1[4:1],
 output byte q);

 always_comb begin
 case(sel)
 0: q = d1[1];
 1: q = d1[2];
 2: q = d1[3];
 3: q = d1[4];
 default : q = '1;
 endcase
 end
endmodule
```

# CG1145

## **@E: Encountered Verilog 2005 keyword "<case>" when expecting a parameter name**

A Verilog 2005 (or SystemVerilog) keyword was used as a parameter name. In the test case below, Verilog keyword `case` is used as a parameter name which causes the error.

```
module top(
 sel,d1,d2,d3,d4,q);
parameter case = 1'bl;
input wire[case:1] sel;
input wire [7:0]d1,d2,d3,d4;
output reg [7:0] q;
```

```
always@* begin
 case(sel)
 0: q = d1;
 1: q = d2;
 2: q = d3;
 3: q = d4;
 default : q = 8'hFF;
 endcase
end
endmodule
```

## Action

Avoid using Verilog keywords in parameter names.

```
module top(
 sel,d1,d2,d3,d4,q);
parameter casel = 1'b1;
input wire[casel:1] sel;
input wire [7:0]d1,d2,d3,d4;
output reg [7:0] q;

always@* begin
 case(sel)
 0: q = d1;
 1: q = d2;
 2: q = d3;
 3: q = d4;
 default : q = 8'hFF;
 endcase
end
endmodule
```

# CG1146

## **@E: Encountered Verilog 1995 keyword "<wire>" when expecting a parameter name.**

A Verilog 1995 keyword was used as a parameter name. In the test case below, Verilog keyword `wire` is used as a parameter name which results in the error.

```
module top(
 sel,d1,d2,d3,d4,q);
parameter wire = 1'b1;
input wire[wire:1] sel;
input wire [7:0]d1,d2,d3,d4;
output reg [7:0] q;

always@* begin
 case(sel)
 0: q = d1;
 1: q = d2;
 2: q = d3;
 3: q = d4;
 default : q = 8'hFF;
 endcase
end
endmodule
```

## Action

Avoid using Verilog keywords in parameter names.

```
module top(
 sel,d1,d2,d3,d4,q);
parameter wr = 1'b1;
input wire[wr:1] sel;
input wire [7:0]d1,d2,d3,d4;
output reg [7:0] q;

always@* begin
 case(sel)
 0: q = d1;
 1: q = d2;
 2: q = d3;
 3: q = d4;
 default : q = 8'hFF;
 endcase
end
endmodule
```

# CG1150

## @E: Could not resolve hierarchical reference '<su.sig>'.

An attempt was made to access a variable hierarchically and the hierarchy was incorrect. In the test case below, su in the assign statement is not referenced in either submodule which causes the error.

```
module sub (input byte d1,d2,output byte q1);
byte sig;
assign q1 = d1 ^ d2;
endmodule

module sub1 (input byte d1,d2,output byte q1 ,q2);
sub sub (.*);
endmodule

module top(input byte d1,d2,output byte q1 ,q2 ,q3);
sub1 sub1 (.*);
assign q3 = sub1.su.sig;
endmodule
```

## Action

Make sure that the correct hierarchical access is specified.

```
module sub (input byte d1,d2,output byte q1);
byte sig;
assign q1 = d1 ^ d2;
endmodule

module sub1 (input byte d1,d2,output byte q1 ,q2);
sub sub (.*);
endmodule

module top(input byte d1,d2,output byte q1 ,q2 ,q3);
sub1 sub1 (.*);
assign q3 = sub1.sub.sig;
endmodule
```

# CG1157

## **@E: Direct import of variable <p1> from package <pack2> conflicts with direct import from package <pack1>**

An attempt was made to import the same parameter/variable from two or more different packages. In the test case below, parameter p1 is assigned a different value in two separate packages which results in the error.

```
package pack1;
 parameter p1 = 23;
endpackage

package pack2;
 parameter p1 = 34;
endpackage

module top(
 input byte d1,
 output byte q1);
 import pack1::p1;
 import pack2::p1;
 assign q1 = d1 + p1;
endmodule
```

### Action

Make sure that there is no duplication of parameter/variable assignments in packages.

```
package pack1;
 parameter p1 = 23;
endpackage

package pack2;
 parameter p1 = 34;
endpackage

module top(
 input byte d1,
 output byte q1);
 import pack1::p1;
 assign q1 = d1 + p1;
endmodule
```

## CG1163

### **@A: Use of old pattern assignment style restricted to support for default; please use '{...}'**

Pattern assignment was performed using Accellera format instead of an aggregate operator (support for Accellera pattern assignment is restricted).

```
module top(
 output byte dout[2:0]);
 assign dout = {default:1};
endmodule
```

#### Action

Avoid using Accellera format.

```
module top(
 output byte dout[2:0]);
 assign dout = '{default:1}';
endmodule
```

## CG1168

### **@E: Illegal defparam. Parameter <p2> cannot be found in module <sub>.**

An attempt was made to override a parameter that was not defined in a module.

```
module sub#(
 parameter p1 = 2)
 (input d1,d2,
 output dout);
 assign dout = d1^d2;
endmodule
```

```
module top(
 input d1,d2,
 output dout);
 defparam u1.p1 = 3;
 defparam u1.p2 = 3;
 sub u1 (d1,d2,dout);
endmodule
```

## Action

Remove the undefined parameter from the module.

```
module sub#(
 parameter p1 = 2)
 (input d1,d2,
 output dout);
 assign dout = d1^d2;
endmodule

module top(
 input d1,d2,
 output dout);
 defparam u1.p1 = 3;
 sub u1 (d1,d2,dout);
endmodule
```

# CG1169

**@E: Could not find port for expanded interface <intf[1]> - possible invalid interface port connection.**

The array of interface instance instantiated during port mapping does not match the module definition of the array of interface port.

```
interface intf();
 byte i1;
endinterface
```

```
module sub(
 intf intf[1:0],
 input byte d1);
assign intf[1].il = d1;
endmodule

module top (
 input byte d1,
 output byte dout1);
intf intf[3:0]();
sub u1 (intf,d1);
assign dout1 = intf[1].il;
endmodule
```

## Action

Make sure that the definition of array of interface is the same in both the module port and instantiated module.

```
interface intf();
 byte il;
endinterface

module sub(
 intf intf[1:0],
 input byte d1);
assign intf[1].il = d1;
endmodule

module top (
 input byte d1,
 output byte dout1);
intf intf[1:0]();
sub u1 (intf,d1);
assign dout1 = intf[1].il;
endmodule
```

## CG1171

### **@E: Unexpected runtime condition in SystemVerilog Interface support**

An unhandled exception in SystemVerilog Interface support was encountered.

#### Action

Please contact technical support. Include a copy of the project file, log file and, if possible, the test case.

## CG1176

### **@E: At least one procedural block write and one continuous assign on bit <0> of <temp>**

At least one bit of the defined signal has been assigned both procedurally and continuously (multiple assignments to the same signal are illegal). In the test case below, signal temp is continuously and procedurally assigned which results in the error.

```
module top (input [7:0] i1,i2, output logic [7:0] o1);
generate
 begin:G1
 logic [7:0] temp;
 always_comb
 temp[0] = i1;
 end
 endgenerate

 assign G1.temp = i2;
 assign o1 = G1.temp;
endmodule
```

## Action

Make sure that only one valid assignment is made for each signal. To eliminate the error in the above test case, avoid multiple assignments to `temp[0]` by assigning only to `temp[7:6]` in the continuous assignment.

```
module top (input [7:0] i1,i2, output logic [7:0] o1);
generate
 begin:G1
 logic [7:0] temp;
 always_comb
 temp[0] = i1;
 end
 endgenerate
 assign G1.temp[7:6] = i2;
 assign o1 = G1.temp;
endmodule
```

## CG1183

**@E: DPI function linkage name conflict. Signatures for two functions with global name '<myfunc2>' do not match.**

The compiler encountered two functions with different signatures, but the same linkage name. In the following test case, export functions `myfunc2` and `myfunc` have the same linkage name (`myfunc2`) which results in the error.

```
module test(input clk, rst);
export "DPI-C" function myfunc2;
function shortint myfunc2(input int a, input byte b,
 input integer c);
begin
 c = a+b;
 return a;
end
endfunction
```

```

export "DPI-C" myfunc2=function myfunc;
function shortint myfunc(input logic[31:0] a, output byte b);
 b = a;
 return a;
endfunction
endmodule

```

## Action

According to the LRM, two DPI functions with different signatures are not allowed to have the same linkage name. This same rule also applies to the import functions. To compile the above design, change the linkage name of one of the functions.

```

module test (input clk,rst);
 export "DPI-C" function myfunc2;
 function shortint myfunc2(input int a, input byte b,
 input integer c);
 begin
 c = a+b;
 return a;
 end
 endfunction

 export "DPI-C" myfunc=function myfunc;
 function shortint myfunc(input logic[31:0] a, output byte b);
 b = a;
 return a;
 endfunction
endmodule

```

## CG1184

**@E: DPI function global name conflict. Functions with global name '*<myfunc2>*' declared with both old SCE-MI syntax and SV SCE-MI syntax.**

Two functions with the same linkage name were encountered that had a different SCE-MI syntax. In the following test case, export functions myfunc2 in module sub1 uses the SystemVerilog SCE-MI syntax while export function

myfunc2 in module sub2 uses the old Verilog style SCE-MI syntax. The mixing of SCE-MI syntax styles is not allowed for any two functions having the same linkage name. This same rule applies for import functions.

```

module sub1(input clk, rst);
 export "DPI-C" function myfunc2; //SV SCE-MI syntax
 function shortint myfunc2(input int a, input byte b,
 output integer c);
 begin
 c = a+b;
 return a;
 end
 endfunction
 endmodule

module sub2(input clk, rst);
 //Old SCE-MI syntax
 (*sce_mi="export DPI-C") function shortint myfunc2
 (input int a, input byte b, output integer c);
 begin
 c = a+b;
 return a;
 end
 endfunction
endmodule

module top (input clk,rst,input int a,b, output integer c,d);
 sub1 i0 (clk,rst);
 sub2 i1 (clk,rst);
endmodule

```

## Action

To eliminate the error in the above test case, be sure to use the same SCE-MI syntax for all export functions with the same linkage name. In the corrected test case below, the SCE-MI syntax for export function myfunc2 in module sub2 is changed to the SystemVerilog SCE-MI syntax.

```

module sub1(input clk, rst);
 export "DPI-C" function myfunc2; //SV SCE-MI syntax
 function shortint myfunc2(input int a, input byte b,
 output integer c);
 begin

```

```
 c = a+b;
 return a;
 end
endfunction
endmodule

module sub2(input clk,rst);
export "DPI-C" function myfunc2; //SV SCE-MI syntax
 function shortint myfunc2(input int a, input byte b,
 output integer c);
 begin
 c = a+b;
 return a;
 end
endfunction
endmodule

module top (input clk,rst,input int a,b, output integer c,d);
sub1 i0 (clk,rst);
sub2 i1 (clk,rst);
endmodule
```

## CG1185

### **@E: DPI function global name conflict. Global Name '<*myfunc2*>' used for both import and export functions.**

The compiler encountered an import function and an export function with the same linkage name. It is illegal for import and export functions to share the same linkage name. In the following test case, export function myfunc2 in module sub1 and import function myfunc2 in the module sub2 have the same linkage name which results in the error.

```
module sub1(input clk, rst);
export "DPI-C" function myfunc2;
 function shortint myfunc2(input int a, input byte b,
 output integer c);
 begin
```

```
 c = a+b;
 return a;
 end
endfunction
endmodule

module sub2(input int a,b,output integer c,d);
import "DPI-C" function shortint myfunc2(input int a,
 input byte b, output integer c);

always @*
 d = myfunc2(a,b,c);
endmodule

module top (input clk,rst,input int a,b, output integer c,d);
sub1 i0 (clk,rst);
sub2 i1 (a,b,c,d);
endmodule
```

## Action

Change the linkage name of one of the functions so that they do not share the same linkage name. In the corrected test case below, the linkage name for export function `myfunc2` in module `sub1` is changed to `myfunc`.

```
module sub1(input clk, rst);
export "DPI-C" myfunc=function myfunc2;
 function shortint myfunc2(input int a, input byte b,
 output integer c);
 begin
 c = a+b;
 return a;
 end
 endfunction
endmodule

module sub2(input int a,b,output integer c,d);
import "DPI-C" function shortint myfunc2(input int a,
 input byte b, output integer c);

always @*
 d = myfunc2(a,b,c);
endmodule
```

```
module top (input clk,rst,input int a,b, output integer c,d);
sub1 i0 (clk,rst);
sub2 il (a,b,c,d);
endmodule
```

## CG1187

### **@E: Top-level array of interface port ('<intf>') not supported for synthesis.**

An array of interface is being incorrectly used in the top-level module.

```
interface intf();
byte il;
assign il = 8'd12;
endinterface

module sub2(
 intf intf[1:0],
 output byte dout1,dout2);
assign dout1 = intf[1].il;
assign dout2 = intf[0].il;
endmodule
```

### Action

Avoid using an array of interface as the top-level module. In the corrected test case below, an additional layer (top) is added which results in a normal instantiation.

```
interface intf();
byte il;
assign il = 8'd12;
endinterface

module sub2(
 intf intf[1:0],
 output byte dout1,dout2);
assign dout1 = intf[1].il;
assign dout2 = intf[0].il;
endmodule
```

```
module top(
 input byte d1,d2,
 output byte dout1,dout2);
intf intf[1:0]();
sub2 u2 (intf,dout1,dout2);
endmodule
```

## CG1192

**@E: Invalid character <@> found in file <*mem.ini*> for task <\$readmem> - only one '@' character allowed in address specifier"**

More than one @ character was used within an address entry in a memory initialization file (only a single @ character can precede a hex address entry). In the mem.ini file specified in the include path, the memory address entry for address 0a includes a second (double) @ character which results in the error.

```
/* mem.ini */
@00 37
@01 59
@02 91
@03 1e
@04 61
@05 0B
@06 1F
@07 43
@08 91
@09 40
@@0a ae
@0B 34
@0c 36
@0d 34
@0e B8
@0F 6e
```

### Action

Remove the extra @ character from the 0a address entry in the memory initialization file. Rerunning the example below with the corrected ini file eliminates the error.

```
/* ROM code */
`define width_ROM1 8
`define depth_ROM 256
`ifdef synthesis

module top
`else
module top_rtl
`endif
(
 input clk,
 input RD_ROM,
 input [0:($clog2(`depth_ROM))-1] Addr_ROM,
 output reg[0:`width_ROM1-1]Dout_ROM1);
 reg[0:`width_ROM1-1]mem_rom1[0:`depth_ROM-1];

initial
begin
 $readmemh("mem.ini",mem_rom1);
end

always@(posedge clk)
begin
 if(RD_ROM)
 begin
 Dout_ROM1 <= mem_rom1[Addr_ROM];
 end
 end
endmodule
```

## CG1197

**@E: Pipe function <*scemi\_pipe\_hdl\_receive*> can only be called once per pipe (1).**

One of the following SCE-MI Verilog pipe functions/tasks was called more than once for the same pipe ID:

---

scemi_pipe_hdl_receive	scemi_pipe_hdl_try_receive
scemi_pipe_hdl_send	scemi_pipe_hdl_try_send
scemi_pipe_hdl_flush	scemi_pipe_hdl_try_flush

Calling the same function/task for a single pipe ID is not supported.

```
module test(output reg[31:0] data,data1,output reg eom,
 eom1,input clk);
parameter pipe_id = 1;
`include "scemi_pipes.vh"

always @(posedge clk)
begin
 scemi_pipe_hdl_receive(pipe_id,1,1,1,data,eom);
end

always @(posedge clk)
begin
 scemi_pipe_hdl_receive(pipe_id,1,1,1,data1,eom1);
end
endmodule
```

## Action

In the above case, pipe function scemi\_pipe\_hdl\_receive is called twice for pipe ID 1. In the corrected case below, a second pipe\_id parameter, pipe\_id1, is created to ensure that the SCE-MI function is called only once for the pipe ID.

```
module test(output reg[31:0] data,data1,output reg eom,
 eom1,input clk);
parameter pipe_id = 1;
parameter pipe_id1 = 2;
`include "scemi_pipes.vh"

always @(posedge clk)
begin
 scemi_pipe_hdl_receive(pipe_id,1,1,1,data,eom);
end
```

```

always @(posedge clk)
begin
 scemi_pipe_hdl_receive(pipe_id1,1,1,1,data1,eom1);
end
endmodule

```

## CG1198

**@E: Only one of 'scemi\_pipe\_hdl\_send' and 'scemi\_pipe\_hdl\_try\_send' can be called on output pipe (1)**

The SCE-MI Verilog pipe subroutine calls for scemi\_pipe\_hdl\_send and scemi\_pipe\_hdl\_try\_send are using the same pipe ID.

```

module test(input [31:0] data,data1,input eom,eom1,input clk);
`include "scemi_pipes.vh"
parameter pipe_id = 1;

always @(posedge clk)
begin
 scemi_pipe_hdl_send(pipe_id,1,1,data,eom);
end

always @(posedge clk)
begin
 scemi_pipe_hdl_try_send(pipe_id,1,1,1,data1,eom1);
end
endmodule

```

### Action

In the above case, the pipe functions scemi\_pipe\_hdl\_send and scemi\_pipe\_hdl\_try\_send are both called with the same pipe ID. In the corrected test case below, a second pipe ID parameter (pipe\_id1) is created for the scemi\_pipe\_hdl\_try\_send function.

```

module test(input [31:0] data,data1,input eom,eom1,input clk);
`include "scemi_pipes.vh"
parameter pipe_id = 1;
parameter pipe_id1 = 2;

```

```
always @(posedge clk)
begin
 scemi_pipe_hdl_send(pipe_id,1,1,data,eom);
end

always @(posedge clk)
begin
 scemi_pipe_hdl_try_send(pipe_id1,1,1,1,data1,eom1);
end
endmodule
```

## CG1200

**@E: No port with name 'bytes\_per\_element' found on Verilog scemi pipe <pipeType>**

The scemi\_pipes.vh file is in use, but the bytes\_per\_element port has been removed.

```
task scemi_pipe_hdl_receive;
 input integer pipe_id;
 input integer num_elements;
 input integer num_elements_valid;
 input [PAYLOAD_MAX_BITS-1:0] data;
 input eom;
 begin
 end
endtask
```

Action:

Include the bytes\_per\_element port.

```
task scemi_pipe_hdl_receive;
 input integer pipe_id;
 input integer bytes_per_element;
 input integer num_elements;
 input integer num_elements_valid;
```

```

 input [PAYLOAD_MAX_BITS-1:0] data;
 input eom;
begin
end
endtask

```

## CG1201

**@E: Port 'bytes\_per\_element' on Verilog scemi pipe '*<scemi\_pipe\_hdl\_send>*' must be tied to a constant**

One of the following SCE-MI Verilog pipe subroutines has the port bytes\_per\_element connected to a non-constant value:

scemi_pipe_hdl_receive	scemi_pipe_hdl_try_receive
scemi_pipe_hdl_send	scemi_pipe_hdl_try_send

In the below test case, the bytes\_per\_element port of pipe function scemi\_pipe\_hdl\_send is connected to an internal signal which may not be of a constant type.

```

module test(input [31:0] data,data1,input eom,eom1,
 input clk, output integer c);
`include "scemi_pipes.vh"
integer bytes_per_elem = 1;
parameter pipe_id1 = 1;

always @(posedge clk)
begin
 scemi_pipe_hdl_send(pipe_id1,bytes_per_elem, 1,data,eom);
end
endmodule

```

### Action

Make sure that the bytes\_per\_element is a constant. In the corrected test below, changing bytes\_per\_element from an internal signal to a parameter eliminates the error.

```
module test(input [31:0] data,data1,input eom,eom1,input clk,
 output integer c);
`include "scemi_pipes.vh"
parameter bytes_per_elem = 1;
parameter pipe_id1 = 1;

always @(posedge clk)
begin
 scemi_pipe_hdl_send(pipe_id1,bytes_per_elem, 1,data,eom);
end
endmodule
```

## CG1202

**@E: No port with name 'pipe\_id' found on Verilog scemi pipe <pipeType>**

The scemi\_pipes.vh file is in use, but the pipe\_id port has been removed.

```
task scemi_pipe_hdl_receive;
 input integer bytes_per_element;
 input integer num_elements;
 input integer num_elements_valid;
 input [PAYLOAD_MAX_BITS-1:0] data;
 input eom;
begin
end
endtask
```

Action:

Include the pipe\_id port.

```
task scemi_pipe_hdl_receive;
 input integer pipe_id;
 input integer bytes_per_element;
 input integer num_elements;
 input integer num_elements_valid;
```

```

 input [PAYLOAD_MAX_BITS-1:0] data;
 input eom;
begin
end
endtask

```

## CG1203

### **@E: Port 'pipe\_id' on Verilog scemi pipe '<scemi\_pipe\_hdl\_send>' must be tied to a constant**

One of the following SCE-MI Verilog pipe subroutines had the port pipe\_id connected to a non-constant value:

scemi_pipe_hdl_receive	scemi_pipe_hdl_try_receive
scemi_pipe_hdl_send	scemi_pipe_hdl_try_send
scemi_pipe_hdl_can_send	scemi_pipe_hdl_can_receive
scemi_pipe_hdl_flush	scemi_pipe_hdl_try_flush

In the below test case, the pipe\_id port of the scemi\_pipe\_hdl\_send pipe function is connected to an internal signal that is not of a constant type.

```

module test(input [31:0] data,data1,input eom,eom1,input clk);
`include "scemi_pipes.vh"
integer pipe_id = 1;

always @(posedge clk)
begin
 scemi_pipe_hdl_send(pipe_id,1,1,data,eom);
end
endmodule

```

### Action

Make sure that the pipe\_id is a constant. In the corrected test below, changing bytes\_per\_element from an internal signal to a parameter eliminates the error.

```
module test(input [31:0] data,data1,input eom,eom1,input clk);
`include "scemi_pipes.vh"
parameter pipe_id = 1;

always @(posedge clk)
begin
 scemi_pipe_hdl_send(pipe_id,1,1,data,eom);
end
endmodule
```

## CG1204

### **@E: Generate if condition expression did not evaluate to a constant**

A generate if condition includes an input expression that is dynamically defined (the check for conditionality must be a constant value).

```
module top(
 input sel,
 input byte d1,
 input byte d2,
 output byte dout1);

generate begin
 if(sel)
 assign dout1 = d1;
 else
 assign dout2 = d2;
 end
endgenerate
endmodule
```

### Action

Make sure that constant values are used in generate if condition statements. In the corrected example below, parameter is used to configure the module.

```
module top# (
 parameter p1 = 1) (
 input sel,
 input byte d1,
 input byte d2,
 output byte dout1);

 generate begin
 if(p1==1)
 assign dout1 = d1;
 else
 assign dout2 = d2;
 end
 endgenerate
endmodule
```

## CG1205

### **@E: Net data types must be 4-state values**

A 2-state data type is incorrectly defined as a member of a structure when the structure data type is explicitly defined as a wire.

```
typedef struct packed
{
 byte st2;
}structdt;

module top(
 input wire structdt d1,
 output bit dout1);
 assign dout1 = d1;
endmodule
```

### Action

The above error is avoided by defining the member of a structure of a 4-state data type when the structure data type is explicitly defined as a wire.

```

typedef struct packed
{
 logic[8:1] st2;
}structdt;

module top(
 input wire structdt d1,
 output bit dout1);
assign dout1 = d1;
endmodule

```

## CG1207

### **@E: always block: miscounted events**

An always block is used for behavioral coding in continue-on-error mode. In the test case below, behavioral coding is enabled by the `always@(*)` statement which is not supported for synthesis.

```

module CG1207 (input clk, din, sel, output reg dout);
always@(*)
 for (int i=0; i< 5; i++)
 @(posedge clk)
 dout = din;
endmodule

```

### Action

To avoid the error in the above test case, specify a sensitivity list by replacing `always@(*)` with `alway@(clk)`.

```

module CG1207 (input clk, din, sel, output reg dout);
always@(clk)
 for (int i=0; i< 5; i++)
 @(posedge clk)
 dout = din;
endmodule

```

## CG1208

### **@E: Unexpected runtime condition while synthesizing instance <instance>**

An error occurred while synthesizing a SystemVerilog interface as a result of either:

- the synthesis of the interface instance could not be completed
- an error was detected while generating the internal data required by the interface client.

## CG1210

### **@E: Expecting a flattened concatenation expression**

An attempt was made to extract a slice or index from an unhandled concatenation expression. Concatenation expressions are flattened prior to extraction to avoid having to track offsets of internal expressions.

## CG1211

### **@E: Second argument for \$readmemb must be a memory in the current module (a hierarchical reference is not supported).**

The memory to be initialized by the \$readmemb/\$readmemh task is being accessed hierarchically (hierarchical references are not supported).

```
module sub(
 output byte dout1 [1:0]);
 byte sig1[1:0];
 assign dout1 = sig1;
endmodule
```

```
module sub1(
 output byte dout1 [1:0],
 output byte dout2);
sub ul(.*);
assign dout2 = 8'd23;
endmodule

module top(
 input byte d1,
 output byte dout1 [1:0],
 output byte dout2,
 output byte dout3);
subl ul(.*);
initial
begin
 $readmemb ("mem.ini", ul.ul.sig1,0,1);
end
assign dout3 = d1;
endmodule
```

## Action

Make sure that the memory is initialized only in the local module (initialization support for memory elements is limited to the local module).

```
module sub(
 output byte dout1 [1:0]);
byte sig1[1:0];
initial
begin
 $readmemb ("mem.ini", sig1, 0, 1);
end
assign dout1 = sig1;
endmodule

module sub1(
 output byte dout1 [1:0],
 output byte dout2);
sub ul(.*);
assign dout2 = 8'd23;
endmodule
```

```
module top(
 input byte d1,
 output byte dout1 [1:0],
 output byte dout2,
 output byte dout3);
 subl u1(.*);
 assign dout3 = d1;
endmodule
```

## CG1212

### **@E: Unexpected runtime condition: unresolved interface or type for signal <sigName>**

An unexpected condition occurred during compilation when the interface or type for a signal could not be resolved.

#### Action

Please contact Synopsys support; if possible, please include the test case that resulted in the error.

## CG1213

### **@E: Could not resolve type operator**

A SystemVerilog type operator with complex expressions was encountered (complex expressions are not supported in type operators). In the test case below, a complex expression ( $a*b$ ) is used as an argument for the SystemVerilog type-operator.

```
module CG1213 (input [15:0] a, b, output type(a*b) dout);
 assign dout = a * b;
endmodule
```

## Action

Use an alternate method of meeting the design requirements. In the test case below, a user-defined datatype (MyDT) is first defined and then used in the type operator.

```
typedef logic [31:0] MyDt;
module CG1213 (input [15:0] a, b, output type(MyDt) dout);
 assign dout = a * b;
endmodule
```

# CG1214

## **@E: Internal error. Value instof not set for instance <instName> of module <modName> in library <libName>**

An internal error occurred during compilation when an instof value could not be set for the instance.

## Action

Please contact Synopsys support; if possible, please include the test case that resulted in the error.

# CG1217

## **@E: Expecting single-bit output for built-in Verilog gate**

A built-in Verilog gate such as a bufif1, bufif0, notif0, or notif1 is incorrectly connected to a vectored output (built-in Verilog gates expect scalar outputs). In the test case below, Verilog built-in gate bufif1 is connected to 8-bit output port out.

```
module CG1217 (input [7:0] in, sel, output [7:0] out);
 bufif1 b1 (out, in, sel);
endmodule
```

## Action

Use an array of instances to instantiate the built-in Verilog gate eight times as shown in the corrected test case below.

```
module CG1217 (input [7:0] in, sel, output [7:0] out);
 bufif1 b1 [7:0] (out, in, sel);
endmodule
```

# CG1218

### **@E: Cannot resolve type operator**

A SystemVerilog type operator is being incorrectly used to directly assign a value to a variable. In the test case below, a SystemVerilog type operator is used to directly assign output dout.

```
module CG1218 (output dout);
 assign dout = type(logic);
endmodule
```

## Action

The SystemVerilog type operator cannot be used in the above context. The type operator can be used for data declaration, data-type declaration, type casting, comparison and case comparison. The test case below, uses a SystemVerilog type operator for type casting.

```
module CG1218 #(parameter type MyOutType = int)
 (input [15:0] a, b, output MyOutType dout);
 assign dout = type(dout)'(a) * type(dout)'(b);
endmodule
```

## CG1219

### **@E: Unexpected type for generate block**

An unexpected condition occurred during compilation when the type for a generate block could not be resolved.

#### Action

Please contact Synopsys support; if possible, please include the test case that resulted in the error.

## CG1220

### **@E: Cannot determine type of variable**

An undefined signal or datatype is being used in a type operator. In the test case below, undefined variable aa is used in the type operator which results in the error.

```
module CG1220 (input [15:0] a, b, output type(aa) dout);
 assign dout = a | b;
endmodule
```

#### Action

Make sure that all type operators are defined. In the test case below, correcting the typographical error “aa” to the intended “a” resolves the error.

```
module CG1220 (input [15:0] a, b, output type(a) dout);
 assign dout = a | b;
endmodule
```

## CG1221

### **@E: Unexpected ‘[‘, expecting a variable or identifier**

An unexpected condition occurred during compilation when an unexpected character was found.

#### Action

Please contact Synopsys support; if possible, please include the test case that resulted in the error.

## CG1222

### **@N: Previous declaration of module <*moduleName*> found**

An earlier module declaration of the same name has been encountered in a design that allows duplicate modules. The source location in the Messages window points to the first occurrence of the duplicate module name.

## CG1223

### **@N: Setting “`set_option -dup {1}`” in project file allows duplicate modules**

A duplicate module declaration with the same name was encountered. This message only occurs when duplicate modules are not allowed and is a reminder to include a `set_option -dup` command with a value of 1 in the project file (or to check Allow Duplicate Modules on the Verilog tab) if the intention is to allow duplicate modules.

## CG1225

### **@E: Unexpected expression: expecting an array index into a structure or multi-dimensional array access**

An unexpected condition occurred during compilation when an array index or multi-dimensional array access could not be resolved.

#### Action

Please contact Synopsys support; if possible, please include the test case that resulted in the error.

## CG1228

### **@E: Could not find variable <*sig1.nu1.nu*> - possible illegal field access of union**

An attempt was made to access the member of a non-existent union. In the test case below, variable sig1.nu1.nu is not correctly defined by the union data type which results in the error.

```
typedef union packed {
 union packed {
 union packed {
 logic[4:1][2:1] u1;
 }nul;
 byte u2;
 }nul;
}dt;

module sub(
 input byte d1,
 output byte dout1);
dt sig1;
assign sig1.nul.nu.u1 = d1;
assign dout1 = sig1.nul.nul.ul;
endmodule
```

## Action

Make sure that the definition and assignment match as shown in the corrected test case below.

```
typedef union packed {
 union packed {
 union packed {
 logic[4:1][2:1] u1;
 }nul;
 byte u2;
 }nul;
}dt;

module sub(
 input byte d1,
 output byte dout1);
dt sig1;
assign sig1.nul.nul.ul = d1;
assign dout1 = sig1.nul.nul.ul;
endmodule
```

# CG1230

## **@E: Synthesis of trireg variables not supported**

Net-type variables declared as trireg are not supported in Verilog.

## Action

Use a wire declaration in place of the trireg construct.

# CG1232

## @E: defparam associated with array of interfaces '<II[1]>' not supported.

A parameter definition (defparam) was incorrectly defined for an array of instances (defparams are not supported with arrays of interfaces).

```
interface leaf #(parameter CNST = 2)();
 logic[3:0] tmp1;
 logic[7:0] tmp2;

 function automatic logic[7:0] myadd(input[3:0] in1);
 myadd = (in1*CNST);
 endfunction
endinterface

module subsub(leaf kk);
 assign kk.tmp2 = kk.myadd(kk.tmp1);
endmodule

module sub #(parameter WIDTH = 2)(leaf jj[WIDTH-1:0]);
 subsub S1(jj[1]);
endmodule

module tst (input logic[3:0] din, output logic[7:0] dout);
 leaf II[3:0]();
 defparam II[1].CNST = 8;

 assign II[1].tmp1 = din;
 sub #(4) S0(II);
 assign dout = II[1].tmp2;
endmodule
```

## Action

Removing (or commenting out) the defparam statement eliminates the error.

```
interface leaf #(parameter CNST = 2)();
 logic[3:0] tmp1;
 logic[7:0] tmp2;

 function automatic logic[7:0] myadd(input[3:0] in1);
 myadd = (in1*CNST);
 endfunction
endinterface
```

```

module subsub(leaf kk);
 assign kk.tmp2 = kk.myadd(kk.tmp1);
endmodule

module sub #(parameter WIDTH = 2)(leaf jj[WIDTH-1:0]);
 subsub S1(jj[1]);
endmodule

module tst (input logic[3:0] din, output logic[7:0] dout);
 leaf II[3:0]();
 // defparam II[1].CNST = 8;

 assign II[1].tmp1 = din;
 sub #(4) S0(II);
 assign dout = II[1].tmp2;
endmodule

```

## CG1235

### **@E: Inside operator not allowed with casex or casez**

A casex or casez statement incorrectly included an inside operator (only the case statement supports the inside operator).

```

module top# (
parameter byte p1[2:1][4:1] = '{'{'{0,2,4,6},'{1,3,5,7}}')
//Input
(input logic[4:1]sel,a,b,
//Output
output logic[3:1] q);

always_comb begin
casez (sel) inside
 8,p1[1],10,12,14:q <= a;
 p1[2],9,11,13,15:q <= b;
endcase
end
endmodule

```

### Action

Either use a case statement (and avoid using “don’t care/high impedance” states) or remove the inside operator.

## CG1237

### **@W: System task \$isunknown() returns false - simulation mismatch possible**

An \$isunknown operator was encountered (\$isunknown is currently unsupported). The associated expression always evaluates false; the message text indicates a possible simulation mismatch.

### Action

As many designs for prototyping may include the \$isunknown operator within assertion statements, be aware of the potential for simulation mismatches.

## CG1238

### **@E: Unable to flatten expression**

The Verilog compiler could not handle a concatenation operation. An example is shown below:

```
typedef struct {
 logic [3:0] alen;
} ach_t;
module top (input clk,
 input reset,
 input ach_t data[1:0],
 output byte data_out);

 ach_t datas_out[1:0];
```

```
dummy_inst u_inst[1:0] (.clk(clk),
 .reset(reset),
 .data(data),
 .data_out({datas_out[1].alen,datas_out[0].alen})); // Not
 supported

assign data_out = datas_out;
endmodule

module dummy_inst (input clk,
input reset,
input ach_t data[1:0],
output logic [3:0] data_out);
 assign data_out = data[0].alen;
endmodule
```

## Action

Please contact Synopsys support. This error occurs in releases prior to the I-2013.09-SP1 release.

# CG1240

## **@E: Using defparams across the hierarchy is not supported. Use inline-parameter assignment.**

defparam is used to override default parameter values. The synthesis tool errors out if the defparam statement is used to hierarchically change the values of parameters of more than one level as shown in the example below:

```
module multiplier (operand1, operand2, result);
parameter operand1_width = 4, operand2_width =2;
input [operand1_width -1:0]operand1;
input [operand2_width -1:0] operand2;
output [operand1_width +operand2_width -2:0] result;
assign result = operand1 * operand2;
endmodule
```

```

module level2(op1,op2,result);
defparam test2.operand1_width =16; // This is supported
defparam test2.operand2_width =8; // This is supported
input [15:0] op1;
input [7:0] op2;
output [22:0]result;
multiplier test2 (op1, op2, result);
endmodule

module level3(op1,op2,result);
defparam test3.test2.operand1_width =10; // Not supported
defparam test3.test2.operand2_width =4; // Not supported
input [9:0] op1;
input [3:0] op2;
output [12:0]result;
level2 test3 (op1, op2, result);
endmodule

```

## Action

Make sure that defparams are used only on the current level of instances.

To change the parameter value across more than one level of hierarchy, use in-line parameter assignment as shown below.

```

module multiplier (operand1, operand2, result);
parameter operand1_width = 4, operand2_width =2;
input [operand1_width -1:0]operand1;
input [operand2_width -1:0] operand2;
output [operand1_width +operand2_width -2:0] result;
 assign result = operand1 * operand2;
endmodule

module level2(op1,op2,result);
parameter operand1_width =16;
parameter operand2_width =8;
input [15:0] op1;
input [7:0] op2;
output [22:0]result;
 multiplier
#(.operand1_width(operand1_width),.operand2_width(operand2_width))
 test2 (op1, op2, result);
endmodule

```

```
module level3(op1,op2,result);
parameter operand1_width =10;
parameter operand2_width =4;
input [9:0] op1;
input [3:0] op2;
output [12:0]result;
level2
#(.operand1_width(operand1_width), .operand2_width(operand2_width))
test3 (op1, op2, result);
endmodule
```

## CG1241

### **@E: Unable to resolve reference to variable <variableName>.**

The synthesis tool does not support a hierarchical reference to a variable within a parameter expression, as shown in the following example:

```
module tst (input din, output dout);
parameter PR1 = GEN.PR2 + 1; // Not supported

generate begin: GEN
 parameter PR2=1;
 assign dout = din;
end endgenerate
endmodule
```

### Action

Do not include hierarchical references within a parameter expression.

## CG1242

### **@E: Unsupported reference encountered during parameter evaluation.**

This error message reports the same issue as message [CG1241](#). The synthesis tool does not support a hierarchical reference to a variable within a parameter expression. In this case, the hierarchical reference to the variable is not available because an internal error occurred.

#### Action

Do not include hierarchical references within a parameter expression.

## CG1245

### **@E: Unexpected data input to array object.**

An internal error occurred during compilation.

#### Action

Contact Synopsys support by logging in to the Synopsys website.

## CG1247

### **@E: Unable to size variable <variableName>: variable may not be defined or its type may not be supported.**

The Verilog compiler encountered a variable that is not defined or is assigned a type that is not supported. For example, you cannot specify port names for a module using special characters such as a period (.).

```
module top (input .a(a) , output b1);
 assign b1 = \.a(a) ;
endmodule
```

## Action

Add the escape identifier (\) to the variable when using special characters and rerun the design.

```
module top (input \.a(a) , output b1);
 assign b1 = \.a(a) ;
endmodule
```

Note that the space following the escaped variable is required to terminate the variable string.

# CG1248

## **@W: default disable is not supported for synthesis**

The SystemVerilog assertion default disable was used, but synthesis does not support this construct as shown below:

```
assign q1 = temp;

default disable iff rst; // Not supported
property p1;
 @(posedge clk) rst |-> ($countones(en) == 1);
endproperty
```

## Action

Note that the default disable construct is not supported and is ignored by the compiler when used.

## CG1250

### @W: Ignoring property block

The SystemVerilog assertion property was used, but synthesis does not support this construct as shown below:

```
assign q1 = temp;

property p1; // Not supported
 @(posedge clk) rst |-> ($countones(en) == 1);
endproperty
```

#### Action

Note that the property construct is not supported and is ignored by the compiler when used.

## CG1251

### @E: Encountered end unit before an endproperty block

The SystemVerilog assertion for property was defined but does not include the endproperty.

#### Action

Make sure that property is defined with an endproperty statement.

```
assign q1 = temp;

property p1;
 @(posedge clk) rst |-> ($countones(en) == 1);
endproperty // endproperty statement required
```

## CG1252

### @W: Ignoring clocking block

The SystemVerilog assertion clocking was used, but synthesis does not support this construct as shown below:

```
assign q1 = temp;

clocking clk1 @(posedge clk); // Not supported
 default input #10ns output #2ns;
endclocking
```

#### Action

Note that the clocking block is not supported and is ignored by the compiler when used.

## CG1253

### @E: Encountered end unit before an endclocking block

The SystemVerilog assertion for clocking was defined but does not include an endclocking statement.

#### Action

Make sure that clocking is defined with an endclocking statement.

```
assign q1 = temp;

clocking clk1 @(posedge clk);
 default input #10ns output #2ns;
endclocking // endclocking statement required
```

## CG1254

### **@E: Label <labelName> must match <labelName>**

The SystemVerilog assertion for clocking was defined, but a label name was not specified after the endclocking statement.

#### Action

Be sure that a label name has been specified at the end of the clocking block to make it complete.

```
assign q1 = temp;

clocking clk1 @(posedge clk);
 default input #10ns output #2ns;
endclocking:clk1 // Label name required
```

## CG1255

### **@W: checker is not supported for synthesis**

The SystemVerilog assertion checker was used, but synthesis does not support this construct as shown below:

```
checker c1; // Not supported
default disable iff rst;
property p1;
 @ (posedge clk) rst |> ($countones(en) == 1);
endproperty
endchecker

module top(input clk,rst,en,input byte d1,output byte q1);
byte temp;
always_ff @(posedge clk) begin
 if(rst)
 temp<=0;
 else if(en)
 temp<=d1;
 else
```

```

 temp <=temp;
end

assign q1 = temp;
endmodule

```

## Action

Note that the checker construct is not supported and is ignored by the compiler when used.

# CG1256

### **@W: final block is not supported for synthesis**

The SystemVerilog assertion final was used, but synthesis does not support this construct as shown below:

```

module top;
 bit clk;
 reg a, b;
 bit SVSflag = 0, SVASflag = 0, SVFFflag = 0, SVAFFflag = 0,
 failedOnce = 0;
 int SVAfailures = 0, SVfailures = 0, SVAsuccesses = 0,
 SVsuccesses = 0;

 always @ (negedge clk) begin
 if ((SVASflag != SVSflag) || (SVAFFflag != SVFFflag)) begin
 $display("[%0d] NON-COMPARE, SVAfailures %0d, SVfailures
 %0d\n",
 $time, SVAfailures, SVfailures,
 "SVAsuccesses %0d, SVsuccesses %0d",
 SVAsuccesses, SVsuccesses);
 failedOnce = 1'b1;
 end
 SVASflag = 0;
 SVSflag = 0;
 SVAFFflag = 0;
 SVFFflag = 10;
end

```

```

final if (failedOnce || // Not supported
 (SVAfailures != SVfailures) ||
 (SVAsuccesses != SVsuccesses))
$display(";-) TEST FAILED, SVAfailures %0d, SVfailures
%0d\n",
SVAfailures, SVfailures,
"SVAsuccesses %0d, SVsuccesses %0d",
SVAsuccesses, SVsuccesses);
else
$display(":-) TEST SUCCEEDED, SVAfailures %0d, SVfailures
%0d\n",
SVAfailures, SVfailures,
"SVAsuccesses %0d, SVsuccesses %0d",
SVAsuccesses, SVsuccesses);

endmodule

```

## Action

Note that the **final** construct is not supported and is ignored by the compiler when used.

# CG1258

## **@W: program is not supported for synthesis**

The SystemVerilog assertion program was used, but synthesis does not support this construct as shown below:

```

module m;
 bit clk;
 reg r, s;
 p p(clk, r);

 global clocking cb1 @(posedge clk);
 output r;
 endclocking
endmodule

program p(input bit clk, output reg r); // Not supported

```

```
clocking cb2 @ (negedge clk);
 output r;
endclocking
default clocking $global_clock;
endprogram
```

## Action

Note that the program construct is not supported and is ignored by the compiler when used.

# CG1260

### **@W: let is not supported for synthesis**

The SystemVerilog assertion let was used, but synthesis does not support this construct as shown below:

```
module top;
 logic a, b;
 let x = a || b; // Not supported
 sequence s;
 x ##1 b;
 endsequence : s
endmodule
```

## Action

Note that the let construct is not supported and is ignored by the compiler when used.

# CG1261

## @E: Cannot assign a packed type to an unpacked type

SystemVerilog. This syntax error occurs when an unpacked declared signal is assigned with a packed dimension. For example, the compiler generates this message for the unpacked register mem that is assigned with a constant zero shown below.

```
module test (
 input [2:0] in1,
 output [2:0] out1,
 input clk, rst
);
reg [2:0] mem [15:0];

assign out1 = mem[15];

always @(posedge clk or posedge rst)
begin
 if (rst)
 mem <= 0;
 else
 mem <= in1;
end

endmodule
```

### Action

Assign the unpacked array using the SystemVerilog aggregate operator instead.

```
module test (
 input [2:0] in1,
 output [2:0] out1,
 input clk, rst
);
reg [2:0] mem [15:0];

assign out1 = mem[15];

always @(posedge clk or posedge rst)
begin
```

```

 if (rst)
 mem <= '{default:0};
 else
 mem <= in1;
 end

endmodule

```

## CG1262

### **@E: Variable <name> has a type that cannot be the target of this assignment**

The compiler cannot apply the continuous assignment you specified for the variable. In this test, the variable P1 is assigned to instance i0 with a dynamic value of a[1:0].

```

module sub #(parameter logic[3:0] P1 = 4)(input logic[3:0]
 a, output logic[3:0] c);
 assign c = a;
endmodule

module top #(parameter[3:0] P1 = 1, parameter[3:0]
 P2 = 2)(input logic[3:0] a,b, output logic[3:0] c);
 sub i0 (a,c);
 assign i0.P1[1:0] = a[1:0];
 endmodule

```

### Action

Avoid using continuous assignments for parameters. Make sure that the parameters are not assigned with dynamic values and use parameter mapping in the instantiation instead.

```

module sub #(parameter logic[3:0] P1 = 4)(input logic[3:0]
 a, output logic[3:0] c);
 assign c = a ;
endmodule

```

```
module top #(parameter[3:0] P1 = 1, parameter[3:0] P2 = 2)
 (input logic[3:0] a,b, output logic[3:0] c);
 sub #(.P1(4'b0100)) i0 (a,c);
 //Use parameter mapping to pass value to the parameter
endmodule
```

## CG1266

### **@E: Too many port connections <integer>: exceeds the number of ports <integer>**

The number of ports specified for the instantiated module exceeds the actual number of ports for the module. In the following example, the module sub only has two ports, but the module instantiation was specified with three ports.

```
module sub (input a, output b
);
 assign b =a;
endmodule

module top (input a, output [15:0] b, output [15:0] c
);
sub [15:0] (a,b,c);
endmodule
```

### Action

Make sure that the number of ports specified for an instantiation of a module matches the actual number of ports for the module as shown below.

```
module sub (input a, output b
);
 assign b =a;
endmodule
```

```
module top (input a, output [15:0] b, output [15:0] c
);
 sub [15:0] (a,b);

endmodule
```

## CG1280

### **@E: The instance name associated with the specified instance clause in the config could not be resolved**

This error occurs when the contents of a configuration contain a syntax error. The design will compile, but this error will be reported.

The following examples contain lines which would result in this error.

Example one:

```
config cfg;
 design tst;
 instance tst.S[1] use work.leaf;
 instance tst.S[1 use work.leaf;
endconfig
```

The error is issued because the ] in the tst.S[1 string is missing.

Example two:

```
config cfg;
 design tst;
 instance tst.S[1] use work.leaf;
 instance tst.S[w] use work.leaf;
endconfig
```

The error is issued because w in tst.S[w] is not a compile-time constant.

### Action

Correct all syntax errors.

## CG1282

### **@E: Exponents with variables must be a positive constant integer**

This error occurs when an exponent expression of type (a \*\* b) is not a positive constant integer.

#### Action

Use only positive constant integers for exponents with variables.

## CG1287

### **@E: A configuration specified as the top level is not supported. Please specify a module or interface as the top level**

This error occurs when a top level is specified as configuration, which is not supported.

#### Action

Specify only a module or interface as a top level.

## CG1289

### **@E: Arguments are supported only for enumerated type methods prev() and next().**

Only enumerated type methods prev and next support arguments. If arguments are specified for other enumerated type methods, this error occurs.

### Action

Specify arguments only for enumerated type methods prev and next.

## CG1290

### **@E: <identifier> is not a valid enumerated type method.**

This error occurs when an invalid enumeration type method is used.

### Action

Use only one of the following enumerated type methods: first(), last(), prev(), next(), num(), or name().

## CG1291

### **@E: Argument to enumerated type method prev() or next () greater than 32 bits is not supported.**

An argument to enumerated type method prev or next is greater than 32-bits. Arguments must be less than or equal to 32-bits.

### Action

Set all prev and next arguments to 32-bits or smaller.

## CG1292

### **@E: Unknown type <type> in function argument declaration**

This error occurs when a function argument is not defined or is defined as an unknown type.

## Action

Define function argument as a known type. For example:

```
typedef logic myType;

module tst (input a, output b);
 function logic fun1(input myType f1);
 fun1 = f1;
 endfunction

 assign b = fun1(a);
endmodule
```

# CG1293

## @E: Enumerated type method name( ) is not supported.

The enumerated type was specified using name( ), but the compiler does not support this method. For example:

```
typedef enum logic[2:0] {red=3'b101, green, blue} colors;
module top(output colors q_name);

colors c;
assign q_name = c.name();
```

## Action

Avoid using the enumerated type method name( ).

## CG1295

### **@E: Multiple configurations are not supported with distributed compilation**

This error occurs when a design has multiple configurations and distributed compilation is enabled.

#### Action

Disable distributed compilation and recompile the design.

## CG1296

### **@E: XMRs on bidirectional ports are not supported in the unified compile flow.**

Assigning an XMR (Cross Module Reference) to a bidirectional port is not supported.

#### Action

Do not assign an XMR to a bidirectional (*inout*) port. Assign the XMR to the *input* or *output* port.

## CG1299

### **@E: The number of iterator variables (<iterators>) exceeds the number of dimensions of the variable <dimensions>**

This error occurs when the number of variables (including unspecified variables) in a list exceeds the number of dimensions in the variable or array.

**Action**

Decrease the number of iterator variables or increase the number of dimensions of the variable or array.

## CG1300

**@E: Illegal or unsupported foreach statement**

Foreach statement contains a System Verilog syntax error.

**Action**

Correct the statement or contact support.

## CG1301

**@E: foreach statements are not supported yet**

Foreach statements are not currently supported in this configuration.

**Action**

Avoid using unsupported statements.

## CG1303

**@E: Expecting an identifier or a comma separator**

An identifier or comma separator is missing between variables in a loop list.

## Action

Separate loop list variables with a comma or other identifier. For example:

[variable1],[variable2],[variable3]

# CG1310

### **@E: Argument count mismatch for function <*countBitsFunction*>, expecting at least two arguments.**

The specified system function requires at least two arguments be provided during the \$countbits function call. This error occurs when less than two arguments are provided.

## Action

Make sure to specify at least two arguments for the \$countbits function call.

# CG1311

### **@E: Control bits of <*countBitsFunction*> must be a constant.**

If the control bit arguments of the \$countbits function do not evaluate to a constant value when compiled, then this error is generated for the specified system function.

## Action

Make sure that the control bits for the \$countbits function evaluate to a constant during compile.

## CG1315

### **@E: function call <*function*> incomplete; missing parentheses**

When a parenthesis in a function is missing, this error is produced.

#### Action

Use both opening and closing parentheses.

## CG1316

### **@E: Unable to attach defparam <*parameter*> to instance <*instance*>.**

This error occurs when a defparam cannot be attached to a hierarchy. For example, if a defparam is used across an array of instances.

#### Action

Do not use defparams to override parameters of modules, instead use inline parameter passing.

## CG1318

### **@E: Instance clause not applied in distributed mode(Instance hier Path:<*path*> cell:<*cell*> lib:<*library*>)."**

The instanced clause has not been correctly applied to the specified path, cell, or library.

#### Action

Contact Synopsys support by logging in to the Synopsys website.

As a temporary workaround, disable Distributed Compiler and recompile.

## CG1319

### **@E: Instance directive not applied in distributed mode(Instance hier Path:<path>)."**

The instance clause in the configuration or in the cross-module referencing could not be resolved.

#### Action

Contact Synopsys support by logging in to the Synopsys website.

As a temporary workaround, disable Distributed Compiler and recompile.

## CG1320

### **@E: View directive not applied in distributed mode(view:<viewName>)."**

The instance clause in the configuration or in the cross-module referencing could not be resolved.

#### Action

Contact Synopsys support by logging in to the Synopsys website.

As a temporary workaround, disable Distributed Compiler and recompile.

## CG1324

### **@E: The begin\_keywords directive can only be used outside of a design unit**

A 'begin\_keywords directive has been encountered within a module, interface, or package. This directive specifies the list of identifiers reserved as keywords and can only be used outside of the design unit.

#### Action

Check the file and line location preceding the message text and relocate the directive outside of the design unit.

## CG1326

### **@E: The end\_keywords directive can only be used outside of a design unit**

An 'end\_keywords directive has been encountered within a module, interface, or package. This directive specifies the end of the list of identifiers reserved as keywords and can only appear outside of the design unit.

#### Action

Check the file and line location preceding the message text and relocate the directive outside of the design unit.

## CG1327

### **@W: Encountered the end\_keywords directive before a begin\_keywords directive**

An end\_keywords directive has been encountered before its corresponding begin\_keywords directive.

#### Action

Check the file and line location preceding the message text and make sure that the end\_keywords directive always follows its initial begin\_keywords directive.

## CHAPTER 16

# CH Messages 100 – 103

---

## CH100

### **@E: Encountered multiple top-level candidates in design; compilation stopped.**

The compiler encountered multiple top-level modules or entities, so could not continue running. In this test, multiple modules/entities were found for the following mixed HDL project.

This is the VHDL file.

```
library ieee;
use ieee.std_logic_1164.all;

entity and_oper is
 port (in1 : in std_logic_vector (31 downto 0);
 in2 : in std_logic_vector (31 downto 0);
 out1 : out std_logic_vector (31 downto 0));
end and_oper;

architecture behav of and_oper is
begin
 out1 <= in1 and in2;
end behav;
```

Here is the Verilog file.

```
module or_oper (
 input int in1,in2,
 output int out1
);
assign out1 = in1 | in2;
endmodule
```

## Action

Make sure the top-level module for the project is specified. To do this, you can enter the top-level module from either the Verilog or VHDL tab of the Implementation Options panel. The Tcl equivalent command is

```
set_option -top_module "/lib.topModuleName"
```

# CH101

## **@E: Structural Verilog compiler does not support selected technology**

The specified technology does not support the structural Verilog flow. This error message results when an attempt is made to use the structural Verilog flow for an unsupported technology.

## Action

The structural Verilog flow is only supported for specific technologies. To use the selected technology, select the default Verilog compiler by changing the `add_file` setting in the project file from `-structver` to `-verilog`.

## CH102

### **@E: Structural Verilog compiler failed. Invoke Verilog compiler when design contains illegal structural Verilog constructs.**

An unsupported construct has been encountered in the structural Verilog flow (only a limited number of Verilog-95 constructs are supported in structural Verilog). For example, the named association for port mapping shown in the code segment below, which is a legal SystemVerilog construct, is not supported in the structural Verilog flow.

```
//named dot port mapping
FDCE U1_Z (
 .Q,
 .D,
 .C,
 .CLR,
 .CE);
```

#### Action

Use the default Verilog compiler flow by changing the `add_file` setting in the project file from `-structver` to `-verilog` for the impacted module (or use a construct compatible with structured Verilog).

## CH103

### **@E: One or more distribution nodes failed to compile.**

While using Distributed Compilation, one or more of the nodes failed to compile. See the log file for the reason for the failure.

#### Action

If the reason for the failure is not related to the design, turn Distributed Compilation off and compile the design again. Otherwise, correct the design error and recompile.

# CH105

## **@E: Error while linking in distributed mode.**

While using the Distributed Compile option, the linking failed while combining the individual netlists.

### Action

A preceding linker error message should contain the reason for the failure. Contact Synopsys support to report this case. As a workaround, you can disable the distributed compilation option (option set distributed\_compile 0) and rerun this design using the non-distributed flow.

## CHAPTER 17

# CL Messages 104 – 168

---

## CL104

### @E: Couldn't find binding for variable <index>

Using variable partial select and the *width expression* does not evaluate to a constant. In the test case below, the *width expression* is specified as a variable (*index*) and not as a constant expression which causes the compiler to error out when attempting to access the input vector.

```
module buffer (input [7:0] in, output [7:0]out);
 wire [5:0] index;
 assign index=0 ;
 assign out = in[index+:index];
endmodule
```

This SystemVerilog assertion error also occurs when the LET construct is forward referenced, but should be ignored.

```
module m;
 bit b1,b3;
 let z =b1;
 assign b3 = acomb.b;
always_comb
begin:acomb
bit b;
```

```

 let y = z;
end

endmodule

```

## Action

To eliminate when using variable partial select expressions, make sure that the *width\_expression* evaluates to a constant at compile time. In the corrected test case below, the variable index is replaced by a constant.

```

module buffer (input [7:0] in, output [7:0]out);
wire [5:0] index;
assign index=0 ;
assign out = in[index+:8];
endmodule

```

For the SystemVerilog assertion error, please contact Synopsys support.

# CL105

## **@E: Found mixed blocking and non-blocking assignments to <mem>**

The compiler encountered an array object with mixed blocking and non-blocking assignments. In the test case below, mem is an array object with each element two bits wide. The compiler encounters mixed blocking and non-blocking assignments to mem which causes the error.

```

module ram0(input [1:0] data, input [1:0] waddr, addr,
 input sel, we,clk, output [1:0] q);
reg [1:0]mem [3:0];
always @(posedge clk)
begin
 if(sel)
 if(we) mem[waddr] <= data;
 else mem[waddr] =!data;
end
assign q = mem[addr];
endmodule

```

## Action

Do not mix blocking and non-blocking assignments of an array object. In the corrected test case below, array mem has only non-blocking statements.

```
module ram0(input [1:0] data, input [1:0] waddr, addr,
 input sel, we,clk, output [1:0] q);
reg [1:0]mem [3:0];
always @(posedge clk)
begin
 if(sel)
 if(we) mem[waddr] <= data;
 else mem[waddr] <=!data;
 end
assign q = mem[addr];
endmodule
```

## CL107

### **@A: Too many clocks (> 8) for set/reset analysis of <q>, try building the latch inside process**

The number of gating or clock signals exceeds eight and a latch is being generated by concurrent signal assignment statements. In the following test case, the input is latched based on a combination of more than eight signals.

```
library ieee;
use ieee.std_logic_1164.all;

entity top is
 port (i, j, k, l, m, n, o, p,r: in std_logic;
 a: in std_logic_vector(3 downto 0);
 q: inout std_logic_vector(3 downto 0));
end;

architecture rtl of top is
begin
 q<= a when ((i or j or k or l or m)
 or (n or o or p or r))='1'
 else q;
end;
```

## Action

By making the latch output assignment within a process block, a sequential element such as a latch can be described accurately with all of its control signals as shown in the modified test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity top is
 port (i, j, k, l, m, n, o, p, r: in std_logic;
 a: in std_logic_vector(3 downto 0);
 q: inout std_logic_vector(3 downto 0));
end;

architecture rtl of top is
signal en: std_logic;
begin
en<=(i or j or k or l or m or n or o or p or r);
process(a,en)
begin
 if en='1' then
 q<=a;
 end if;
end process;
end;
```

## CL108

**@W: Too many clocks (> 8) for set/reset analysis of <q>, try building the latch inside an always block**

The number of gating signals or clock signals exceeds 8. In the following test case, the input is latched based on a combination of more than eight signals.

```
module error (
 input i, j, k, l, m, n, o, p, r,
 input [3:0] a,
 output [2:0] q);
 assign q = (i | j | k | l | m | n | o | p | r) ? a : q;
endmodule
```

## Action

To eliminate the warning in the above test case, use an always block to make the output assignments as shown in the modified test case below.

```
module error (
 input i, j, k, l, m, n, o, p, r
 input [3:0] a
 output [2:0] q);
 wire en;
 reg[2:0]q;
 assign en = (i | j | k | l | m | n | o | p | r);

 always @ (a or en)
 begin
 if (en)
 q <= a;
 end
 endmodule
```

By making the latch output assignment in an always block, a sequential element such as a latch can be described accurately with all of its control signals.

## CL109

### **@A: Too many clocks (> 8) for set/reset analysis of temp, try moving enabling expressions outside process**

The number of gating or clock signals that controls latches/registers exceeds eight and a latch is being generated by sequential assignment statements. Large numbers of control signals such as gating signals for latches can degrade performance when building the complex control logic for synchronizing data with the control signals. This advisory alerts you during the compilation phase of potential timing issues that may arise later during synthesis. In the following test case, the latch is driven by too many clocks within the process block.

```
library ieee;
use ieee.std_logic_1164.all;
```

```

entity top is
 port (i, j, k, l, m, n, o, p, r: in std_logic;
 a: in std_logic_vector(3 downto 0);
 q: inout std_logic_vector(3 downto 0));
end;

architecture rtl of top is
begin
 process(i,j,k,l,m,n,o,p,r,a)
 begin
 if (((i or j or k or l or m) or (n or o or p or r))='1') then
 q<=a;
 end if;
 end process;
end;

```

## Action

Use an enable signal outside the process and make the output assignments as shown in the modified test case below.

```

library ieee;
use ieee.std_logic_1164.all;

entity top is
 port (i, j, k, l, m, n, o, p, r: in std_logic;
 a: in std_logic_vector(3 downto 0);
 q: inout std_logic_vector(3 downto 0));
end;

architecture rtl of top is
signal en: std_logic;
begin
en<=(i or j or k or l or m or n or o or p or r);
process(a,en)
begin
 if en='1' then
 q<=a;
 end if;
end process;
end;

```

## CL110

### @A: Too many clocks (> 8) for set/reset analysis of <temp>, try moving enabling expressions outside always block

Too many clocks are driving a latch. Large numbers of control signals such as gating signals for latches can degrade performance when building the complex control logic for synchronizing data with the control signals. This advisory alerts you during the compilation phase of potential timing issues that may arise later during synthesis. In the following test case, the latch is driven by too many clocks within the always block.

```
module gr8clk (
 input i, j, k, l, m, n, o, p, r,
 input [3:0] a,
 output [2:0] q);
 reg[2:0]q;

 always @ (*)
 begin
 if (i | j | k | l | m | n | o | p | r)
 q <= a;
 end
 endmodule
```

### Action

Try moving the enabling expressions outside of the always block as shown in the modified test case below or use as small a number of controlling signals as possible.

```
module gr8clk (
 input i, j, k, l, m, n, o, p, r,
 input [3:0] a,
 output [2:0] q);
 wire en;
 reg[2:0]q;
```

```

assign en = (i | j | k | l | m | n | o | p | r);
always @ (a or en)
begin
 if (en)
 q <= a;
 end
endmodule

```

## CL111

**@W: All reachable assignments to <out1(0)> are <1>; removing register. To preserve a constant register, use the syn\_preserve attribute.**

An output or a signal is assigned the same constant value in every reachable branch of the design. As a result, the register is removed and the signal/output is tied to the constant value. In the following test case, out1 is a primary output. The design has two assigning branches to out1. Because both branches assign the same constant value to 1, the out1 register is optimized away.

This is a valid optimization.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity test is
 port (a, b: in std_logic_vector (7 downto 0);
 clk, set: in std_logic;
 out1, out2: out std_logic_vector (7 downto 0));
end entity;

architecture only of test is
begin
 process (clk, set)
 begin
 if (set = '1') then
 out1 <= (others => '1');

```

```
 elsif clk'event and clk = '1' then
 out1 <= (others => '1');
 end if;
 end process;
end only;
```

### Action

Make sure this is the intended behavior. To preserve the registers tied to constants, use `syn_preserve` on the architecture of the entity.

Sequential optimizations are enabled by default.

## CL113

**@W: Feedback mux created for signal <`d_out[3:0]`>. To avoid the feedback mux, assign values explicitly under all conditions of conditional assignment statements.**

A conditional assignment block of a clocked process (or multiple event triggered always block) does not have an assignment for one of the conditions. As a result, a feedback mux is created to retain the register value for that condition.

### Verilog Test Case

In the following test case, under the condition of `set`, register `d_out` is not assigned a value which results in the warning.

```
module c1113_v (clk,rst,set,a,d_out);
 input clk,rst,set,a;
 output reg d_out;
 always @ (posedge clk,posedge rst,posedge set)
 begin
 if (rst)
 d_out<=1'b0;
 else if (set)
 begin
```

```

end
 else if(clk)
 d_out<=a;
 end
endmodule

```

## Action

To avoid any unintentional feedback and the above warning, assign values explicitly under all conditions of conditional assignment statements. To eliminate the warning in the above test case, add `d_out <=1'b1` after the `set` condition check statement in the always block as shown below.

```

module c1113_v (clk,rst,set,a,d_out);
 input clk,rst,set,a;
 output reg d_out;
 always@(posedge clk,posedge rst,posedge set)
 begin
 if (rst)
 d_out<=1'b0;
 else if(set)
 begin
 d_out <=1'b1;
 end
 else if(clk)
 d_out<=a;
 end
 endmodule

```

## VHDL Test Case

In the following test case, register `d_out` is not assigned a value under condition `set ='1'` which results in the warning.

```

library ieee;
use ieee.std_logic_1164.all;

entity warn is
 port (clk: in std_logic;
 rst,set: in std_logic;
 a: in std_logic;
 d_out: out std_logic);
end warn;

```

```
architecture beh of warn is
begin
 process(clk,rst,a,set)
 begin
 if rst='0' then
 d_out<='0';
 elsif set='1' then
 elsif rising_edge(clk) then
 d_out <= a;
 end if;
 end process;
 end beh;
```

## Action

To avoid any unintentional feedback and the above warning, assign values explicitly under all conditions of conditional assignment statements. To eliminate the warning in the above test case, add `d_out <= '1'` after the `set = '1'` condition check statement in the process block as shown below.

```
library ieee;
use ieee.std_logic_1164.all;

entity warn is
 port (clk: in std_logic;
 rst,set: in std_logic;
 a: in std_logic;
 d_out: out std_logic);
end warn;

architecture beh of warn is
begin
 process(clk,rst,a,set)
 begin
 if rst='0' then
 d_out<='0';
 elsif set='1' then
 d_out<='1';
 elsif rising_edge(clk) then
 d_out <= a;
 end if;
 end process;
 end beh;
```

## CL117

### @W: Latch generated from process for signal <outp(7 downto 0)>; possible missing assignment in an if or case statement

All conditions are not declared in a case statement when an if statement exists that describes purely combinatorial logic. Under these conditions, the compiler infers a latch. In the following test case, a latch is generated for signal outp because the if condition inp = "111" is missing.

```
library ieee;
use ieee.std_logic_1164.all;

entity decoder is
 port (inp: in std_logic_vector(2 downto 0);
 outp: out std_logic_vector(7 downto 0));
end decoder;

architecture behave of decoder is
begin
 process (inp) begin
 if (inp = "000")then
 outp <= "00000001";
 else
 if (inp = "001") then
 outp <= "00000010";
 else
 if (inp = "010") then
 outp <= "00000100";
 else
 if (inp = "011") then
 outp <= "00001000";
 else
 if (inp = "100") then outp <= "00010000";
 else if (inp = "101") then outp <= "00100000";
 else if (inp = "110") then outp <= "01000000";
 end if;
 end if;
 end if;
 end if;
 end if;
 end process;
end;
```

```

 end if;
 end if;
-- end if;
 end process;
end behave;
```

## Action

Three ways to avoid this warning are:

- Complete the if clause by using the else clause in an if-then-else statement.
- Always use a default clause in a case statement.
- Use the Verilog full\_case directive, /\* synthesis full\_case \*/, if the case statement includes all possible case choices.

To eliminate this warning in the above test case, add an else clause describing the if condition (inp = "111").

```

library ieee;
use ieee.std_logic_1164.all;

entity decoder is
 port (inp: in std_logic_vector(2 downto 0);
 outp: out std_logic_vector(7 downto 0));
end decoder;

architecture behave of decoder is
begin
 process (inp) begin
 if (inp = "000")then
 outp <= "00000001";
 else
 if (inp = "001") then
 outp <= "00000010";
 else
 if (inp = "010") then
 outp <= "00000100";
 else
 if (inp = "011") then
 outp <= "00001000";
 else
 if (inp = "100") then outp <= "00010000";
 else if (inp = "101") then outp <= "00100000";
 else if (inp = "110") then outp <= "01000000";
```

```

 else if (inp = "111") then outp <= "10000000";
 else outp <= "XXXXXXXX";
 end if;
 end if;
-- end if;
end process;
end behave;

```

For case conditions, make sure all case choices are covered. To retain the value of the signal for unspecified cases, use the when others clause to cover the default cases to make sure that no unnecessary latches are inferred.

## CL118

### **@W: Latch generated from always block for signal <out1>; possible missing assignment in an if or case statement**

All conditions are not declared in a case statement or an if statement exists describing purely combinatorial logic. Under these conditions, the compiler infers a latch. In the following test case, a latch is generated for signal out1 because the if condition sel = 2'b00 is missing.

```

module newmux (out1, a, b, c, sel);
 input a, b, c;
 output out1;
 input[1:0] sel;
 reg out1;

 always@(a or b or c or sel)
 begin
 if (sel == 2'b10)
 out1 = a;
 else if (sel == 2'b01)

```

```
 out1 = b;
else if (sel == 2'b11)
 out1 = c;
end
endmodule
```

## Action

Three ways to avoid this warning are:

- Complete the if clause by using the else clause in an if-then-else statement.
- Always use a default clause in a case statement.
- Use the Verilog full\_case directive, /\* synthesis full\_case \*/, if the case statement includes all possible case choices.

To eliminate the warning in the above test case, edit the code as shown in the corrected test case below.

```
module newmux (out1, a, b, c, sel);
input a, b, c;
output out1;
input[1:0] sel;
reg out1;

always@(a or b or c or sel)
begin
 if (sel ==2'b10)
 out1 = a;
 else if (sel == 2'b01)
 out1 = b;
 else if (sel == 2'b11)
 out1 = c;
 else out1 = a;
end
endmodule
```

# CL119

## @E: More than one edge sensitive signal unused in always block

All of the edge-sensitive signals in the sensitivity list were not used inside the corresponding `always` block (an `always` block must cover all cases of edge-sensitive signals in the sensitivity list). In the test case below, the conditions for edge-sensitive signals `set` and `rst` are not specified in the `always` block which causes the error.

```
module seq(d,rst,clk,set,q);
 input d,rst,set,clk;
 output reg q;

 always@(posedge clk or negedge rst or posedge set)
 begin
 if(clk)
 q=d;
 end
 endmodule
```

### Action

Make sure that all the signals defined in the sensitivity list are described in the `always` block. In the corrected test case below, each signal in the sensitivity list appears in the `always` block definition.

```
module seq(d,rst,clk,set,q);
 input d,rst,set,clk;
 output reg q;

 always@(posedge clk or negedge rst or posedge set)
 begin
 if(!rst)
 q=0;
 else if(set)
 q=1;
 else if(clk)
 q=d;
 end
 endmodule
```

# CL121

## @E: Can't find control signal for <madd>

An assignment was made inside a process when an edge-triggered condition was not true and a subsequent condition was true. When coding sequential elements, the last `elsif` statement must specify the clock edge. In the test case below, because the second `elsif` expression is checked when there is no clock event, it is not possible to infer logic from this description.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity test is
 port (clk, rst: in std_logic;
 mode: in std_logic;
 radd: in std_logic_vector(1 downto 0);
 a: out std_logic_vector(1 downto 0);
 madd: buffer std_logic_vector(1 downto 0));
end test;

architecture beh of test is
begin
 process (rst, clk, mode, radd, madd)
 begin
 if (rst = '0') then
 madd <= "00";
 a <= "00";
 elsif (rising_edge(clk)) then
 madd <= madd + 1;
 a <= madd;
 elsif (mode = '1') then
 madd <= radd;
 end if;
 end process;
end beh;
```

## Action

Make sure that the check on the condition (`mode = '1'`) is made synchronous by placing it under the rising edge of the clock. To eliminate the error in the above test case, avoid the `elsif` chain as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity test is
 port (clk, rst: in std_logic;
 mode: in std_logic;
 radd: in std_logic_vector(1 downto 0);
 a: out std_logic_vector(1 downto 0);
 madd: buffer std_logic_vector(1 downto 0)
);
end test;

architecture beh of test is
begin
 process (rst, clk, mode, radd, madd)
 begin
 if (rst = '0') then
 madd <= "00";
 a <= "00";
 elsif (rising_edge(clk)) then
 if (mode = '1') then
 madd <= radd;
 else
 madd <= madd + 1;
 a <= madd;
 end if;
 end if;
 end process;
end beh;
```

## CL123

### **@E: Logic for <out[7:0]> does not match a standard flip-flop**

An attempt was made to create a register that did not represent a standard flip-flop. The compiler only extracts standard flip-flops that are clocked by a single clock edge.

## Verilog Test Case

In the following test case, the process is executed whenever there are multiple edges (on clock or on reset), but the flip-flop is to be loaded with the current data on both edges. This behavior does not match the normal operation of the flip-flop.

```
module counter2 (out, data, rst, clk);
 output [7:0] out;
 input [7:0] data;
 input rst,clk;
 reg [7:0] out;
 reg cout;

 // create the 8-bit register
 always @(posedge clk or negedge rst)
 begin
 out <= data;
 end
endmodule
```

## Action

Recode the process to match a flip-flop in hardware. To eliminate this error in the above test case, add a condition to reset the flip-flop as shown in the corrected test case below.

```
module counter2 (out, data, rst, clk);
 output [7:0] out;
 input [7:0] data;
 input rst,clk;
 reg [7:0] out;
 reg cout;

 // create the 8-bit register
 always @(posedge clk or negedge rst)
 begin
 if (!rst)
 out <= 8'b0;
 else
 out <= data;
 end
endmodule
```

## VHDL Test Case

In the following test case, the process is executed whenever there are multiple edges (on clock or on reset), but on both edges, the flip-flop is to be loaded with the current data. This action does not match the normal operation of a flip-flop which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity error is
 port (clk,a,en,rst: in std_logic;
 dout: out std_logic);
end error;

architecture behave of error is
begin
 process (clk,rst)
 begin
 if (rst = '0') then
 dout <= '0';
 if (clk ='1' and clk'event) then
 if (en = '1' and en'event) then
 dout <= a;
 end if;
 end if;
 end if;
 end process;
end behave;
```

## Action

Recode the process to match a flip-flop in hardware. To eliminate this error in the above test case, add a condition to reset the flip-flop as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity error is
 port (clk,a,en,rst: in std_logic;
 dout: out std_logic);
end error;
```

```

architecture behave of error is
begin
 process (clk,rst)
 begin
 if (rst = '0') then
 dout <= '0';
 elsif (clk ='1' and clk'event) then
 if (en = '1') then
 dout <= a;
 end if;
 end if;
 end process;
end behave;

```

## CL125

### **@E: Logic for <q> does not match a standard flip-flop -- ambiguous clock**

The signal functioning as a clock (based on *signal/event*) was not properly defined. In the test case below, the compiler interprets signal *clk* as the clock source based on *clk'event*, but since the other condition (i.e., *clk='1'* or *clk='0'*) is not specified, the compiler errors out when it is unable to determine if the flip-flop is negative-edge or positive-edge triggered.

```

library ieee;
use ieee.std_logic_1164.all;

entity latchor2 is
 port (a, b, clk : in std_logic ;
 q: out std_logic);
end latchor2;

architecture behave of latchor2 is
begin
 process (clk, a, b)
 begin
 if clk'event then
 q <= a or b;
 end if;
 end process;
end behave;

```

## Action

Make sure all clocks for flip-flops are properly defined. In the test case below, specifying the condition as `clk'event` and `clk='1'` clearly defines signal `clk` as the clock source and the flip-flop as being positive-edge triggered.

```
library ieee;
use ieee.std_logic_1164.all;

entity latchor2 is
 port (a, b, clk : in std_logic ;
 q: out std_logic);
end latchor2;

architecture behave of latchor2 is
begin
 process (clk, a, b)
 begin
 if (clk'event and clk = '1') then
 q <= a or b;
 end if;
 end process;
end behave;
```

## CL126

### **@E: Asynchronous load of non-constant data for <count(0)> is not supported**

The HDL code contained an incomplete description of an asynchronous load on a sequential element. This error may mask other errors such as “the logic does not match a standard flip-flop.” In the following test case, the `clk` transition is incomplete (missing the `clk ='1'` clause) which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity counter is
 port (clk, reset: in bit;
 count: buffer integer range 0 to 255);
end counter;
```

```
architecture behave of counter is
begin
 p1: process (clk)
 begin
 if (reset = '1' or count = 255) then
 count <= 0;
 elsif (clk'event) then
 count <= count + 1;
 end if;
 end process p1;
end behave;
```

## Action

Be sure to include the full transition of the clock as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity counter is
 port (clk, reset: in bit;
 count: buffer integer range 0 to 255);
end counter;

architecture behave of counter is
begin
 p1: process (clk)
 begin
 if (reset = '1' or count = 255) then
 count <= 0;
 elsif (clk'event and clk ='1')
 count <= count + 1;
 end if;
 end process p1;
end behave;
```

# CL132

## @W: Read Address width <2> does not cover RAM depth

The compiler inferred a RAM and the read address width was insufficient to read the entire RAM as declared in the source code.

### Verilog Test Case

In the test case below, the RAM declared is 16 x 8 (inferred from the statement `reg [3:0] mem [7:0]`). Accordingly, the address width must be four bits width to access this memory for reading.

```
module ramtest (z, raddr, d, waddr, we, clk);
 output [7:0] z;
 input [7:0] d;
 input [1:0] raddr;
 input [3:0] waddr;
 input we;
 input clk;
 reg [3:0] mem [7:0];
 assign z = mem[raddr];

 always @(posedge clk) begin
 if (we)
 mem[waddr] = d;
 end
 endmodule
```

### VHDL Test Case

In the test case below, a 2-bit address is specified for a 16 x 8 RAM that requires a 4-bit address to read or write the memory which results in the warning.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
```

```

entity ramtest is
 port (q: out std_logic_vector (7 downto 0);
 d: in std_logic_vector (7 downto 0);
 addr: in std_logic_vector (1 downto 0);
 we: in std_logic;
 clk: in std_logic);
end ramtest;

architecture rtl of ramtest is
type mem_type is array (15 downto 0) of
 std_logic_vector (7 downto 0);
signal rd_addr: std_logic_vector (1 downto 0);
signal mem: mem_type;
begin
 q <= mem(conv_integer(rd_addr));
 process (clk, we, addr)
 begin
 if rising_edge (clk) then
 if (we = '1') then
 mem (conv_integer (addr)) <= d;
 end if;
 rd_addr <= addr;
 end if;
 end process;
end rtl;

```

## Action

Make sure that the address and data widths match the specified RAM.

## Corrected Verilog Test Case

Since the RAM is declared as 16 x 8, make sure that the address width is four bits wide. To eliminate the warning in the Verilog test case, increase the read address width as shown in the corrected test case below.

```

module ramtest (z, raddr, d, waddr, we, clk);
output [7:0] z;
input [7:0] d;
input [3:0] raddr;
input [3:0] waddr;
input we;
input clk;
reg [3:0] mem [7:0];
assign z = mem[raddr];

```

```

always @(posedge clk) begin
 if (we)
 mem[waddr] = d;
 end
endmodule

```

## Corrected VHDL Test Case

Since the RAM is declared as 16 x 8, make sure that the address width is four bits wide. To eliminate the warning in the VHDL test case, change the width of the read address as shown in the corrected test case below.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity ramtest is
 port (q: out std_logic_vector (7 downto 0);
 d: in std_logic_vector (7 downto 0);
 addr: in std_logic_vector (3 downto 0);
 we: in std_logic;
 clk: in std_logic);
end ramtest;

architecture rtl of ramtest is
type mem_type is array (15 downto 0) of
 std_logic_vector (7 downto 0);
signal rd_addr: std_logic_vector (3 downto 0);
signal mem: mem_type;
begin
q <= mem(conv_integer(rd_addr));
process (clk, we, addr)
begin
 if rising_edge (clk) then
 if (we = '1') then
 mem (conv_integer (addr)) <= d;
 end if;
 rd_addr <= addr;
 end if;
end process;
end rtl;

```

This warning is generated when a RAM exists in the RTL code. The warning is accompanied by the following note:

@N: Found RAM mem, depth=8, width=4

This indicates the size of the RAM inferred in the code.

## CL133

### **@W: Write Address width <2> does not cover RAM depth**

The compiler inferred a RAM and the write address width was insufficient to write into the entire RAM as declared in the code.

#### Verilog Test Case

In the test case below, the RAM declared is 16x8 (inferred from the statement `reg [3:0] mem [7:0]`). Accordingly, the address width must be four bits wide to access this memory for writing.

```
module ramtest (z, raddr, d, waddr, we, clk);
 output [7:0] z;
 input [7:0] d;
 input [3:0] raddr;
 input [1:0] waddr;
 input we;
 input clk;
 reg [3:0] mem [7:0];
 assign z = mem[raddr];

 always @ (posedge clk) begin
 if (we)
 mem[waddr] = d;
 end
endmodule
```

#### VHDL Test Case

In the following test case, a 2-bit address is specified for a 16 x 8 RAM that requires a 4-bit address to read or write the memory which results in the warning.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
```

```

entity ramtest is
 port (q: out std_logic_vector (7 downto 0);
 d: in std_logic_vector (7 downto 0);
 addr: in std_logic_vector (1 downto 0);
 we: in std_logic;
 clk: in std_logic);
end ramtest ;

architecture rtl of ramtest is
type mem_type is array (15 downto 0) of
 std_logic_vector (7 downto 0);
signal wr_addr : std_logic_vector (1 downto 0);
signal mem : mem_type;
begin
q <= mem(conv_integer(wr_addr));
process (clk, we, addr)
begin
 if rising_edge (clk) then
 if we = '1' then
 mem (conv_integer (addr)) <= d;
 end if;
 wr_addr <= addr;
 end if;
end process;
end rtl;

```

## Action

Make sure that the address and data widths match the specified RAM. Since the RAM is declared as 16 x 8, make sure that the address width is four bits wide.

## Corrected VHDL Test Case

To eliminate the warning in the VHDL test case, change the width of the write address as shown in the corrected test case below.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

```

```
entity ramtest is
 port (q: out std_logic_vector (7 downto 0);
 d: in std_logic_vector (7 downto 0);
 addr: in std_logic_vector (3 downto 0);
 we: in std_logic;
 clk: in std_logic);
end ramtest ;

architecture rtl of ramtest is
type mem_type is array (15 downto 0) of
 std_logic_vector (7 downto 0);
signal wr_addr : std_logic_vector (3 downto 0);
signal mem : mem_type;
begin
 q <= mem(conv_integer(wr_addr));
 process (clk, we, addr)
 begin
 if rising_edge (clk) then
 if we = '1' then
 mem (conv_integer (addr)) <= d;
 end if;
 wr_addr <= addr;
 end if;
 end process;
end rtl;
```

## Corrected Verilog Test Case

To eliminate the warning in the Verilog test case, change the width of the write address as shown in the corrected test case below.

```
module ramtest (z, raddr, d, waddr, we, clk);
output [7:0] z;
input [7:0] d;
input [3:0] raddr;
input [3:0] waddr;
input we;
input clk;
reg [3:0] mem [7:0];
assign z = mem[raddr];

always @(posedge clk) begin
 if (we)
 mem[waddr] = d;
 end
endmodule
```

This warning is generated when a RAM exists in the RTL code. The warning is accompanied by the following note

@N: Found RAM mem, depth=8, width=4

which indicates the size of the RAM inferred in the code.

## CL134

### **@N: Found RAM <mem>, depth=<128>, width=<8>.**

A RAM was detected. The compiler detects all high-level operators such as RAMs, ROMs, adders, and multipliers at compile stage and optimally maps the operators using technology-specific resources and primitives. The message indicates the depth and the width of the RAM. The following test case has an address width of 7 bits (128) and a data bit width of 8.

```
module ram_test (q, a, d, we, clk);
 output [7:0] q;
 input [7:0] d;
 input [6:0] a;
 input clk, we;
 reg [6:0] read_add;
 /* The array of an array register ("mem") the RAM will be
 inferred from */
 reg [7:0] mem [127:0] /* synthesis syn_ramstyle = "no_rw_check" */;
 assign q = mem[read_add];

 always @(posedge clk) begin
 read_add <= a;
 if(we)
 /* Register RAM Data */
 mem[a] <= d;
 end
endmodule
```

## Action

Make sure that all of the intended RAMs are detected by the compiler so that the correct resources are used to implement these structures in the mapper. RAMs are typically placed in Block RAMs. Specific implementation of the RAM as Select RAM, logic, or Block RAM can be forced by using the attribute `syn_ramstyle` on the RAM register.

This note appears whenever a RAM exists in the RTL code.

## CL135

### **@N: Found seqShift <shift>, depth=<10>, width=<8>**

A sequential shifter was detected. The compiler detects all high-level operators such as RAMs, ROMs, sequential shifts, adders, and multipliers at compile stage and optimally maps the operators using technology-specific resources and primitives. The message indicates the depth and the width of the shifter. The following test case address depth is 10 bits and a data bit width of 8.

```
module srle_example (clk, enable, data_in, result);
parameter cycle=10;
parameter width = 8;
input clk, enable;
input [0:width] data_in;
output [0:width] result;
reg [0:width-1] shift [cycle-1:0];
integer i;

always @(posedge clk)
begin
 if (enable == 1) begin
 for (i = (cycle-1);i >0; i=i-1) shift[i] =
 shift[i-1];
 shift[0] = data_in;
 end
end
assign result = shift[cycle-1];
endmodule
```

## Action

Make sure that all the intended sequential shifters are detected by the compiler so that the right resources are used to implement these structures in the mapper. For some families, the sequential shifters are mapped to technology-specific SRL primitives.

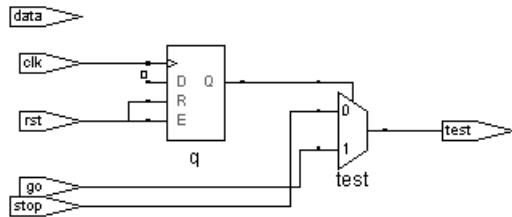
## CL138

### **@W: Removing register '<q>' because it is only assigned 0 or its original value**

A register was inferred with either a constant value of 0 or 1 or its old value. This register is optimized away. A constant value is propagated through the logic and further optimizations are done in the synthesis tool. In the test case below, q is a register that always has a 0 value (see following schematic) and is subsequently removed during synthesis.

```
// assign test = q ? go : stop;
// evaluates to output port test assigned the input stop.
module dff_1 (rst, data, clk, go, stop, test);
output test;
input data, rst, clk, go, stop;
reg q;

always @ (posedge clk)
begin
if (rst)
 q <= data & 1'b0;
else
 q <= q;
end
assign test = q ? go : stop;
endmodule
```



## Action

Make sure that the registers written in the code do not have constant values. The synthesis tool optimizes these and propagates the constants through the path. To eliminate this warning in the above test case, remove the constant assignment as shown in the corrected test case below.

```

// assign test = q ? go : stop;
// evaluates to output port test assigned the input stop.
module dff_1 (rst, data, clk, go, stop, test);
output test;
input data, rst, clk, go, stop;
reg q;

always @ (posedge clk)
begin
 if (rst)
 q <= data;
 else
 q <= q;
end
assign test = q ? go : stop;
endmodule

```

You can also use the `syn_preserve` directive to preserve a register. This directive can be applied globally to a module or locally to an individual register. To apply the directive to the `q` register, add the directive to the `reg` statement as shown in the example below.

```
// assign test = q ? go : stop;
// evaluates to output port test assigned the input stop.
module dff_1 (rst, data, clk, go, stop, test);
output test;
input data, rst, clk, go, stop;
reg q /* synthesis syn_preserve =1 */;

always @ (posedge clk)
begin
if (rst)
 q <= data & 1'b0;
else
 q <= q;
end
assign test = q ? go : stop;
endmodule
```

## CL138

### **@W: Removing register '<q>' because it is only assigned 0 or its original value.**

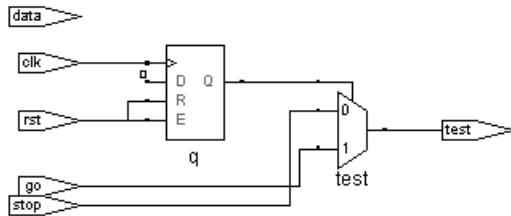
This warning appears when the compiler infers a register with either a constant value of 0 or its original value. This register is optimized away. A constant value is propagated through the logic and further optimizations are done in the synthesis tool. In the test case below, q is a register that always has a 0 value (see following schematic) and is subsequently removed during synthesis.

```
// assign test = q ? go : stop;
// evaluates to output port test assigned the input stop.
module dff_1 (rst, data, clk, go, stop, test);
output test;
input data, rst, clk, go, stop;
reg q;
```

```

always @ (posedge clk)
begin
 if (rst)
 q <= data & 1'b0;
 else
 q <= q;
end
assign test = q ? go : stop;
endmodule

```



## Action

Make sure that the registers written in the code do not have constant values. The synthesis tool optimizes these and propagates the constants through the path. To eliminate this warning in the above test case, change the q assignment in the always block.

```

// assign test = q ? go : stop;
// evaluates to output port test assigned the input stop.
module dff_1 (rst, data, clk, go, stop, test);
 output test;
 input data, rst, clk, go, stop;
 reg q;

 always @ (posedge clk)
 begin
 if (rst)
 q <= data;
 else
 q <= q;
 end
 assign test = q ? go : stop;
endmodule

```

You can also use the `syn_preserve` directive to preserve a register. This directive can be applied on a module or an individual register. To apply the directive to the `q` register, add the directive to the `reg` statement as follows.

```
// assign test = q ? go : stop;
// evaluates to output port test assigned the input stop.
moduledff_1(rst, data, clk, go, stop, test);
output test;
input data, rst, clk, go, stop;
reg q /* synthesis syn_preserve =1 */;

always @ (posedge clk)
begin
 if (rst)
 q <= data & 1'b0;
 else
 q <= q;
end
assign test = q ? go : stop;
endmodule
```

## CL144

**@W: Entity port `<sum[<-8,<-1>]>` has negative indices. It is remapped to positive range `<[<0>,<7>]>`**

The compiler detected a negative indices for ranges in the VHDL port list. The compiler converts the negative indices to a positive range and maintains the direction (ascending/descending) as specified in the HDL code. In the following test case, the port `sum` has negative indices (-8 to -1).

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity adder is
 port (a, b: in std_logic_vector (0 to 7);
 sum: out std_logic_vector (-8 to -1));
end adder;
```

---

```
architecture behave of adder is
begin
 sum <= a + b;
end behave;
```

## Action

Due to the conversion of -ve ranges to +ve ranges in the VHDL design, possible mismatches can exist in RTL/post synthesis simulation. It is always advisable to use positive ranges for the port list. To eliminate the warning in the above test case, change the port sum indices to a positive range (0 to 7).

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity adder is
 port (a, b: in std_logic_vector (0 to 7);
 sum: out std_logic_vector (0 to 7));
end adder;

architecture behave of adder is
begin
 sum <= a + b;
end behave;
```

# CL146

## **@E: Generic <k> found in component <and\_gate> but not in entity <and\_gate>**

The name of the generic declared in a component was different from the name of the generic declared in the actual entity. In the test case below, the name of the generic declared in component and\_gate is k, but the compiler cannot find a generic with the name k in entity and\_gate which results in the error.

```
library ieee;
use ieee.std_logic_1164.all;
```

```

entity and_gate is
generic (p:natural:=2);
 port (a,b : in std_logic_vector(p-1 downto 0);
 c: out std_logic_vector(p-1 downto 0));
end and_gate;

architecture df of and_gate is
begin
 c<= a and b;
end df;

library ieee;
use ieee.std_logic_1164.all;
use work.and_gate.all;

entity cl146 is
 port (a,b : in std_logic_vector(1 downto 0);
 e: out std_logic_vector(1 downto 0));
end cl146;

architecture df of cl146 is
component and_gate
generic(k:natural:=2);
 port (a,b : in std_logic_vector(p-1 downto 0);
 c: out std_logic_vector(p-1 downto 0));
end component;

begin
 a1: and_gate port map (a,b,e);
end df;

```

## Action

Make sure that the generic name is the same in both the entity and component definitions as shown in the test case below.

```

library ieee;
use ieee.std_logic_1164.all;

entity and_gate is
generic (p:natural:=2);
 port (a,b : in std_logic_vector(p-1 downto 0);
 c: out std_logic_vector(p-1 downto 0));
end and_gate;

```

```
architecture df of and_gate is
begin
 c<= a and b;
end df;

library ieee;
use ieee.std_logic_1164.all;
use work.and_gate.all;

entity cl146 is
 port (a,b : in std_logic_vector(1 downto 0);
 e: out std_logic_vector(1 downto 0));
end cl146;

architecture df of cl146 is
component and_gate
generic(p:natural:=2);
 port (a,b : in std_logic_vector(p-1 downto 0);
 c: out std_logic_vector(p-1 downto 0));
end component;

begin
 a1: and_gate port map (a,b,e);
end df;
```

The above error also can be eliminated by using a configuration statement where the generic in the component is bound to the generic in the entity using a generic map statement as shown in the following code segment.

```
component and_gate
generic(k:natural:=2);
 port (a,b : in std_logic_vector(k-1 downto 0);
 c: out std_logic_vector(k-1 downto 0));
end component;

for a1: and_gate use entity work.and_gate generic map (p=>k);
```

## CL153

### **@W: Unassigned bits of <x> are referenced and tied to 0 -- simulation mismatch possible.**

The bit or bit slices of internal registers/signals are undriven, but are referenced in assign statements or comparison statements. The compiler treats unassigned bits as unknowns during simulation and ties them to 0 or 1 which can cause a mismatch between the RTL simulation and the post synthesis simulation. In the following test case, x is undriven, however it is referenced in the assign statement.

```
module test (a, b);
 input a;
 output b;
 reg x;
 assign b = x;
endmodule
```

### Action

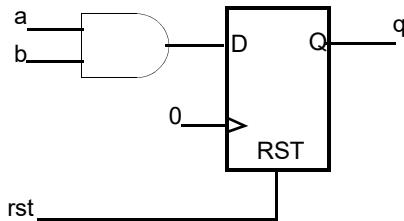
Make sure that there are no undriven bits in the design. It is possible that due to constant propagation and other optimizations, certain signals may become undriven. To eliminate this warning (assuming x is driven), edit the HDL code as shown in the corrected test case below.

```
module test (a, b);
 input a;
 output b;
 reg x;
 always @ (a)
 x <= a;
 assign b = x;
endmodule
```

# CL154

## @W: Clock on register <q> tied to a constant

The clock in a design is either tied to a constant as shown in the following schematic or is floating.



## Verilog Test Case

In the following test case, clk is tied to 0. The register is optimized away during synthesis.

```

module test (a, b, rst, q);
 input a, b, rst;
 output q;
 reg q;
 wire clk;
 assign clk = 1'b0;

 always @ (posedge clk)
 begin
 if (rst)
 q <= 1'b0;
 else
 q <= a&b;
 end
endmodule

```

## VHDL Test Case

In the following test case, clk is tied to 0. The register is optimized away during synthesis.

```
library ieee; use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity test is
 port (a,b,rst: in std_logic;
 q: out std_logic);
end entity;

architecture only of test is signal temp:
 std_logic_vector (7 downto 0);
signal clk: std_logic;
begin
 clk <= '1';
 process (clk)
 begin
 if (rst = '1') then
 q <= '1';
 elsif (clk'event and clk = '1')
 then q <= a and b;
 end if;
 end process;
end only;
```

## Action

Make sure this is the intended behavior. Otherwise, to avoid this warning, make sure that the clocks of sequential elements are not connected to constants in the design.

# CL155

### **@W: Reset/Set on register ‘q’ is tied to a constant**

The reset or set in a design is either tied to a constant or is floating. Thus in such cases the reset/set signal has no affect on the register.

In the following test case, rst is floating and tied to zero, hence the reset signal has no affect on the register

```
module test (a, b, clk, q);
 input a, b, clk;
 output q;
 reg q;
 wire rst;

 always @ (posedge clk)
 begin
 if (rst)
 q <= 1'b0;
 else
 q <= a&b;
 end
endmodule
```

## Action

Make sure this is the intended behavior. If true reset behavior is intended, edit the code to represent this. To eliminate the warning in the above test case, add an `rst` signal to the input port list.

```
module test (a, b, clk, rst, q);
 input a, b, clk, rst;
 output q;
 reg q;
 wire rst;

 always @ (posedge clk)
 begin
 if (rst)
 q <= 1'b0;
 else
 q <= a&b;
 end
endmodule
```

## CL156

**@W: \*Input <c[0]> to expression [<mux>] has undriven bits that are tied to 0 -- simulation mismatch possible.**

Wires in the design are used, but not assigned a value in their corresponding module description. In the following test case, wire c is used in the assign statement without having been assigned a value.

```
// Comparator
module compare (equal, a, b);
parameter size = 1;
output equal;
input [size-1:0] a, b; // declare inputs
wire equal;
wire [size-1:0] c;
assign equal = c? (a == b) : 1'b0;
endmodule
```

### Action

To eliminate the above warning, edit the code so that either all bits of c are assigned values or define c as an input port as shown in the corrected test case below.

```
// Comparator
module compare (equal, a, b, c);
parameter size = 1;
output equal;
input [size-1:0] a, b, c; // declare inputs
wire equal;
wire [size-1:0] c;
assign equal = c? (a == b) : 1'b0;
endmodule
```

## CL157

**@W: Output <c> has undriven bits; assigning undriven bits to 0. Simulation mismatch possible. Assign all bits of the output.**

In an RTL simulation, all undriven bits are treated as unknowns. While synthesizing, there is no hardware to produce an unknown. Hence the compiler ties it to high impedance by placing a tristate buffer and ties its enable to 0.

In the following test case, c is undriven.

```
module test (a, b, c, ce, clk, rst);
 input a, b, clk, rst;
 output c, ce;
 reg ce;

 always @ (posedge clk)
 begin
 if (rst)
 ce <= 1'b0;
 else
 ce <= a & b;
 end
endmodule
```

### Action

Make sure this is the intended behavior. The compiler represents the undriven bits as either constant 0 or 1 which can cause an RTL versus post synthesis simulation mismatch. To eliminate the warning, edit the code so that c is not undriven.

```
module test (a, b, c, ce, clk, rst);
 input a, b, clk, rst;
 output c, ce;
 reg ce;
 reg c;

 always @ (posedge clk)
 begin
 if (rst)
 begin
 ce <= 1'b0;
 c <= 1'b0;
 end
 end
endmodule
```

```
end
else
begin
 ce <= a & b;
 c <= a | b;
end
end
endmodule
```

## CL158

### @W: Inout <c> is unused

The inouts of a design are declared, but not read from (input function) or assigned to (output function) a corresponding module description.

#### Verilog Test Case

In the following test case, inout port c is not used within the module.

```
module compare(equal, a, b, c);
parameter size = 1;
output equal;
input [size-1:0] a, b;
inout [size-1:0] c; // declare inputs
wire equal;
wire [size-1:0] c;
assign equal = a == b;
endmodule
```

#### VHDL Test Case

In the following test case, inout port c is not used within the module.

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

```

entity dff2 is
 port (a,b: in std_logic_vector (7 downto 0) ;
 c: inout std_logic_vector (7 downto 0);
 equal: out std_logic_vector (7 downto 0));
end entity;

architecture only of dff2 is
--signal test: std_logic_vector (2 downto 0);
begin
 equal <= (a and b) ;
end only;

```

## Action

The top-level port interface needs to be maintained after synthesis. Make sure that all the ports are used within the module/architecture of the design. This prevents unnecessary pin allocations on the top-level design.

## Corrected Verilog Test Case

To eliminate this warning in the Verilog test case, edit the code to use c as an input or an output.

```

module compare(equal, a, b, c);
parameter size = 1;
output equal;
input [size-1:0] a, b;
inout [size-1:0] c; // declare inputs
wire equal;
wire [size-1:0] c;
assign equal = c? a == b : 1'b0;
endmodule

```

## Corrected VHDL Test Case

To eliminate the warning in the VHDL test case, edit the code to use c as an input or an output.

```

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

```

```

entity dff2 is
 port (a,b: in std_logic_vector (7 downto 0) ;
 c: inout std_logic_vector (7 downto 0);
 equal: out std_logic_vector (7 downto 0));
end entity;

architecture only of dff2 is
--signal test: std_logic_vector (2 downto 0);
begin
 equal <= (a and b) and c ;
end only;

```

## CL159

### **@W: Input <b> is unused.**

Inputs in the top level of the design are declared, but not read from (input function) a corresponding module description.

#### Verilog Test Case

In the following test case, input port b is not used within the module.

```

module compare(equal_xr, a, b);
parameter size = 8;
output [size -1:0] equal_xr;
input [size-1:0] a, b;
assign equal_xr = a == (8'h42);
endmodule

```

#### VHDL Test Case

In the following test case, input port b is not used within the module.

```

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

```

```
entity dff2 is
 port (a,b: in std_logic_vector (7 downto 0) ;
 c: in std_logic_vector (7 downto 0);
 equal: out std_logic_vector (7 downto 0));
end entity;

architecture only of dff2 is
begin
 equal <= a and c ;
end only;
```

## Action

The top-level port interface must be maintained after synthesis. To prevent unnecessary pin allocations on the top-level design, make sure that all ports are used within the module/architecture of the design.

## Corrected Verilog Test Case

To eliminate the warning in the Verilog test case, edit the code to use b as an input.

```
module compare(equal_xr, a, b);
parameter size = 8;
output [size -1:0] equal_xr;
input [size-1:0] a, b;
assign equal_xr = b? a == (8'h42);
endmodule
```

## Corrected VHDL Test Case

To eliminate the warning in the VHDL test case, edit the code to use b as an input.

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

entity dff2 is
 port (a,b: in std_logic_vector (7 downto 0) ;
 c: in std_logic_vector (7 downto 0);
 equal: out std_logic_vector (7 downto 0));
end entity;
```

```
architecture only of dff2 is
begin
 equal <= (a and b) and c;
end only;
```

## CL161

### @W: syn\_keep not supported on bidirectional ports (port:<b>)

A syn\_keep synthesis directive has been placed on a bidirectional port. The directive is ignored. The compiler and mapper optimize out or remove logic that is unnecessary or redundant. The syn\_keep directive preserves nets from being optimized away. This directive also preserves registers and instantiated components. The following test case uses syn\_keep on ports.

```
library ieee;
use ieee.std_logic_1164.all;

entity dff_1 is
 port (a,clk: in std_logic;
 b: inout std_logic);
attribute syn_keep : boolean;
attribute syn_keep of b : signal is true;
end dff_1;

architecture behave of dff_1 is
--attribute syn_keep : boolean;
--attribute syn_keep of b : port is true;
--signal temp1,temp2 : std_logic;
begin
 process (clk)
 begin
 if (clk = '1' and clk'event) then
 b <= a;
 end if;
 end process;
end;
```

## Action

Because b is an inout port and is maintained at the top level, syn\_keep does not add any value. If a port in a lower-level module must be preserved, use syn\_hier. Use syn\_keep to keep nets from getting merged or removed during synthesis due to optimizations performed by both the compiler and the mapper.

In the following test case, nets keep1 and keep2 would merge if syn\_keep was not used since they are being driven by an AND gate (in1 and in2). The following test case shows the proper use of the syn\_keep directive.

```
library ieee;
use ieee.std_logic_1164.all;

entity example2 is
 port (in1, in2: in bit;
 clk : in bit;
 out1, out2 : out bit);
end example2;

architecture rtl of example2 is
attribute syn_keep : boolean;
signal and_out, keep1, keep2: bit;
attribute syn_keep of keep1, keep2 : signal is true;
begin
and_out <= in1 and in2;
keep1 <= and_out;
keep2 <= and_out;
process(clk)
begin
 if (clk'event and clk = '1') then
 out1 <= keep1;
 out2 <= keep2;
 end if;
end process;
end rtl;
```

# CL167

## @W: Input <b> of instance <U1> is floating

The scalar signals connected to the input of an instance are not driven in the design. In the following test case, the internal signal `temp` (connects input `b` of the instance `U1`) is undriven.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity test is
 port (a, b: in std_logic;
 clk, set: in std_logic;
 out1: out std_logic);
end entity;

architecture only of test is
begin
 process (clk, set)
 begin
 if (set = '1') then
 out1 <= '1';
 elsif clk'event and clk = '1' then
 out1 <= a and b;
 end if;
 end process;
end only;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity test_top is
 port (a1, b1: in std_logic;
 clk1, set1: in std_logic;
 out1: out std_logic);
end entity;

architecture only of test_top is
component test is
 port (a, b: in std_logic;
 clk, set: in std_logic;
 out1: out std_logic);
end component;
```

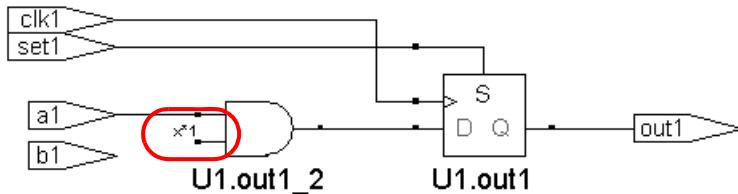
```

signal temp:std_logic;
begin
U1: test port map
 (a => a1,
 b => temp,
 clk => clk1,
 set => set1,
 out1 => out1
);
end only;

```

## Action

In the above test case, since temp is unconnected, its value is a ‘don’t care’ which results in the AND gate equating to ‘0’ in the RTL view, and the mapped netlist connecting the flip-flop to a constant.



Make sure this is the intended behavior. To avoid this warning, always drive the inputs with valid values or inputs from the top level. To eliminate the warning in the above test case and include the top-level input b1 instead of an unassigned signal, edit the instantiation as shown in the corrected test case below.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity test is
 port (a, b: in std_logic;
 clk, set: in std_logic;
 out1: out std_logic);
end entity;

```

```
architecture only of test is
begin
 process (clk, set)
 begin
 if (set = '1') then
 out1 <= '1';
 elsif clk'event and clk = '1' then
 out1 <= a and b;
 end if;
 end process;
end only;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity test_top is
 port (a1, b1: in std_logic;
 clk1, set1: in std_logic;
 out1: out std_logic);
end entity;

architecture only of test_top is
component test is
 port (a, b: in std_logic;
 clk, set: in std_logic;
 out1: out std_logic);
end component;

signal temp:std_logic;
begin
U1: test port map
 (a => a1,
 b => b1,
 clk => clk1,
 set => set1,
 out1 => out1
);
end only;
```

# CL168

## @W: Pruning instance <instanceName> - not in use ...

The compiler removed an instance that is instantiated in the HDL code, but its outputs do not drive the outputs of the top-level design. The compiler optimizes this instance away since the same functionality can be achieved with less logic.

### Verilog Test Case

In the following test case, the module rotate is instantiated in the top-level design, but because its outputs are not used, the instance rotate\_1 is removed.

```
// Example of a hierarchical design
// -----
// First define the lower level modules: mux, reg8, & rotate
// -----
module mux (out, a, b, sel);
 input [7:0] a, b;
 input sel;
 output [7 : 0] out;
 assign out = sel ? a : b;
endmodule

module reg8 (q, data, clk, rst); // eight bit register
 output [7:0] q;
 input [7:0] data;
 input clk, rst;
 reg [7:0] q;

 always @(posedge clk or posedge rst)
 begin
 if (rst)
 q = 0;
 else
 q = data;
 end
endmodule
```

```

module rotate (q, data, clk, r_l, rst); // rotates bits or loads
output [7:0] q;
input [7:0] data;
input clk, r_l, rst;
reg [7:0] q;

// when r_l is high, it rotates; if low, it loads data
always @(posedge clk or posedge rst)
begin
 if (rst)
 q = 8'b0;
 else if (r_l)
 q = {q[6:0], q[7]};
 else
 q = data;
end
endmodule

// Top.v
module top2 (q, a, b, sel, r_l, clk, rst);
output [7:0] q;
input [7:0] a, b;
input sel, r_l, clk, rst;
wire [7:0] mux_out, reg_out, q_out;
wire rst;
mux mux_1 (.out(mux_out), .a(a), .b(b), .sel(sel));
// notice that port connections listed by name can be in any order
reg8 reg8_1 (.clk(clk), .data(mux_out), .q(q), .rst(rst));
// can mix port connections "in order" (below) with port
connections
// "by name" (above)
rotate rotate_1 (q_out, a, clk, r_l, rst);
endmodule

```

## VHDL Test Case

In the following test case, the module `rotate` is instantiated in the top-level design, but because its outputs are not used, the instance `mux_inst` is removed.

```

library ieee;
use ieee.std_logic_1164.all;

```

```
entity muxhier is
 port (outvec: out std_logic_vector (7 downto 0);
 a_vec, b_vec: in std_logic_vector (7 downto 0);
 sel: in std_logic);
end muxhier;

architecture mux_design of muxhier is -- mux
begin
with sel select
 outvec <= a_vec when '1',
 b_vec when '0',
 "XXXXXXXX" when others;
end mux_design;

library ieee;
use ieee.std_logic_1164.all;

entity reg8 is
 port (q: buffer std_logic_vector (7 downto 0);
 data: in std_logic_vector (7 downto 0);
 clk, rst: in std_logic);
end reg8;

architecture reg8_design of reg8 is -- eight bit register
begin
process (clk, rst)
begin
 if rst = '1' then
 q <= X"00";
 elsif rising_edge(clk) then
 q <= data;
 end if;
end process;
end reg8_design;

library ieee;
use ieee.std_logic_1164.all;

entity rotate is
 port (q: buffer std_logic_vector (7 downto 0);
 data: in std_logic_vector (7 downto 0);
 clk, rst, r_l: in std_logic);
end rotate;
```

```
architecture rotate_design of rotate is
-- rotates bits or loads
-- when r_l is high, it rotates; if low, it loads data
begin
 process (clk, rst)
 begin
 if rst = '1' then
 q <= X"00";
 elsif rising_edge(clk) then
 if r_l = '1' then
 q <= q (6 downto 0) & q (7);
 else
 q <= data;
 end if;
 end if;
 end process;
end rotate_design;

-- Top level

library ieee;
use ieee.std_logic_1164.all;

entity top_level is
 port (q:buffer std_logic_vector (7 downto 0);
 a, b: in std_logic_vector (7 downto 0);
 sel, r_l, clk, rst: in std_logic);
end top_level;

architecture structural of top_level is
component muxhier -- component declaration for mux
 port (outvec: out std_logic_vector (7 downto 0);
 a_vec, b_vec: in std_logic_vector (7 downto 0);
 sel: in std_logic);
end component;

component reg8 -- component declaration for reg8
 port (q: buffer std_logic_vector (7 downto 0);
 data: in std_logic_vector (7 downto 0);
 clk, rst: in std_logic);
end component;
```

```
component rotate -- component declaration for rotate
 port (q: buffer std_logic_vector (7 downto 0);
 data: in std_logic_vector (7 downto 0);
 clk, rst, r_l: in std_logic);
end component;

-- declare the internal signals here
signal mux_out, reg_out: std_logic_vector (7 downto 0);

begin -- structural description begins
-- instantiate a mux, name it inst1, and wire it up
-- here we connect the mux with positional port mapping (by
position)
mux_inst: muxhier port map (mux_out, a, b, sel);

-- instantiate a rotate, name it inst2, and wire it up
rotate_inst: rotate port map (q , reg_out, clk, rst, r_l);

-- instantiate a reg8, name it inst3, and wire it up
-- reg8 is connected with named port mapping (by name)
-- the port connections can be given in any order
-- Note that the local signal names are on the right of the
-- '>=' mapping operators, and the signal names from the
-- component declaration are on the left.

reg_inst: reg8
 port map (clk => clk, data => open,
 q => reg_out, rst => rst);

end structural;
```

## Action

Make sure that all instantiated instances in the code drive the logic associated with the output of the design. For primitives, use the `syn_noprune` synthesis directive to keep the instance from being removed, even if the outputs are not connected.

```
BUF U1 (O, I); /* synthesis syn_noprune =1 */
```



## CHAPTER 18

# CL Messages 169 – 322

---

## CL169

**@W: Pruning unused register <regName>. Make sure that there are no unused intermediate registers.**

The compiler pruned an unused intermediate register.

### Verilog Test Case

In the following test case, intermediate register temp is used to store input that is not used anywhere in the design. The compiler prunes the register and issues the warning.

```
module ff_check (q, clk, en, d);
 input clk,d,en;
 output reg q;
 reg temp;

 always @ (posedge clk)
 if (en)
 begin
 temp = d;
 q = d;
 end
 endmodule
```

## Action

Make sure that there are no unused intermediate registers. The above test case can be corrected by removing unused register `temp` as shown in the corrected test case below.

```
module ff_check (q, clk, en, d);
 input clk,d,en;
 output reg q;

 always @ (posedge clk)
 if (en)
 begin
 q = d;
 end
 endmodule
```

## VHDL Test Case

In the following test case, intermediate register `temp` is used to store input that is not used anywhere in the design. The compiler prunes the register and issues the above warning.

```
entity ff_check is
 port (d,en,clk: in bit;
 q: out bit);
end;

architecture test of ff_check is
signal temp: bit;
begin
 process (clk,en,d)
 begin
 if(clk'event and clk='1') then
 if(en='1') then
 temp<=d;
 q<=d;
 end if;
 end if;
 end process;
end;
```

## Action

Make sure that there are no unused intermediate registers. The above test case can be corrected by removing unused register `temp` as shown in the corrected test case below.

```
entity ff_check is
 port (d,en,clk: in bit;
 q: out bit);
end;

architecture test of ff_check is
signal temp: bit;
begin
 process (clk,en,d)
 begin
 if(clk'event and clk='1') then
 if(en='1') then
 q<=d;
 end if;
 end if;
 end process;
end;
```

## CL172

### @E: Only one always block can assign a given variable <q>

More than one always block assigned a value to the same variable. In simulation this is possible, however no equivalent logic can be created in the hardware. In the following test case, `reg` signal `q` is assigned a value at both edges of the clock which results in the error.

```
module dff(q, data1, data2, clk);
output q;
input data1, data2, clk;
reg q;
always @(posedge clk)
begin
 q <= data1;
end
```

```
always @(negedge clk)
begin
 q <= data2;
end

endmodule
```

## Action

Make sure that no more than one always block assigns a value to a reg signal. If you require a structure where the output of the circuit is dependent on both edges of the clock (for example, dual data-rate registers), define separate variables for the always blocks as shown in the corrected test case below.

```
module dff(q, data1, data2, clk);
output q;
input data1, data2, clk;
reg q1, q2;
wire q;

always @(posedge clk)
begin
 q1 <= data1;
end

always @(negedge clk)
begin
 q2 <= data2;
end
assign q = q1 | q2;

endmodule
```

# CL177

## **@N: Sharing sequential element <out2>. Add a syn\_preserve attribute to the element to prevent sharing.**

More than one sequential element, driven by the same inputs, has been detected. The compiler merges the sequential elements. The fanout on the merged element increases by the number of sequential elements merged. In the test case below, the out1 and out2 outputs are driven by the same sequen-

tial logic, hence the two registers (out1 and out2) are merged into a single register (out1). The fanout on this register is two as it drives outputs out1 and out2.

```
module counter2 (out1,out2, data, rst, clk, en1);
output [7:0] out1,out2;
input [7:0] data;
input rst,clk,en1;
reg [7:0] out1,out2;

// create the 8-bit register
always @(posedge clk or negedge rst)
begin
 if (!rst) begin
 out1 <= 8'b0;
 out2 <= 8'b0;
 end
 else begin
 out1 <= data;
 out2 <= data;
 end
end
endmodule
```

## Action

Make sure merging is acceptable to the design. Sequential merging and replication are done throughout the synthesis process depending on the performance goals. To disable sequential optimizations, use the `syn_preserve` directive.

# CL179

## @W: Found combinational loop at <*un1\_qrs[0]*>

A combinational loop (a loop where the output of the combinational gates drives the inputs) is found in the HDL code (combinational loops can cause unexpected behavior and mismatches in the RTL to post-synthesis simulation). In the following test case, qrs is an output of an assignment that uses itself as one of the inputs as shown in the schematic below.

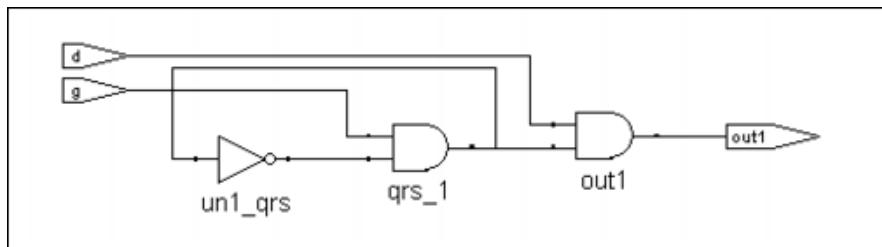
```

library ieee;
use ieee.std_logic_1164.all;

entity dff2 is
 port (g,d: in std_logic;
 out1: out std_logic);
end entity;

architecture only of dff2 is
begin
 process(g,d)
 variable qrs:std_logic;
 begin
 qrs:= not qrs and g;
 out1 <= (qrs and d);
 end process;
end only;

```



## Action

To avoid the above warning, make sure that there are no combinational loops in the design.

```

library ieee;
use ieee.std_logic_1164.all;

entity dff2 is
 port (g,d: in std_logic;
 out1: out std_logic);
end entity;

```

```
architecture only of dff2 is
begin
 process(g,d)
 variable qrs:std_logic;
 begin
 qrs:= g;
 out1 <= (d and qrs);
 end process;
end only;
```

## CL181

### **@W: Found combinational loop**

A combinational loop (a loop where the output of the combinational gates drives the inputs) is found in the HDL code, but the specific source for the loop cannot be readily determined.

#### Action

To avoid the above warning, make sure that there are no combinational loops in the design. To help locate the HDL source responsible for the combinational loop, click in the source location field in the message.

## CL182

### **@W: Signal read but some bits are never set, assigning initial value to unassigned bits: '<temp1>'**

When looping assignments exist in a design with signals that are not initialized, a race condition is present. The compiler assigns initial values so it can build the hardware correctly.

## Verilog Test Case

In the following test case, the temp1 and temp2 signals drive each other. They are not initialized and have no other signals driving them. This condition results in these signals having unknown values. To avoid the potential race condition, the compiler initializes one of the signals.

```
module test (a, b, clk);
 input a, clk;
 output b;
 wire temp1;
 reg temp2;
 assign b = temp2;
 assign temp1 = temp2;

 always @ (a)
 begin
 temp2 = temp1;
 end
endmodule
```

## VHDL Test Case

In the following test case, the temp1 and temp2 signals drive each other. They are not initialized and have no other signals driving them. This condition results in these signals having unknown values. To avoid the potential race condition, the compiler initializes one of the signals.

```
library ieee;
use ieee.std_logic_1164.all;

entity dff_1 is
 port (a,clk: in std_logic;
 b: out std_logic);
end dff_1;

architecture behave of dff_1 is
signal temp1,temp2 : std_logic;
begin
b <=temp2;
temp1 <= temp2;
process (a)
begin
 temp2 <= temp1;
end process;
end behave;
```

## Action

To avoid having signals that drive each other in a loop without any other primary input or valid signal driving them. Failure to meet this condition results in combinational loops with initial assumed values which can cause RTL post synthesis simulation mismatches.

# CL185

### **@E: Found self-reference combinational loop at <qrs[1]>**

A signal of array type was assigned to itself which created a self-referencing combinational loop.

#### Verilog Test case

In the test case below, signal qrs is assigned to itself which causes the compiler to error out.

```
module cl185_v (input a,b,output reg out_1);
reg [1:0]qrs;
always @(a,b,qrs)
begin
 qrs<= qrs;
 out_1<= (qrs[0] & qrs[1] & a & b);
end
endmodule
```

## Action

Do not use self-referencing combinational loops.

```
module cl185_v (input a,b,output reg out_1);
reg [1:0]qrs;
always @(a,b,qrs)
begin
 qrs<= {a,b};
 out_1<= (qrs[0] & qrs[1]);
end
endmodule
```

## VHDL Test case

In the test case below, vector signal qrs is assigned to itself which causes the compiler to error out.

```
library ieee;
use ieee.std_logic_1164.all;

entity cl185 is
 port (g,d: in std_logic;
 out1: out std_logic);
end entity;

architecture only of cl185 is
signal qrs:std_logic_vector (1 downto 0);
begin
 process(g,d,qrs)
 begin
 qrs <= qrs;
 out1 <= (g and d and qrs(0) and qrs(1)) ;
 end process;
end only;
```

## Action

Do not use self-referencing combinational loops.

```
library ieee;
use ieee.std_logic_1164.all;

entity cl185 is
 port (g,d: in std_logic;
 out1: out std_logic);
end entity;

architecture only of cl185 is
signal qrs:std_logic_vector (1 downto 0);
begin
 process(g,d,qrs)
 begin
 qrs <= (1=>g,0=>d);
 out1 <= qrs(0) and qrs(1) ;
 end process;
end only;
```

# CL186

## @E: Found self-reference combinational loop at <qrs>

A signal was assigned to itself creating a self-referencing combinational loop.

### Verilog Test Case

In the test case below, signal qrs is assigned to itself which causes the compiler to error out.

```
module cl186_v (input a,b,output reg out_1);
reg qrs;
always @(a,b,qrs)
begin
 qrs<= qrs;
 out_1<= (qrs & a & b);
end
endmodule
```

### Action

Do not use self-referencing combinational loops.

```
module cl186_v (input a,b,output reg out_1);
reg qrs;
always @(a,b,qrs)
begin
 qrs<= a;
 out_1<= (qrs & b);
end
endmodule
```

### VHDL Test Case

In the test case below, signal qrs is assigned to itself which causes the compiler to error out with the above message.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity cl186 is
 port (g,d: in std_logic;
 out1: out std_logic);
end entity;

architecture only of cl186 is
signal qrs:std_logic;
begin
 process(g,d,qrs)
 begin
 qrs <= qrs;
 out1 <= (g and d and qrs);
 end process;
end only;
```

## Action

Do not use self-referencing combinational loops.

```
library ieee;
use ieee.std_logic_1164.all;

entity cl186 is
 port (g,d: in std_logic;
 out1: out std_logic);
end entity;

architecture only of cl186 is
signal qrs:std_logic;
begin
 process(g,d,qrs)
 begin
 qrs <= d;
 out1 <= (g and qrs) ;
 end process;
end only;
```

# CL188

## @E: Multiple drivers for net <regin> with same enable

Multiple drivers on a net/port are possible when there is more than one driver on the net that is not tri-stated. This check is done by both the compiler and the mapper. However, the compiler does not dissolve hierarchy, and detects multiple drivers only when they are on the same level of hierarchy (inside a module or in the top level).

### Verilog Test Case

The compiler detected tri-states on a net/port with the same enable in the same level of hierarchy.

```
module top (en1, in1, in2, in3, clk, regout);
 input en1, in1, in2, in3, clk;
 output regout;
 reg regout;
 wire tmp1, tmp2, tmp3;
 wire regin;
 assign tmp1 = en1 ? in1 : 1'bz;
 assign tmp2 = en1 ? in2 : 1'bz;
 assign tmp3 = en1 ? in3 : 1'bz;
 assign regin = tmp1;
 assign regin = tmp2;
 assign regin = tmp3;

 always @(posedge clk)
 regout <= regin;

endmodule
```

### VHDL Test Case

The compiler detected tri-states on a net/port with the same enable in the same level of hierarchy.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
```

```
entity adder is
 port (en1,in1,in2,in3,clk: in std_logic;
 regout:out std_logic);
end adder;

architecture behave of adder is
signal tmp1,tmp2,tmp3,regin: std_logic;
begin
 process (en1, in1)
begin
 if (en1 = '1') then
 tmp1 <= in1;
 else
 tmp1 <= 'Z';
 end if;
end process;

process (en1,in2)
begin
 if (en1 = '1') then
 tmp2 <= in2;
 else
 tmp2<= 'Z';
 end if;
end process;

process (en1,in3)
begin
 if (en1 = '1') then
 tmp3<= in3;
 else
 tmp3<= 'Z';
 end if ;
end process;

regin <= tmp1;
regin <= tmp2;
regin <= tmp3;

process (clk)
begin
 if (clk = '1' and clk'event) then
 regout <= regin;
 end if;
end process;

end behave;
```

## Action

Make sure that when there is more than one driver for a net/port, that the drivers are tri-stated. The enables for the tri-state must be mutually exclusive so that there is only one driver for each net/port with respect to a certain condition.

## Corrected Verilog Test Case

Make the enable signals of the tri-state of `regin` are mutually exclusive (different inputs `en1`, `en2`, and `en3`).

```
module top (en1, en2, en3, in1, in2, in3, clk, regout);
 input en1, en2, en3, in1, in2, in3, clk;
 output regout;
 reg regout;
 wire tmp1, tmp2, tmp3;
 wire regin;
 assign tmp1 = en1 ? in1 : 1'bz;
 assign tmp2 = en2 ? in2 : 1'bz;
 assign tmp3 = en3 ? in3 : 1'bz;
 assign regin = tmp1;
 assign regin = tmp2;
 assign regin = tmp3;

 always @(posedge clk)
 regout <= regin;

endmodule
```

## Corrected VHDL Test Case

Make sure the enable signals of the tri-state of `regin` are mutually exclusive (different inputs `en1`, `en2`, and `en3`).

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity adder is
 port (en1,en2,en3,in1,in2,in3,clk: in std_logic;
 regout:out std_logic);
end adder;
```

```
architecture behave of adder is
signal tmp1,tmp2,tmp3,regin: std_logic;
begin
 process (en1, in1)
 begin
 if (en1 = '1') then
 tmp1 <= in1;
 else
 tmp1 <= 'Z';
 end if;
 end process;

 process (en2,in2)
 begin
 if (en2 = '1') then
 tmp2 <= in2;
 else
 tmp2<= 'Z';
 end if;
 end process;

 process (en3,in3)
 begin
 if (en3 = '1') then
 tmp3<= in3;
 else
 tmp3<= 'Z';
 end if ;
 end process;

 regin <= tmp1;
 regin <= tmp2;
 regin <= tmp3;

 process (clk)
 begin
 if (clk = '1' and clk'event) then
 regout <= regin;
 end if;
 end process;

end behave;
```

# CL189

## @N: Register bit <out1> is always <0>.

A register with a constant fixed value was detected, either through optimization or constant propagation, and is being removed. The note identifies the register being removed.

### Verilog Test Case

In the test case below, signal out1 is tied to 0 because the if condition always has zero results.

```
module asyncl (out1, clk, g, d);
 output out1;
 reg out1;
 input g, d,clk;
 always @ (posedge clk)
 if (g & d)
 out1 = !g & out1; // signal out1 is used
 // in the R.H.S and L.H.S
 else // of the expression and no clock is defined
 out1 = out1;
endmodule
```

### VHDL Test Case

In the test case below, the signal out1 is tied to 0 because the if condition always has a 0 result.

```
library ieee;
use ieee.std_logic_1164.all;

entity asyncl is
 port (clk,g,d: in std_logic;
 out1: inout std_logic);
end asyncl;

architecture behave of asyncl is
begin
 process (clk)
 begin
 if (clk ='1' and clk'event) then
```

```

 if (g == '1') then out1 <= (not g) and out1;
 else out1 <= out1;
 end if;
end if;
end process;
end behave;

```

## Action

Make sure that the removal of registers does not affect the intended functionality of the design.

## Corrected Verilog Test Case

To eliminate this warning in the Verilog test case and preserve a register, edit the code to change the register value.

```

module asyncl (out1, clk, g, d);
output out1;
reg out1;
input g, d,clk;

always @ (posedge clk)
if (g & d)
 out1 = g & out1; // signal out1 is used in the R.H.S and
L.H.S
else
 out1 = out1; // of the expression and no clock is defined
endmodule

```

You can also use the `syn_preserve` directive to preserve a register. This directive can be applied globally on a module or locally on a register. To apply the directive to the `out1` register, edit the `reg out1` statement.

```

module asyncl (out1, clk, g, d);
output out1;
reg out1 /* synthesis syn_preserve =1 */;
input g, d,clk;

```

```
always @ (posedge clk)
 if (g & d)
 out1 = !g & out1; // signal out1 is used
 // in the R.H.S and L.H.S
 else // of the expression and no clock is defined
 out1 = out1;
endmodule
```

## CL190

### @W: Optimizing register bit <DATA0[7]> to a constant <0>. To keep the instance, apply constraint syn\_preserve=1 on the instance.

All logic associated with a particular register evaluates the register with a constant value of 0 or 1. The compiler or mapper removes this register. In the following test case, DATA0[0] is always defined as a constant value 0.

```
module prep2 (DATA0, CLK, RST, SEL, LDCOMP, LDPRE, DATA1, DATA2);
output [7:0] DATA0;
input CLK, RST, SEL, LDCOMP, LDPRE;
input [7:0] DATA1, DATA2;
reg [7:0] DATA0;
reg [7:0] highreg_output, lowreg_output; // internal registers
wire compare_output = DATA0 == lowreg_output; // comparator
wire [7:0] mux_output = SEL ? DATA1 : highreg_output; // mux
// registers

always @ (posedge CLK or posedge RST)
begin
 if (RST) begin
 highreg_output = 0;
 lowreg_output = 0;
 end else begin
 if (LDPRE)
 highreg_output = DATA2;
 if (LDCOMP)
 lowreg_output = DATA2;
 end
end
```

```

// counter
always @(posedge CLK or posedge RST)
begin
 if (RST)
 DATA0 = 0;
 else if (compare_output) // load
 begin
 DATA0 = mux_output & 8'b01111111;
 end
 else
 begin
 DATA0 = DATA0 + 1'b0;
 end
 end
endmodule

```

## Action

Make sure that the logic removed does not affect the overall intended functionality. If you want to keep registers from being removed, even if they are tied to constants, use the `syn_preserve` directive. This directive can be applied globally on a module or locally on a register. To eliminate the warning in the above test case, add the `syn_preserve` directive.

```

module prep2 (DATA0, CLK, RST, SEL, LDCOMP, LDPRE, DATA1, DATA2);
output [7:0] DATA0;
input CLK, RST, SEL, LDCOMP, LDPRE;
input [7:0] DATA1, DATA2;
reg [7:0] DATA0 /* synthesis syn_preserve =1 */;
reg [7:0] highreg_output, lowreg_output; // internal registers
wire compare_output = DATA0 == lowreg_output; // comparator
wire [7:0] mux_output = SEL ? DATA1 : highreg_output; // mux
// registers

always @ (posedge CLK or posedge RST)
begin
 if (RST) begin
 highreg_output = 0;
 lowreg_output = 0;
 end else begin
 if (LDPRE)

```

```
 highreg_output = DATA2;
 if (LDCOMP)
 lowreg_output = DATA2;
 end
end

// counter
always @(posedge CLK or posedge RST)
begin
if (RST)
 DATA0 = 0;
else if (compare_output) // load
begin
 DATA0 = mux_output & 8'b01111111;
end
else
begin
 DATA0 = DATA0 + 1'b0;
end
end
endmodule
```

## CL193

### **@W: syn\_probe attribute found on possible state machine: may inhibit state machine extraction**

The `syn_probe` attribute is used to bring out internal nets as output ports for observability and debugging. When `syn_probe` is applied on any net, the compiler places the `syn_keep` attribute to ensure the net is not optimized away during synthesis. The `syn_keep` attribute prevents any optimizations around that net and hence the FSM compiler cannot extract the state machines.

#### Verilog Test Case

In the following test case, state register `state` has an applied `syn_probe` attribute which prevents the compiler from extracting the state machine and results in the above warning.

```
module FSM1 (clk, rst, enable, data_in, data_out, state0,
 state1, state2);
 input clk, rst, enable;
 input [2:0] data_in;
 output data_out, state0, state1, state2;
 /* Defined state labels; this style preferred over 'defines'*/
 parameter deflt=2'bxx;
 parameter idle=2'b00;
 parameter read=2'b01;
 parameter write=2'b10;
 reg data_out, state0, state1, state2;
 reg [1:0] state/* synthesis syn_probe = 1 */;
 reg [1:0]next_state;

 /* always block with sequential logic*/
 always @(posedge clk or negedge rst)
 if (!rst) state <= idle;
 else state <= next_state;

 /* always block with combinational logic*/
 always @(state or enable or data_in) begin
 /* Default values for FSM outputs*/
 state0 <= 1'b0;
 state1 <= 1'b0;
 state2 <= 1'b0;
 data_out <= 1'b0;
 case (state)
 idle : if (enable) begin
 state0 <= 1'b1;
 data_out <= data_in[0];
 next_state <= read;
 end
 else begin
 next_state <= idle;
 end
 read : if (enable) begin
 state1 <= 1'b1;
 data_out <= data_in[1];
 next_state <= write;
 end
 else begin
 next_state <= read;
 end
 write : if (enable) begin
 state2 <= 1'b1;
 data_out <= data_in[2];
 next_state <= idle;
 end
 end
 end
```

```
 end
 else begin
 next_state <= write;
 end
/* Default assignment for simulation */
default : next_state <= deflt;
endcase
end
endmodule
```

## VHDL Test Case

In the following test case, the state register state has the `syn_probe` attribute applied on it which prevents the compiler from extracting the state machine.

```
library ieee;
use ieee.std_logic_1164.all;

entity stmchl is
 port (clk, in1, rst: in std_logic;
 out1: out std_logic);
end stmchl;

architecture behave of stmchl is
type state_values is (sx, s0, s1);
signal state, next_state: state_values;
attribute syn_probe : boolean;
attribute syn_probe of state : signal is true;
begin
 process (clk, rst)
 begin
 if rst = '1' then
 state <= s0;
 elsif rising_edge(clk) then
 state <= next_state;
 end if;
 end process;

 process (state, in1)
 begin
 -- set defaults for output and state
 out1 <= '0';
 next_state <= sx; -- catch missing assignments to next_state
 case state is
 when s0 =>
 if in1 = '0' then
```

```
 out1 <='1';
 next_state <= s1;
 else
 out1 <= '0';
 next_state <= s0;
 end if;
when s1 =>
 if in1 = '0' then
 out1 <='0';
 next_state <= s0;
 else
 out1 <= '1';
 next_state <= s1;
 end if;
when sx =>
 next_state <= sx;
end case;
end process;
end behave;
```

## Action

If probes are required, FSM extraction will not occur and the FSM compiler can be turned off. If FSM extraction is required, probes on the state register must be avoided. The state transition can be observed using the state diagrams in the FSM viewer. The viewer can isolate states and check for transitions into and out of the states.

# CL201

### **@N: Trying to extract state machine for register <*next\_state*>.**

The compiler encountered a state machine either through the HDL code description (FSM compiler enabled) or when a `syn_state_machine` attribute is explicitly included the HDL code. The following code describes the state machine for the register `next_state` which depends on `present_state` as well as the inputs.

```
module statmch1(launch_shuttle, land_shuttle, start_countdown,
 start_trip_meter, clk, all_systems_go,
 just_launched, is_landed, cnt, abort_mission);
output launch_shuttle, land_shuttle, start_countdown,
 start_trip_meter;
input clk, just_launched, is_landed, abort_mission,
 all_systems_go;
input [3:0] cnt;
reg launch_shuttle, land_shuttle, start_countdown,
 start_trip_meter;

// parameters for the one-hot encoding of the states
parameter HOLD = 5'h1, SEQUENCE = 5'h2, LAUNCH = 5'h4;
parameter ON_MISSION = 5'h8, LAND = 5'h10;
reg [4:0] present_state, next_state;

always @ (negedge clk or posedge abort_mission)
begin
/* set the outputs to some default values (so you don't have to set
them in every case below) */
{launch_shuttle,land_shuttle,start_trip_meter,start_countdown}=4'b
0;

/* check the value of the asynchronous reset */
if (abort_mission)
 next_state = LAND;
else begin

/* set the next_state to be the present_state by default */
next_state = present_state;

/* depending on the present_state and inputs, the case items set
the next_state variable and output values */
case (present_state)
HOLD: if (all_systems_go) begin
 next_state = SEQUENCE;
 start_countdown = 1;
end
SEQUENCE: if (cnt == 0) next_state = LAUNCH;
LAUNCH: begin
 next_state = ON_MISSION;
 launch_shuttle = 1;
end
ON_MISSION:
```

```

// Stay on mission until abort mission
if (just_launched) start_trip_meter = 1;
LAND: if (is_landed)
 next_state = HOLD;
else land_shuttle = 1;

/* set the default case to 'bx (don't care) or some known state, so
it matches simulation before you do a reset */
 default: next_state = 'bx;
endcase
end // end of if-else

/* make sure you set the present_state variable to the next_state
that you just assigned, so it has the correct value at the next
active clock edge */
present_state = next_state;
end // end of always
endmodule

```

The compiler gives an initial encoding to the state machine depending on the number of states present in the state machine (0-4 is sequential, 5-24 is one-hot, and 25 and above is gray). The default encoding can be overridden with the `syn_encoding` directive.

The one-to-one correspondence of the RTL encoding to the mapped encoding can be viewed in the log file as follows:

```

Encoding state machine work.statmch1(verilog)-next_state[4:0]
original code -> new code
 00001 -> 00001
 00010 -> 00010
 00100 -> 00100
 01000 -> 01000
 10000 -> 10000

```

The FSM Viewer provides a detailed view of the state machine represented as a bubble diagram with detailed descriptions of the transitions.

## Action

If this note is not followed by a description as indicated below

```

Extracted state machine for register next_state
State machine has 5 reachable states with original encodings of:

```

```
00001
00010
00100
01000
10000
```

then:

- Make sure that the FSM compiler is on
- Use `syn_state_machine` directive on the state register to force the state machine to be extracted.

## CL204

### **@W: Illegal assignment to <directiveName>, a 1/0 are legal.**

A value other than 0 or 1 is assigned to a directive that only accepts a binary value. In the test case below, a `syn_keep` directive is assigned an illegal value of 11 (only 0 and 1 are allowed) which results in the above warning.

```
module top(out1, out2, clk, in1, in2);
 output out1, out2;
 input clk;
 input in1, in2;
 wire and_out;
 wire keep1 /* synthesis syn_keep=11 */;
 wire keep2 /* synthesis syn_keep=1 */;
 reg out1, out2;
 assign and_out=in1&in2;
 assign keep1=and_out;
 assign keep2=and_out;

 always @(posedge clk)
 begin;
 out1<=keep1;
 out2<=keep2;
 end
endmodule
```

## Action

To avoid the above warning, make sure that any directive requiring a binary value is assigned either 0 or 1. In the corrected test case below, the syn\_keep directive is reassigned a legal value of 1.

```
module top(out1, out2, clk, in1, in2);
 output out1, out2;
 input clk;
 input in1, in2;
 wire and_out;
 wire keep1 /* synthesis syn_keep=1 */;
 wire keep2 /* synthesis syn_keep=1 */;
 reg out1, out2;
 assign and_out=in1&in2;
 assign keep1=and_out;
 assign keep2=and_out;

 always @(posedge clk)
 begin;
 out1<=keep1;
 out2<=keep2;
 end
endmodule
```

## CL207

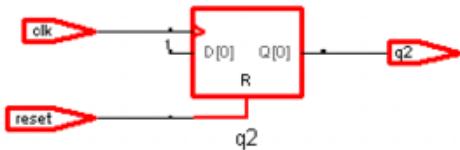
### **@W: All reachable assignments to <q1> assign <0>, register removed by optimization.**

A register data type takes the same value under all conditions. If a constant is registered, the compiler may optimize it away. In the first test case below, because q1 is always zero, a flip-flop is not required (the register is optimized away) as indicated by the warning.

```
module test (input clk, reset, output reg q1);
 always @ (posedge clk or posedge reset)
 if (reset)
 q1 <= 1'b0;
 else
 q1 <= 1'b0;
endmodule
```

In the next test case, q2 has different values and a flip-flop with a constant 1 at its data input is generated.

```
module test (input clk, reset, output reg q2);
 always @ (posedge clk or posedge reset)
 if(reset)
 q2 <= 1'b0;
 else
 q2 <= 1'b1;
endmodule
```



If reset was not present, the flip-flop would be optimized away with the same warning message. Similarly, if q2 was reset to 1'b1, the flip-flop would be optimized away with the above warning.

## Action

To preserve a register from optimization, use the `syn_preserve` attribute.

# CL208

## **@W: All reachable assignments to bit <0> of <q1[7:0]> assign <1>, register removed by optimization**

A bit of a vector of register data type takes the same value under all conditions. If a constant is registered, the compiler may optimize it away. In the following test case, because bits 0, 1, 3, 4, 5, 6 take the same value irrespective of reset, their corresponding registers are optimized away, and this warning is issued for each of the six bits.

```

module test (input clk, reset, output reg [7:0] q1);
always @ (posedge clk or posedge reset)
 if (reset)
 q1 <= 8'b10001111;
 else
 q1 <= 8'b0001011;
endmodule

```

## Action

To preserve a register from optimization, use the `syn_preserve` attribute.

# CL214

**@N: Found multi-write port RAM <mem>, number of write ports=<2>, depth=<256>, width=<8>**

The compiler encountered a RAM with more than one write port. The number of write ports and the depth and width of the RAM are also reported.

## Verilog Test Case

The RAM in the following Verilog test case has two write ports, an 8-bit address width (depth of 256), and an 8-bit data width.

```

module ram (data0,data1,
 waddr0,waddr1,
 we0,we1,
 clk0,clk1,q0,q1);
parameter d_width = 8;
parameter addr_width = 8;
parameter mem_depth = 256; input [d_width-1:0] data0, data1;
input [addr_width-1:0] waddr0, waddr1;
input we0, we1, clk0, clk1;
output [d_width-1:0] q0, q1;
reg [addr_width-1:0] reg_addr0, reg_addr1;
reg [d_width-1:0] mem [mem_depth-1:0]
 /* synthesis syn_ramstyle="no_rw_check" */;
assign q0 = mem[reg_addr0];
assign q1 = mem[reg_addr1];

```

```

always @(posedge clk0)
begin
 reg_addr0 <= waddr0;
 if (we0)
 mem[waddr0] <= data0;
end

always @(posedge clk1)
begin
 reg_addr1 <= waddr1;
 if (we1)
 mem[waddr1] <= data1;
end
endmodule

```

## VHDL Test Case

The RAM in the following VHDL test case has two write ports, an 8-bit address width (depth of 256), and an 8-bit data width.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity ramtest is
 port (q0,q1: out std_logic_vector (7 downto 0);
 data0,data1: in std_logic_vector (7 downto 0);
 waddr0,waddr1: in std_logic_vector (3 downto 0);
 we1,we0: in std_logic;
 clk0,clk1: in std_logic);
end ramtest ;

architecture rtl of ramtest is
type mem_type is array (15 downto 0) of
 std_logic_vector (7 downto 0);
signal reg_addr1,reg_addr0: std_logic_vector (3 downto 0);
signal mem : mem_type;
attribute syn_ramstyle: string;
attribute syn_ramstyle of mem : signal is "no_rw_check";

begin
q0 <= mem(conv_integer(reg_addr0));
q1 <= mem(conv_integer(reg_addr1));
process (clk0, we0, waddr0)
begin
 if rising_edge (clk0) then
 if we0 = '1' then

```

```

 mem (conv_integer (waddr0)) <=data0;
 end if;
 reg_addr0<= waddr0;
 end if;
end process;

process (clk1, we1, waddr1)
begin
 if rising_edge (clk1) then
 if we1 = '1' then
 mem (conv_integer (waddr1)) <= data1;
 end if;
 reg_addr1 <= waddr1;
 end if;
end process;

end rtl;

```

## Action

Make sure that all of the intended RAMs are reported by the compiler so that the correct resources are used during mapping to implement the RAM structure. For some technologies, RAMs are automatically implemented in Block RAM. Specific implementation of a RAM as Select RAM, registers, or Block RAM can be forced by attaching a `syn_ramstyle` attribute on the RAM register.

# CL216

## **@W: always\_ff does not infer sequential logic**

The `always_ff` process in SystemVerilog models sequential logic that is triggered by clocks. If the behavior within the `always_ff` block does not represent sequential logic as in the test case below, the compiler issues the above warning.

```

module Test01 (a,b,clk,out1);
input a,b,clk;
output out1;
reg out1;

```

```
always_ff@ (clk,a,b)
begin
 if (clk)
 out1=a & b;
 else
 out1= a | b;
end
endmodule
```

## Action

Make sure that the logic within an `always_ff` process always infers sequential logic triggered by clocks. In the corrected test case below, the signal in the sensitivity list is edge-triggered and acts as clock.

```
module Test01 (a,b,clk,out1);
input a,b,clk;
output out1;
reg out1;

always_ff@ (posedge clk)
begin
 if (clk)
 out1=a & b;
 else
 out1= a | b;
end
endmodule
```

# CL217

## **@W: always\_comb does not infer combinatorial logic**

The `always_comb` process in SystemVerilog models combinational logic. If the behavior within the `always_comb` block does not represent combinational logic as in the test case below, the compiler issues the above warning.

```
module test01 (a, b,out1);
input a,b;
output out1;
reg out1;
```

```
always_comb
begin
 if (a)
 out1 = a & b;
 end
endmodule
```

## Action

Make sure that the logic within an `always_comb` process always infers a combinational block. In the corrected test case below, signal `a` (which acted as a condition) is removed so that the behavior of the logic is purely combinational.

```
module test01 (a, b,out1);
 input a,b;
 output out1;
 reg out1;

 always_comb
 begin
 out1 = a & b;
 end
endmodule
```

# CL218

## @W: `always_latch` does not infer latch logic

The `always_latch` process in SystemVerilog models latched logic. If the behavior within the `always_latch` block does not infer a latch as in the test case below, the compiler issues the above warning.

```
module adder8 (sum,c0,a,b,cin);
 input [7:0]a,b;
 output reg [7:0]sum;
 output reg [0:0] c0;
 input [0:0] cin;

 always_latch
 {c0,sum} = a + b + cin;
endmodule
```

## Action

Make sure that the logic within an `always_latch` process always infers a latch. In the corrected test case below, signal `a` is written as an enable signal for the latch.

```
module adder8 (sum,c0,a,b,cin);
 input [7:0]a,b;
 output reg [7:0]sum;
 output reg [0:0] c0;
 input [0:0] cin;

 always_latch
 if(a)
 {c0,sum} = a + b + cin;
endmodule
```

# CL219

### **@E: Multiple non-tristate drivers for net <out1> in <seq>**

The above error occurs when a net with multiple drivers includes a driver that is not a tri-state.

## VHDL Test Case

In the following test case, output `out1` has two drivers, inside the process and outside the process. Because the driver outside the process is not tri-stated, the compiler errors out.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
 port (a: in std_logic;
 b: in std_logic;
 out1 : out std_logic);
end;
```

```
architecture rtl of seq is
signal sig:std_logic;
begin
sig <=a;
out1 <=a when b='1' else '1';
process(a,b)
begin
if (a='1') then
out1 <=b;
else
out1 <='Z';
end if;
end process;
end;
```

## Action

If out1 has multiple drivers, all drivers must be tri-stated. To eliminate the error in the above test case, the code must have a tri-state implementation for out1 as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;

entity seq is
port (a: in std_logic;
 b: in std_logic;
 out1 : out std_logic);
end;

architecture rtl of seq is
signal sig:std_logic;
begin
sig <=a;
out1 <=a when b='1' else 'Z';
process(a,b)
begin
if (a='1') then
out1 <=b;
else
out1 <='Z';
end if;
end process;
end;
```

## Verilog Test Case

In the following test case, output out1 has multiple drivers. Because the b driver is not tri-stated, the compiler errors out.

```
module seq (input a, b, sel1, sel2, output out1);
 assign out1=sel1? a :1'bz;
 assign out1=sel2? b :1'b0;
endmodule
```

## Action

For nets with multiple drivers, all drivers must be tri-stated. To eliminate the error, add a tri-state implementation for out1 as shown in the corrected test case below.

```
module seq (input a, b, sel1, sel2, output out1);
 assign out1=sel1? a :1'bz;
 assign out1=sel2? b :1'bz;
endmodule
```

# CL224

## **@E: ROM clash detected: address <0000000> with multiple data entries <1011>, <1010> found**

A parallel\_case synthesis directive was specified within a ROM description and one of the addresses had multiple data entries. Because of the parallel\_case directive, both case branches have equal priority, and the compiler is unable to resolve the multiple entry. In the test case below, address 0000000 has two entries (1011 and 1010) which results in the error.

```
module test_rom_style(z, a,clk);
 input clk;
 output reg [3:0] z ;
 input [6:0] a;
```

```

always @(posedge clk)
begin
 case (a) /* synthesis parallel_case */
 7'b00000000: z = 4'b1011;
 7'b00000000: z = 4'b1010;
 7'b00000001: z = 4'b0001;
 7'b00000010: z = 4'b0011;
 7'b00000011: z = 4'b0010;
 7'b00000100: z = 4'b1110;
 7'b00010001: z = 4'b0111;
 7'b00010100: z = 4'b0101;
 7'b00011001: z = 4'b0100;
 7'b00100000: z = 4'b1100;
 7'b00100001: z = 4'b1101;
 7'b00100010: z = 4'b1111;
 7'b00100011: z = 4'b1110;
 7'b00110000: z = 4'b1010;
 7'b00110010: z = 4'b1011;
 7'b00111110: z = 4'b1001;
 7'b11111111: z = 4'b1000;
 7'b10010100: z = 4'b1111;
 7'b11010111: z = 4'b1101;
 7'b11000011: z = 4'b1100;
 default: z = 4'b0000;
 endcase
end
endmodule

```

## Action

Make sure that each address has only a data entry when specifying a ROM in conjunction with a `parallel_case` synthesis directive. Note that if the directive is omitted, the error does not occur even when an address has multiple entries.

```

module test_rom_style(z, a,clk);
 input clk;
 output reg [3:0] z ;
 input [6:0] a;

 always @(posedge clk)
 begin
 case (a) /* synthesis parallel_case */
 7'b00000000: z = 4'b1011;
 7'b00000001: z = 4'b0001;
 7'b00000010: z = 4'b0011;
 7'b00000011: z = 4'b0010;

```

```
7'b0000100: z = 4'b1110;
7'b0001001: z = 4'b0111;
7'b0001010: z = 4'b0101;
7'b0001101: z = 4'b0100;
7'b0010000: z = 4'b1100;
7'b0010001: z = 4'b1101;
7'b0010010: z = 4'b1111;
7'b0010011: z = 4'b1110;
7'b0011000: z = 4'b1010;
7'b0011010: z = 4'b1011;
7'b0011110: z = 4'b1001;
7'b1111111: z = 4'b1000;
7'b1001010: z = 4'b1111;
7'b1101011: z = 4'b1101;
7'b1100011: z = 4'b1100;
default: z = 4'b0000;
endcase
end
endmodule
```

## CL226

### **@N: Turning <off> resource sharing in module <cl226> (attribute syn\_sharing has been specified)**

A syn\_sharing directive on an individual module or entity is set to a value opposite from the global resource sharing option set in the project view.

#### Verilog Test Case

For the test case below, the global resource sharing option is enabled in the project view. When the compiler encounters the syn\_sharing directive with the value FALSE in the module, this note is displayed to indicate that the global setting is being overridden.

```
module cl226_v(a, b, c, d, e, f, sel, data_out)
 /* synthesis syn_sharing =0 */;
 input a,b,c,d,e,f;
 input [1:0]sel;
 output reg data_out;
```

```
always @(a,b,c,d,e,f,sel)
 if(sel==2'b01)
 data_out<= a + b;
 else if(sel==2'b10)
 data_out <= c + d;
 else if(sel==2'b11)
 data_out <= e + f;
 else
 data_out <= a + e;
endmodule
```

## Action

This note informs you of resource sharing conflicts. Use the `syn_sharing` directive to set the appropriate values.

## VHDL Test Case

For the test case below, the global resource sharing option is enabled in the project view. When the compiler encounters the `syn_sharing` directive with the value FALSE in the entity, this note is displayed to indicate that the global setting is being overridden.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity cl226 is
 port (a,b,c,d,e,f: in std_logic_vector(0 downto 0);
 sel : in std_logic_vector(1 downto 0);
 data_out: out std_logic_vector(0 downto 0));
end cl226;

architecture rtl of cl226 is
attribute syn_sharing : boolean;
attribute syn_sharing of rtl: architecture is FALSE;
begin
 process (sel,a,b,c,d,e,f)
 begin
 if (sel = "01") then
 data_out <= a + b;
 elsif (sel = "10") then
 data_out <= c + d;
 elsif (sel = "11") then
 data_out <= e + f;
```

```
 else
 data_out <= a + e;
 end if;
 end process;
end rtl;
```

## Action

This note informs you of resource sharing conflicts. Use the syn\_sharing directive to set the appropriate values.

# CL230

## **@N: Contact Synopsys Customer Support**

This is an informative note. If you want to resolve a problem, contact Synopsys by opening a case through the Synopsys website.

When you contact Synopsys, include the error code and message text in all correspondence. If possible, please include a copy of your project file and the source file or files generating the error or a test case along with a description of the problem.

# CL240

## **@W: <Out2> is not assigned a value (floating)**

An output port is undriven in the current design. In the following test case, out2 is an undriven output port.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
```

```

entity test is
 port (a, b: in std_logic_vector (7 downto 0);
 clk: in std_logic;
 out1, out2: out std_logic_vector (7 downto 0));
end entity;

architecture only of test is
begin
 process (clk)
 begin
 if clk'event and clk = '1' then
 out1 <= a and b;
 end if;
 end process;
end only;

```

## Action

Make sure this is the intended behavior. The compiler represents the undriven bits as either a constant '0' or '1' which causes an RTL post-synthesis simulation mismatch. If out2 is driven by logic, the code must be edited to represent this. To eliminate the warning, edit the code to drive the out2 port.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity test is
 port (a, b: in std_logic_vector (7 downto 0);
 clk: in std_logic;
 out1, out2: out std_logic_vector (7 downto 0));
end entity;

architecture only of test is
begin
 process (clk)
 begin
 if clk'event and clk = '1' then
 out1 <= a and b;
 out2 <= a or b;
 end if;
 end process;
end only;

```

## CL245

**@W: Bit %d of input %n of instance %n is undriven. If this is not the intended behavior, drive the inputs with valid values, or inputs from the top level.**

An undriven bit or bit slices of bus signals connect to the input of an instance. In the following test case, the internal signal temp (connecting the input b of the instance U1) has no driven bits.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity test is
 port (a, b: in std_logic_vector (7 downto 0);
 clk, set: in std_logic;
 out1: out std_logic_vector (7 downto 0));
end entity;

architecture only of test is
begin
 process (clk, set)
 begin
 if (set = '1') then
 out1 <= (others => '1');
 elsif clk'event and clk = '1' then
 out1 <= a and b;
 end if;
 end process;
end only;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity test_top is
 port (a1, b1: in std_logic_vector (7 downto 0);
 clk1, set1: in std_logic;
 out1: out std_logic_vector (7 downto 0));
end entity;
```

```

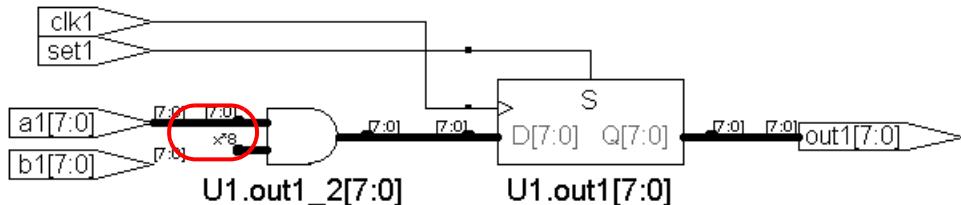
architecture only of test_top is
component test is
 port (a, b: in std_logic_vector (7 downto 0);
 clk, set: in std_logic;
 out1: out std_logic_vector (7 downto 0));
end component;

signal temp:std_logic_vector (7 downto 0);
begin
U1: test port map
(a => a1,
 b => temp,
 clk => clk1,
 set => set1,
 out1 => out1
);
end only;

```

## Action

In the above test case, because `temp` is unconnected, its value is a ‘don’t care’ which results in the AND gate equating to ‘0’ in the RTL view below, and the mapped netlist connecting the flip-flop to a constant during synthesis.



Make sure that this is the intended behavior. To avoid this warning, always drive the inputs with valid values or inputs from the top level. To eliminate the warning in the above test case and to include the top-level input `b1` instead of an unassigned signal, edit the instantiation as shown in the code example below.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

```

```
entity test is
 port (a, b: in std_logic_vector (7 downto 0);
 clk, set: in std_logic;
 out1: out std_logic_vector (7 downto 0));
end entity;

architecture only of test is
begin
 process (clk, set)
 begin
 if (set = '1') then
 out1 <= (others => '1');
 elsif clk'event and clk = '1' then
 out1 <= a and b;
 end if;
 end process;
end only;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity test_top is
 port (a1, b1: in std_logic_vector (7 downto 0);
 clk1, set1: in std_logic;
 out1: out std_logic_vector (7 downto 0));
end entity;

architecture only of test_top is
component test
 port (a, b: in std_logic_vector (7 downto 0);
 clk, set: in std_logic;
 out1: out std_logic_vector (7 downto 0));
end component;

signal temp:std_logic_vector (7 downto 0);
begin
U1: test port map
 (a => a1,
 b => b1,
 clk => clk1,
 set => set1,
 out1 => out1
);
end only;
```

# CL246

**@W: Input port bits %d to %d of %B are unused. Assign logic for all port bits or change the input port size.**

Not all of the bits of a vectored input port are used.

## Verilog Test Case

In the test case below, input port sel is declared as a 16-bit vector (3 downto 0). However, within the case block (with full\_case directive set which implies all cases are given), only the first four cases (binary equivalents 0, 1, 2, and 3) are specified which requires only two bits of sel. As a result, the compiler issues the warning to indicate that the remaining two bits are unused.

```
module cl234_v (input a,b,c,d, input [3:0]sel,output reg q);
always @ (a,b,c,d,sel)
 case (sel) /* synthesis full_case */
 4'b0000: q<=a;
 4'b0001: q<=b;
 4'b0010: q<=c;
 4'b0011: q<=d;
 endcase
endmodule
```

## Action

Assign logic for all possibilities of a vectored input or change the input port to the required size to include all required cases. In the test case below, by including an others clause in the case statement and removing the full\_case directive, some default assignment covering all the remaining possibilities is provided and all bits of the input port are now used.

```
module cl234_v (input a,b,c,d, input [3:0]sel,output reg q);
always @ (a,b,c,d,sel)
 case (sel)
 4'b0000: q<=a;
 4'b0001: q<=b;
 4'b0010: q<=c;
 4'b0011: q<=d;
 default: q<= 1'bZ;
 endcase
endmodule
```

## VHDL Test Case

In the test case below, input port sel is declared as a 16-bit vector (3 downto 0). However, within the case block, only the first four cases (binary equivalents 0, 1, 2, and 3) are specified which requires only two bits of sel. As a result, the compiler issues this warning to indicate that the remaining two bits are unused.

```

library ieee;
use ieee.std_logic_1164.all;

entity check is
 port (a,b,c,d: in std_logic;
 sel: in std_logic_vector(3 downto 0);
 q:out std_logic);
end check;

architecture test of check is
begin
 process(sel,a,b,c,d)
 begin
 case(sel) is
 when "0000"=>q<=a;
 when "0001"=>q<=b;
 when "0010"=>q<=c;
 when "0011"=>q<=d;
 end case;
 end process;
end test;
```

## Action

Assign logic for all possibilities of a vectored input or change the input port to the required size to include all required cases. In the test case below, by including an others clause in the case statement, some default assignment covering all the remaining possibilities is provided and all bits of the input port are now used.

```

library ieee;
use ieee.std_logic_1164.all;

entity check is
 port (a,b,c,d: in std_logic;
 sel: in std_logic_vector(3 downto 0);
 q:out std_logic);
end check;
```

```

architecture test of check is
begin
 process(sel,a,b,c,d)
 begin
 case(sel) is
 when "0000"=>q<=a;
 when "0001"=>q<=b;
 when "0010"=>q<=c;
 when "0011"=>q<=d;
 when others=>q<='Z';
 end case;
 end process;
end test;

```

## CL247

### **@W: Input port bit <7> of <din[7:0]> is unused**

A single input port bit is not used within the design. In the following test case, bit 7 of input port din is not used as indicated in the text of the message.

```

module CL247 (
 input [7:0] din,
 output [6:0] dout);
 assign dout = din;
endmodule

```

### Action

If the design is intended to read all of the bits of the input port, modify the design. In the above test case, the input port is eight bits wide, and the output port is only seven bits wide. In the corrected test case below, the output port width is set to eight bits to match the input port width.

```

module CL247 (
 input [7:0] din,
 output [7:0] dout);
 assign dout = din;
endmodule

```

# CL249

## @W: Initial value is not supported on state machine <state>

A state machine is being extracted for a register that has an initial value (initial values are not supported for state machines). The initial value is ignored and not propagated to the netlist which can potentially result in a simulation mismatch during initial start-up (that is, before any reset is applied during simulation).

Two cases of the warning are given. In the first case, which uses the traditional Verilog method of coding a state machine, the warning is generated when register state is explicitly initialized. In the second case, which uses the SystemVerilog ENUM style state machine; the warning is generated when register state is assigned an initial value through implicit initialization.

### Case 1

```
module CL249 (clk, rst, data_in, data_out);
 input clk, rst;
 input [2:0] data_in;
 output logic data_out;
 localparam bit [1:0] deflt = 0;
 localparam bit [1:0] idle = 1;
 localparam bit [1:0] read = 2;
 localparam bit [1:0] write = 3;
 logic [1:0] state = deflt;
 logic [1:0] next_state;

 always @(posedge clk or negedge rst)
 if (!rst) state <= idle;
 else state <= next_state;

 always_comb begin
 case (state)
 idle : begin
 data_out <= data_in[0];
 next_state <= read;
 end
 read : begin
 data_out <= data_in[1];
 next_state <= write;
 end
 write : begin
 data_out <= data_in[2];
 next_state <= idle;
 end
 endcase
 end
endmodule
```

```
 data_out <= data_in[2];
 next_state <= idle;
 end
 default : begin
 data_out <= 1'b0;
 next_state <= deflt;
 end
endcase
end
endmodule
```

## Case 2

```
module CL249 (clk, rst, data_in, data_out);
input clk, rst;
input [2:0] data_in;
output logic data_out;
typedef enum bit [1:0] {deflt, idle, read, write} MyStateType;
MyStateType state;
MyStateType next_state;

always @(posedge clk or negedge rst)
 if (!rst) state <= idle;
 else state <= next_state;

always_comb begin
 case (state)
 idle : begin
 data_out <= data_in[0];
 next_state <= read;
 end
 read : begin
 data_out <= data_in[1];
 next_state <= write;
 end
 write : begin
 data_out <= data_in[2];
 next_state <= idle;
 end
 default : begin
 data_out <= 1'b0;
 next_state <= deflt;
 end
 endcase
end
endmodule
```

## Action

If the initial value for the corresponding register is a critical design requirement, FSM extraction must be disabled for the register either by turning off the FSM Compiler (which globally disables FSM Extraction for the entire design) or by explicitly disabling FSM extraction for the register with the `syn_state_machine` directive.

```
MyStateType state /*synthesis syn_state_machine = 0*/;
```

## CL252

### **@W: Bit <0> of signal <temp> is floating -- simulation mismatch possible.**

Bit or bit slices of bus signals are not being driven in the design. In the following test case, the internal signal `temp` is driving the output `out1`, but its bits are not being driven.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity test is
 port (clk, set: in std_logic;
 out1: out std_logic_vector(7 downto 0));
end entity;

architecture only of test is
signal temp:std_logic_vector (7 downto 0);
begin
 process (clk, set)
 begin
 if (set = '1') then
 out1 <= (others => '1');
 elsif clk'event and clk = '1' then
 out1 <= temp;
 end if;
 end process;
end only;
```

## Action

Make sure this is the intended behavior. Otherwise, make sure that the outputs are driven by logic. To eliminate the warning, redefine the process as shown in the corrected test case below.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity test is
 port (clk, set: in std_logic;
 out1: out std_logic_vector(7 downto 0));
end entity;

architecture only of test is
signal temp:std_logic_vector (7 downto 0);
begin
 process (clk, set)
 begin
 if (set = '1') then
 temp <= (others => '1');
 elsif clk'event and clk = '1' then
 temp <= temp + 1;
 end if;
 out1 <= temp;
 end process;
end only;
```

## CL260

**@W: Pruning register bit %d of %B. If this is not the intended behavior, drive the input with valid values, or an input from the top level.**

One of the bits of a register port or variable is not driven by any logic or driven by a constant value. In the test case below, dout[4] is always driven with a value of 0 and, accordingly, register bit 4 of dout is pruned as reported in the message.

```
module top (input clk, rst, input [4:0] din,
 output reg [4:0] dout);
```

```

always @(posedge clk) begin
 if(rst)
 dout <= 0;
 else
 dout <= din[3:0];
end
endmodule

```

## Action

Make sure this is the intended behavior. Otherwise, make sure that all outputs are driven by logic.

```

module top (input clk, rst, input [4:0] din,
 output reg [4:0] dout);

 always @(posedge clk) begin
 if(rst)
 dout <= 0;
 else
 dout <= din; // instead of din[3:0]
 end
endmodule

```

# CL264

### **@E: Range bounds are not constants (variable <j>).**

A variable partial select operator (+/-) is not used following a base expression while indexing any non-scalar variable using variable partial select. In the test case below, the range of vector data1 is specified using expression (j\*8) but, because the variable partial select operator is missing, the compiler errors out.

```

module data_buffer (input clk, input [7:0]data1,
 output reg [7:0]bytel);
 integer j=0;
 always@(posedge clk)
 bytel= data1[(j*8):8];
endmodule

```

## Action

When indexing a non-scalar variable, make sure that a variable partial select operator (+/-) is used following the base expression as shown in the corrected test case below.

```
module data_buffer (input clk, input [7:0]data1,
 output reg [7:0]bytel);
 integer j=0;
 always@(posedge clk)
 bytel= data1[(j*8)+:8];
endmodule
```

# CL265

### **@W: Pruning bit <3> of <temp\_3[3:0]> - not in use ...**

An intermediate register includes unused bits that the compiler has pruned away. In the example below, intermediate 4-bit register temp is used to register the inputs of three bits; the unused fourth bit is pruned by the compiler as indicated by the warning.

```
module top (d,clk,q);
 input clk;
 input [2:0]d;
 output [2:0] q;
 reg [3:0]temp;

 always @(posedge clk)
 begin
 temp <= d;
 end
 assign q = temp;
endmodule
```

## Action

Make sure that all intermediate registers are of the required size. To correct the test case, reduce the size of intermediate register temp from four bits to three bits.

```
module top (d,clk,q);
 input clk;
 input [2:0]d;
 output [2:0] q;
 reg [2:0]temp;

 always @(posedge clk)
 begin
 temp <= d;
 end
 assign q = temp;
endmodule
```

## CL269

**@W: State error detection not built. Check if state machine is inferred or if case statement is missing a clause.**

In safe mode (Preserve and Decode Unreachable States is enabled on the High Reliability tab of the Implementation Options panel), the compiler generates a state error detection component for all case statements used to synthesize the state machine logic. If the component is removed, then this warning is generated for you to confirm the following:

1. A state machine was not inferred for a particular case statement.
2. A state machine was inferred from a case statement, but the case statement is missing an others clause (VHDL) or default clause (Verilog).

### Action

Check for the correct intended behavior. In the first condition above, verify whether the logic should not be a state machine or if the compiler was unable to extract it. For the second scenario, you may need to add an others or default clause to the source code so the compiler knows how to handle a bad state.

## CL271

**@W: Pruning unused bits <7> to <6> of <dout[7:0]>. If this is not the intended behavior, drive the inputs with valid values, or inputs from the top level.**

A range of register ports or variables is not driven by any logic or driven by a constant value. In the test case below, unnecessary registers are being assigned to bits 7 and 6 of dout. The unused registers are being removed as reported in the message.

```
module top (input clk, rst, input [5:0] din, output reg[7:0] dout);

 always @(posedge clk) begin
 if(rst)
 dout <= 8'b0;
 else
 dout[5:0] <= din;
 end
 assign dout[7:6] = 2'b11;
endmodule
```

### Action

Avoid defining any unnecessary outputs.

```
module top (input clk, rst, input [5:0] din, output reg[7:0] dout);

 always @(posedge clk) begin
 if(rst)
 dout[5:0] <= 8'b0;
 else
 dout[5:0] <= din;
 end
 assign dout[7:6] = 2'b11;
endmodule
```

## CL279

**@W: Pruning register bits %d to %d of %B. If this is not the intended behavior, drive the inputs with valid values, or inputs from the top level.**

Bits of a register port or variable are not driven by any logic or driven by a constant value. In the test case below, only six bits are defined in the else clause and the remaining two bits are driven by constant values. The unnecessary constant registers are being removed as reported in the message.

```
module top (input clk,rst, input [5:0] din, output reg[7:0] dout);
 always @ (posedge clk) begin
 if(rst)
 dout <= 8'b0;
 else
 dout[5:0] <= din[5:0];
 end
endmodule
```

### Action

Make sure this is the intended behavior. Otherwise, make sure that all outputs are driven by logic.

```
module top (input clk, rst, input [5:0] din, output reg[7:0] dout);
 always @ (posedge clk) begin
 if(rst)
 dout <= 8'b0;
 else
 dout[5:0] <= din[5:0];
 dout[7:6] <= din[2:1];
 end
endmodule
```

## CL282

**@A: Feedback mux created for signal <d\_out[3:0> -- possible set/reset assignment for signal missing. Specifying a reset value will improve timing and area."**

The reset block of a clocked process does not have an assignment for any of the registered signals and indicates that a feedback mux will be created to retain the register value on the reset condition. The following test case results in the warning.

```
library ieee;
use ieee.std_logic_1164.all;

entity warn is
 port (clk: in std_logic;
 rst: in std_logic;
 a: in std_logic_vector(3 downto 0);
 d_out: out std_logic_vector(3 downto 0));
end warn;

architecture beh of warn is
begin
 process(clk,rst)
 begin
 if rst = '1' then
 elsif rising_edge(clk) then
 d_out <= a;
 end if;
 end process;
end beh;
```

### Action

Assigning a value explicitly under all conditions of the if-else statement avoids any unintentional feedback and the corresponding warnings. To eliminate the warning from the above test case, Add `d_out <= "0000"` after the `rst` statement in the architecture as shown below.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity warn is
 port (clk: in std_logic;
 rst: in std_logic;
 a: in std_logic_vector(3 downto 0);
 d_out: out std_logic_vector(3 downto 0));
end warn;

architecture beh of warn is
begin
 process(clk,rst)
 begin
 if rst = '1' then
 d_out <= "0000";
 elsif rising_edge(clk) then
 d_out <= a;
 end if;
 end process;
end beh;
```

## CL291

**@A: Register  $<q>$  with asynchronous load is being synthesized in compatibility mode. A synthesis/simulation mismatch is possible.**

When the compiler encounters an asynchronous load, it uses the set and reset inputs of the flip-flop to implement the load. In the following test case, the RTL describing the flip-flop is non-standard in that it does not define the set/reset inputs. For this case, a flip-flop with an asynchronous set and reset is inferred and the hardware is built with an internal set/reset implementation. This inferencing creates the possible synthesis/simulation mismatch between the RTL and the post-synthesis netlist as indicated in the advisory message.

### Case 1

```
module asyncload (d, d0, load, clk, q);
 input d, d0, load, clk;
 output q;
 reg q;
```

```
always @(posedge clk or posedge load)
begin
 if (load)
 q = d0;
 else
 q = d;
end
endmodule
```

The following modified test case illustrates how to group the logic outside the always block using wire statements to define the asynchronous set/reset inputs to avoid the potential synthesis/simulation mismatch.

## Case 2

```
module asyncload (d, d0, load, clk, q);
 input d, d0, load, clk;
 output q;
 reg q;
 wire tmp_set = load & d0;
 wire tmp_rst = load & ~d0;

 always @(posedge clk or posedge tmp_rst or posedge tmp_set)
 begin
 if (tmp_rst)
 q = 0;
 else if (tmp_set)
 q = 1;
 else
 q = d;
 end
endmodule
```

In the actual hardware implementation, the resulting logic is the same for both cases 1 and 2 and there is nothing detrimental in case 1 other than the potential synthesis/simulation mismatch. If you require the code as in case 1 and can tolerate the mismatch, you can ignore the advisory.

## CL308

**@E: Latch RAM support is a beta feature that does not generate a technology-mapped netlist. Turn off latch RAM support to create a technology implementation.**

The -latchram option, when set, infers latch RAMs, but does not generate a technology-mapped netlist.

### Action

When a latch RAM is inferred, the compiler generates the above error, but creates a schematic of the compiled view showing the latch memories. To get a technology-mapped netlist, disable the -latchram option (option set -latchram 0) and recompile the design before mapping. The mapped view schematic displays the implementation as individual latches.

## CL309

**@E: Unable to create intermediate object -- make sure top level is specified correctly**

This error is displayed by the Verilog compiler when there is no output netlist. The output netlist is expected to be written towards the end of the Verilog compiler. The most common reason for not writing the netlist is wrong specification of the top-level name. This can be the actual top or intermediate top created due to language crossing.

### Action

Correct the top-level name.

## CL317

**@W: Safe encoding was requested for %r but no state machine was inferred from the flip-flop. Check if a state machine should be inferred.**

Indicates that the compiler is unable to extract a safe state machine. This occurs when the syn\_encoding directive is set to safe and the VHDL Default Enum Encoding option is set to onehot.

### Action

Verify the state machine logic.

### Workaround

Select a different safe encoding option or let the tool set a default option. To choose a different or default encoding option:

From the GUI, go to Implementation Options > VHDL > Default Enum Encoding.

Using TCL script:

```
set_option -default_enum_encoding default
```

- Enable the Preserve and Decode Unreachable States option in the High-Reliability tab.
- Make sure the state machine has when others clause specified in the VHDL case statements.

## CL319

**@E: Error reading RTLDB \'%s\' from unified compile database\n**

The RTL name database generated by the launch uc command is either corrupted or incomplete.

### Action

Run the launch uc command again.

## CL321

### **@E: File %s: Expecting identifier value on line %d\n**

This error occurs while reading the file that provides the VHDL 2019 conditional analysis identifiers and values. The expected format is one identifier and string literal per line. For example:

```
DEBUG_LEVEL "1"
MULT_STYLE "array"
```

### Action

Verify the following conditions:

- Each line contains only one identifier/value-pair.
- Values of each identifier are given as strings.

## CL322

### **@E: File %s: Expecting quoted string value for identifier '%r' on line %d\n**

This error occurs while reading the file that provides the VHDL 2019 conditional analysis identifiers and values. The expected format is one identifier and string literal per line. For example:

```
DEBUG_LEVEL "1"
MULT_STYLE "array"
```

The values must be string literals (quoted values, as shown above).

## Action

Make sure that the values given for the conditional analysis identifiers are provided as quoted strings.

## CHAPTER 19

# CS Messages 101 – 268

---

## CS101

### **@W: Index <4> is out of range for variable <out>**

A signal is indexed with a value that is outside its defined range. In the following test case, an assignment is made to signal out[4] that is outside the range of the defined signal which causes the compiler to error out.

```
module error(input i, j,
 input clk,
 output[3:0] out,
 output reg q);
 reg temp;
 always@(posedge clk)
 begin
 temp <= j;
 end
 assign out[4] = temp; // 4 is out of range for signal out
endmodule
```

## Action

Either increase the range of the defined signal or modify the index such that its value is within the defined index range as shown in the corrected test case below.

```
module error(input i, j,
 input clk,
 output[3:0] out,
 output reg q);
 reg temp;
 always@(posedge clk)
 begin
 temp <= j;
 end
 assign out[3] = temp;
endmodule
```

# CS102

## @E: Negative repeat count

The repeat count evaluates to a negative integer. Verilog allows you to create an expression by repetitive concatenation of a smaller expression. The replication is done by specifying a repeat count that must evaluate to a constant positive integer. In the following test case, the compiler errors out when the repeat count evaluates to a negative integer.

```
module temp (
 input clk,
 input [1:0] b,
 output [3:0] q);
 reg [1:0] tmp_reg;
 parameter temp = -1;

 always@(posedge clk)
 begin
 tmp_reg = b;
 end
```

```
assign q = {temp{tmp_reg}}; // temp = -1
endmodule
```

## Action

Make sure that the repeat count is a constant positive integer. The test case below shows the correct use of repeat concatenation in Verilog.

```
module temp (
 input clk,
 input [1:0] b,
 output [3:0] q);
 reg [1:0] tmp_reg;
 parameter temp = 2;

 always@(posedge clk)
 begin
 tmp_reg = b;
 end

 assign q = {temp{tmp_reg}}; // temp = 2
endmodule
```

# CS109

## @E: Expecting module level statement

There is an illegal or incorrect statement in a Verilog module. The following test case includes a typographical error in the assign statement which results in the error.

```
module test (input[3:0] i, j,
 input clk,
 output[3:0] out, out2);
 reg[3:0] temp;

 always@(posedge clk)
 begin
 temp = j;
 end
```

```
aassign out2 = temp; // assign is spelled incorrectly
and (out[0], i[3], j[0]);
endmodule
```

### Action

Be sure to use the correct spelling in the assign statement as shown in the corrected test case below.

```
module test (input[3:0] i, j,
 input clk,
 output[3:0] out, out2);
 reg[3:0] temp;

 always@(posedge clk)
 begin
 temp = j;
 end

 assign out2 = temp;
 and (out[0], i[3], j[0]);
endmodule
```

# CS110

### @E: Expecting terminal list

The terminal (port connection) list was missing. When instantiating a module in Verilog, the module name and instance identifier must be followed by a port connection list. The absence of a port list causes the error.

```
module fad1 (output c,
 input a, b);
 assign c = a & b;
endmodule

module my_reg (input[3:0] i, j,
 input clk,
 output[3:0] out, out2);
 reg[3:0] temp;
```

```
always@(posedge clk)
begin
 temp = j;
end

assign out2 = temp;
fad1 my_inst // missing port connection
endmodule
```

## Action

Add the required port connection list using an ordered list or a named list as shown in the corrected test case below.

```
module fad1 (output c,
 input a, b);
 assign c = a & b;
endmodule

module my_reg (input[3:0] i, j,
 input clk,
 output[3:0] out, out2);
 reg[3:0] temp;

 always@(posedge clk)
 begin
 temp = j;
 end

 assign out2 = temp;
 fad1 my_inst (out[3], i[2], i[1]);
endmodule
```

# CS112

## @E: Expecting port name

An input/output declaration statement was incomplete. In the following test case, the output declaration does not contain a port name which causes the error.

```
moduledff(q, data, clk, rst);
output [7:0];
input [7:0]data;
inputrst, clk;
reg [7:0]q;

always @(posedge clk)
begin
if (rst)
 q <= 8'b01011011;
else
 q <= data;
end

endmodule
```

## Action

Be sure to specify a port name in the output declaration statement as shown in the corrected test case below.

```
moduledff(q, data, clk, rst);
output [7:0]q;
input [7:0]data;
inputrst, clk;
reg [7:0]q;

always @(posedge clk)
begin
if (rst)
 q <= 8'b01011011;
else
 q <= data;
end

endmodule
```

# CS134

## @W: Bad property value for <syn\_probe> -- only 0 and 1 are legal.

A syn\_probe attribute is being applied to a net with a numeric value of other than 1 or 0. In the following test case, signal tmp is assigned the syn\_probe attribute with the numeric value 21. The tool ignores the syn\_probe attribute and generates this warning to indicate that the only legal value for the syn\_probe attribute is 0 or 1.

```
module reg_reg (a, clk, data, qb);
 input clk;
 input a, data;
 output qb;
 wire tmp /* synthesis syn_probe=21 */;
 reg qa;
 reg qb;
 assign tmp = data & a;

 always@ (posedge clk)
 qb <= qa;
 always@ (posedge clk)
 qa <= tmp;

endmodule
```

### Action

Assign only values 0 or 1 to the syn\_probe attribute. To eliminate this warning in the above test case, specify the numeric value 0 to disable or 1 to enable the syn\_probe attribute.

```
module reg_reg (a, clk, data, qb);
 input clk;
 input a, data;
 output qb;
 wire tmp /* synthesis syn_probe=1 */;
 reg qa;
 reg qb;
 assign tmp = data & a;

 always@ (posedge clk)
 qb <= qa;
 always@ (posedge clk)
 qa <= tmp;
```

```
endmodule
```

## CS138

### **@W: Macro definition for <MAX\_BUS> not found. Cannot undefine.**

A ‘`undef` Verilog compiler directive is being used with a macro that has not been previously defined. In the test case below, the macro `MAX_BUS` is undefined (the macro was not defined with a preceding ‘`define` compiler directive). In the test case below, the compiler cannot `undefine MAX_BUS` which results in the above warning.

```
'define WIDTH 16
#define WORD 18
#define WIDTH 8
`undef MAX_BUS
module dff1(q, d, clk, set, reset);
 input [`WIDTH-1 :0] d;
 input clk, set, reset;
 output [`WIDTH-1 :0] q;
 reg [`WIDTH-1 :0] q;

 always @(posedge clk or posedge set or posedge reset)
 begin
 if (reset)
 q <= 0;
 else if (set)
 q <= 1;
 else
 q <= d;
 end
endmodule
```

### Action

Make sure that the `‘undef` is used to remove the definition of a previously defined text macro as shown in the following code segment:

```
`define WORD 16 // Creates a macro for text substitution
wire [`WORD :1] BUS;
...
`undef `WORD
// The definition of WORD is no longer available after
// this `undef directive
```

## CS139

**@W: state\_machine attribute has been renamed to syn\_state\_machine.**  
**Change state\_machine usage to: synthesis state\_machine = 1/0**  
**for upward compatibility.**

The synthesis directive state\_machine has been encountered in the HDL code. The new synthesis directive for enabling/disabling state machine extraction on a per state machine basis is syn\_state\_machine, which is used on the state machine register as follows:

```
/* synthesis syn_state_machine =1 */ ;
```

The old directive functions correctly and is downward compatible. Use the synthesis syn\_state\_machine directive for software upward compatibility. Note that you must assign a value to the new directive.

```
module statmch1 (launch_shuttle, land_shuttle, start_countdown,
 start_trip_meter, clk, all_systems_go, just_launched,
is_landed,
 cnt, abort_mission);
output launch_shuttle, land_shuttle, start_countdown,
 start_trip_meter;
input clk, just_launched, is_landed, abort_mission,
all_systems_go;
input [3:0] cnt;
reg launch_shuttle, land_shuttle, start_countdown,
start_trip_meter;
// parameters for the one-hot encoding of the states
parameter HOLD = 5'h1, SEQUENCE = 5'h2, LAUNCH = 5'h4;
parameter ON_MISSION = 5'h8, LAND = 5'h10;
reg [4:0] present_state /* synthesis state_machine */;
reg [4:0] next_state;
```

```
always @ (negedge clk or posedge abort_mission)
begin
 /* set the outputs to some default values (so you don't have to
 set them in every case below */
 {launch_shuttle, land_shuttle, start_trip_meter,
 start_countdown} = 4'b0;

 /* check the value of the asynchronous reset */
 if (abort_mission)
 next_state = LAND;
 else begin
 /* set the next_state to be the present_state by default */
 next_state = present_state;
 /* depending on the present_state and inputs, the case items set
 the next_state variable and output values */
 case (present_state)
 HOLD: if (all_systems_go) begin
 next_state = SEQUENCE;
 start_countdown = 1;
 end
 SEQUENCE: if (cnt == 0) next_state = LAUNCH;
 LAUNCH: begin
 next_state = ON_MISSION;
 launch_shuttle = 1;
 end
 ON_MISSION:
 // Stay on mission until abort mission
 if (just_launched) start_trip_meter = 1;
 LAND: if (is_landed)
 next_state = HOLD;
 else land_shuttle = 1;
 /* set the default case to 'bx (don't care) or some known state,
 so it matches simulation before you do a reset */
 default: next_state = 'bx;
 endcase
 end // end of if-else
 /* make sure you set the present_state variable to the next_state
 that you just assigned, so it has the correct value at the next
 active clock edge */
 present_state = next_state;
end // end of always

endmodule
```

## Action

For upward software compatibility, make sure that you use the `syn_state_machine` synthesis directive instead of the old `state_machine` synthesis directive. To eliminate the warning from the above test case, change the name of the attribute to `syn_state_machine` as shown in the corrected test case below.

```
module statmch1 (launch_shuttle, land_shuttle, start_countdown,
 start_trip_meter, clk, all_systems_go, just_launched,
 is_landed,
 cnt, abort_mission);
 output launch_shuttle, land_shuttle, start_countdown,
 start_trip_meter;
 input clk, just_launched, is_landed, abort_mission,
 all_systems_go;
 input [3:0] cnt;
 reg launch_shuttle, land_shuttle, start_countdown,
 start_trip_meter;
 // parameters for the one-hot encoding of the states
 parameter HOLD = 5'h1, SEQUENCE = 5'h2, LAUNCH = 5'h4;
 parameter ON_MISSION = 5'h8, LAND = 5'h10;
 reg [4:0] present_state /* synthesis syn_state_machine =1 */;
 reg [4:0] next_state;

 always @ (negedge clk or posedge abort_mission)
 begin
 /* set the outputs to some default values (so you don't have to
 set them in every case below */
 {launch_shuttle, land_shuttle, start_trip_meter,
 start_countdown} = 4'b0;

 /* check the value of the asynchronous reset */
 if (abort_mission)
 next_state = LAND;
 else begin
 /* set the next_state to be the present_state by default */
 next_state = present_state;
 /* depending on the present_state and inputs, the case items set
 the next_state variable and output values */
 case (present_state)
 HOLD: if (all_systems_go) begin
 next_state = SEQUENCE;
 start_countdown = 1;
 end
 SEQUENCE: if (cnt == 0) next_state = LAUNCH;
```

```
LAUNCH: begin
 next_state = ON_MISSION;
 launch_shuttle = 1;
 end
ON_MISSION:

// Stay on mission until abort mission
if (just_launched) start_trip_meter = 1;
LAND: if (is_landed)
 next_state = HOLD;
else land_shuttle = 1;
/* set the default case to 'bx (don't care) or some known state,
so it matches simulation before you do a reset */
default: next_state = 'bx;
endcase
end // end of if-else
/* make sure you set the present_state variable to the next_state
that you just assigned, so it has the correct value at the next
active clock edge */
present_state = next_state;
end // end of always

endmodule
```

## CS140

**@W: black\_box attribute has been renamed to syn\_black\_box.**  
**Change black\_box usage to synthesis syn\_black\_box for upward compatibility.**

A synthesis black\_box directive has been encountered in the HDL code. The new synthesis directive for declaring a black box is syn\_black\_box.

### Verilog Test Case

The syn\_black\_box directive is used on a module that must be defined as a black box as shown in the test case below.

```
/* synthesis syn_black_box */
```

The old black box directive is downward compatible. Use the synthesis `syn_black_box` directive to make sure that your design is upward compatible with future software versions. The following Verilog test case results in the warning.

```
module adder (cout, sum, a, b, cin) /* synthesis black_box */;
parameter size = 8; /* declare a parameter. default required */
output cout;
output [size-1:0] sum; // sum uses the size parameter
input cin;
input [size-1:0] a, b; // 'a' and 'b' use the size parameter
assign {cout, sum} = a + b + cin;
endmodule

module adder8 (cout, sum, a, b, cin);
output cout;
output [7: 0] sum;
input [7: 0] a, b;
input cin;
adder adder_1 (cout, sum, a, b, cin);
endmodule
```

## VHDL Test Case

The `syn_black_box` directive is used on the architecture of the entity, which must be defined as a black box as shown below.

```
attribute black_box : boolean;
attribute black_box of behave : architecture is true;
```

The old black box directive is downward compatible. Use the new synthesis directive `syn_black_box` to ensure your design is upward compatible with future software versions. The following VHDL test case causes the above warning.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity addern is
 port (a,b: in std_logic_vector (7 downto 0);
 result: out std_logic_vector(7 downto 0));
end addern;
```

```

architecture behave of addern is
attribute black_box : boolean;
attribute black_box of behave : architecture is true;
begin
 result <= a + b;
end;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity adders is
 port (a,b: in std_logic_vector(7 downto 0);
 result: out std_logic_vector(7 downto 0));
end adders;

architecture behave of adders is
component addern is
 port (a,b: in std_logic_vector (7 downto 0);
 result: out std_logic_vector(7 downto 0));
end component;

begin
I1:addern
 port map (a => a, b=>b,result=>result);
end behave;

```

## Action

Use the `syn_black_box` synthesis directive instead of the old synthesis directive `black_box` for upward compatibility with subsequent versions of the tool.

## Corrected Verilog Test Case

To avoid this warning in the Verilog test case, change the name of the directive to `syn_black_box`.

```

module adder (cout, sum, a, b, cin) /* synthesis syn_black_box */;
parameter size = 8; /* declare a parameter. default required */
output cout;
output [size-1:0] sum; // sum uses the size parameter
input cin;
input [size-1:0] a, b; // 'a' and 'b' use the size parameter
assign {cout, sum} = a + b + cin;
endmodule

```

```
module adder8 (cout, sum, a, b, cin);
output cout;
output [7: 0] sum;
input [7: 0] a, b;
input cin;
adder adder_1 (cout, sum, a, b, cin);
endmodule
```

## Corrected VHDL Test Case

To eliminate the warning in the VHDL test case, replace `black_box` with `syn_black_box`.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity addern is
 port (a,b: in std_logic_vector (7 downto 0);
 result: out std_logic_vector(7 downto 0));
end addern;

architecture behave of addern is
attribute syn_black_box : boolean;
attribute syn_black_box of behave : architecture is true;

begin
 result <= a + b;
end;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity adders is
 port (a,b: in std_logic_vector(7 downto 0);
 result: out std_logic_vector(7 downto 0));
end adders;

architecture behave of adders is
component addern is
 port (a,b: in std_logic_vector (7 downto 0);
 result: out std_logic_vector(7 downto 0));
end component;
```

```
begin
I1: addern
 port map (a => a, b=>b,result=>result);
end behave;
```

## CS141

### **@W: Unrecognized synthesis directive <i><looplimit></i>. Verify the correct directive name.**

An undefined synthesis directive was encountered by the synthesis tool. Usually it is a typographical error or a directive intended for another tool (e.g., synopsys sync\_set\_reset). The compiler ignores any synthesis directive that it does not recognize. The following test case reports this warning.

```
module test (bus_a,clk,bus_out);
input [3:0] bus_a;
output [3:0] bus_out;
input clk;
reg [3:0] bus_out;
integer i;

always @(posedge clk) // synthesis looplimit 2001
 for (i=0; i <= 1997; i=i+1)
 bus_out <= bus_a + i;
endmodule
```

### Action

Make sure that the correct synthesis directive is used in the code. To eliminate the warning in the above test case, correct the `loop_limit` directive as shown in the corrected test case below.

```
module test (bus_a,clk,bus_out);
input [3:0] bus_a;
output [3:0] bus_out;
input clk;
reg [3:0] bus_out;
integer i;
```

```
always @(posedge clk) // synthesis loop_limit 2001
 for (i=0; i <= 1997; i=i+1)
 bus_out <= bus_a + i;
endmodule
```

The warning is generated if the comment includes either the word synthesis or synopsys followed by an unrecognized directive string. For example, the following line displays the message “Unrecognized synthesis directive fails.”

```
always @(posedge clk) // synthesis fails if this is missing
```

Similarly, encountering a Synopsys directive in the RTL code displays a message such as “Unrecognized synthesis directive sync\_set\_reset.”

## CS142

### **@W: Range of port <out> in port declaration and body are different.**

The port declared in the module has a different range from that used in the module description. For the following test case, the range (output [4:2] out) is different from the range of the out port used within the module (reg [2:0] out).

```
module encoder1(none_on, out, in);
 output none_on;
 output [4:2] out;
 input [7:0] in;
 reg [2:0] out;
 reg none_on;

 always @(in)
 begin: local
 integer i;
 out = 0;
 none_on = 1;
 /* returns the value of the highest bit number turned on */
 for (i = 0; i < 8; i = i +1)
 begin
 if (in[i]) begin
 out = i;
```

```
 none_on = 0;
 end
end
end
endmodule
```

## Action

Make sure that the ranges match so that there is no mismatch between RTL and post synthesis simulation. To eliminate the warning in the above test case, change the range for out as shown in the corrected test case below.

```
module encoder1(none_on, out, in);
output none_on;
output [2:0] out;
input [7:0] in;
reg [2:0] out;
reg none_on;

always @(in)
begin: local
integer i;
out = 0;
none_on = 1;
/* returns the value of the highest bit number turned on */
for (i = 0; i < 8; i = i +1)
begin
if (in[i]) begin
out = i;
none_on = 0;
end
end
end
endmodule
```

# CS151

## @E: Index <16> is out of range for memory <mem>

When addressing a memory element using an index for a read or write operation, the index was out of the defined address range for that memory element. In the following test case, the index evaluates to a value outside the defined address range which results in the error.

```
module error (
 input clk, en,
 input [3:0] addr, raddr,
 input [7:0] data,
 output [7:0] q, q2);
 reg [7:0]mem[15:0];
 parameter test_addr = 16; // 16 is outside the range 0 to 15

 always@(posedge clk)
 if (en)
 mem[addr] <= data;
 assign q = mem[raddr];
 assign q2 = mem[test_addr];
endmodule
```

### Action

Make sure that the index used to address the memory element always evaluates to a value within the defined address range as shown in the corrected test case below.

```
module test1 (
 input clk, en,
 input [3:0] addr, raddr,
 input [7:0] data,
 output [7:0] q, q2);
 reg [7:0]mem[15:0];
 parameter test_addr = 15;

 always@(posedge clk)
 if (en)
 mem[addr] <= data;
 assign q = mem[raddr];
 assign q2 = mem[test_addr];
endmodule
```

# CS153

## @E: Can't mix blocking and non-blocking assignments to a variable

Blocking and non-blocking statements are assigned to the same variable of a scalar type. Procedural statements (statements within an always or initial block) in Verilog can be one of two forms:

- blocking procedural statement (=)
- non-blocking procedural statement(<=)

Blocking procedural assignments are executed before any statements that follow them, whereas in a non-blocking assignment, the assignment to the target is not blocked, but is scheduled to occur at the end of the current time step. These two statements create different logic structures and their results cannot be synthesized when both types of statements are assigned to the same variable. The compiler errors out on a design that uses both statement types in the code for the same variable.

In the test case below, both a blocking statement and a non-blocking statement are used for registered output signal q within an always block which causes the error.

```
module dff1(d, clk, rst, q);
 output q;
 input d;
 input clk,rst;
 reg q;

 always @(posedge clk)
 begin
 if (rst)
 q <= 1'b0; // non-blocking statement
 else
 q = d; // blocking statement
 end
endmodule
```

### Action

Use only one type of statement when making a variable assignment. Use blocking statements to model combinational logic, and use non-blocking statements (shown below) to model sequential logic.

```
module dff1(d, clk, rst, q);
output q;
input d;
input clk,rst;
reg q;

always @(posedge clk)
begin
 if (rst)
 q <= 1'b0;
 else
 q <= d;
end
endmodule
```

## CS157

### @E: Range of port in module definition and body do not overlap

The range defined for a port at the module-definition level does not match the range defined in the corresponding port declaration in the body as shown in the test case below.

```
module test (in1[4:0], out1[3:0]);
input [3:0] in1;
output [3:0] out1;

assign out1 = in1;
endmodule
```

### Action

Change the range either at the module-definition or port-declaration level to have matching range values.

# CS158

## @E: Duplicate port names defined

The same port name appears more than once in the port list of the module declaration as shown in the test case below.

```
module test (bus_a, clk, bus_out, bus_out);
 input [3:0] bus_a;
 output [7:0] bus_out;
 input clk;
 reg [3:0] bus_out;

 always @(posedge clk)
 bus_out[3:0] <= bus_a;

endmodule
```

### Action

Make sure that the ports in the module have unique names. In the test case above, you can correct the error by deleting the duplicate name from the port list.

```
module test (bus_a, clk, bus_out);
 input [3:0] bus_a;
 output [7:0] bus_out;
 input clk;
 reg [3:0] bus_out;

 always @(posedge clk)
 bus_out[3:0] <= bus_a;

endmodule
```

# CS160

## @E: Bad or missing port direction for '<q>'

A module declaration has a port in the port list, but not in the body of the module. In the following test case, q is defined in the port list, but is not declared in the module body which results in the error.

```
module dff (q, data1, data2, clk);
 input data1, data2, clk;
 reg q1,q2;
 wire q;

 always @(posedge clk)
 begin
 q1 <= data1;
 end

 always @(negedge clk)
 begin
 q2 <= data2;
 end

 assign q = q1 | q2;
endmodule
```

## Action

Edit the code to define the port in the body of the module as shown in the corrected test case below or, if the port is not required, remove the port name from the port list.

```
module dff (q, data1, data2, clk);
 input data1, data2, clk;
 output q; // define port q
 reg q1,q2;
 wire q;

 always @(posedge clk)
 begin
 q1 <= data1;
 end
```

```
always @(negedge clk)
begin
 q2 <= data2;
end

assign q = q1 | q2;
endmodule
```

## CS162

### **@E: Loop iteration limit <limit> exceeded - add '// synthesis loop\_limit <limit>' before the loop construct**

The loop iteration limit has been exceeded. The compiler currently limits loop iteration index to 1999 to avoid an infinite loop that can result in out-of-memory errors. To override the loop limit and compile loops larger than 1999, specify the `loop_limit` synthesis directive before compiling your design. The following design results in this error because the loop is taken 2001 times which exceeds the default limit of 1999.

```
module error (input [7:0] bus_a, output wire [7:0] bus);
reg [7:0] bus_out;
integer i;

always @(bus_a)
for (i=0; i <= 2000; i=i +1)
 bus_out <= bus_a + bus_out;

assign bus = bus_out;
endmodule
```

### Action

The test case can be rewritten using the synthesis directive `synthesis loop_limit` before the loop construct to increase the loop limit as shown in the test case below.

```
module error (input [7:0] bus_a, output wire [7:0] bus);
reg [7:0] bus_out;
integer i;
```

```
always @(bus_a) /* synthesis loop_limit 2001 */
for (i=0; i <= 2000; i=i +1)
 bus_out <= bus_a + bus_out;

assign bus = bus_out;
endmodule
```

## CS167

### @E: Size mismatch of ports in array instances

There is a mismatch between the port width and the array of instances specified. Verilog allows you to instantiate arrays of instances in a single instantiation statement. In the test case below, there are nine instances of the module `my_dff` (instance names `r0` through `r8`). However, the `d` and `q` ports are defined as eight bits wide which results in the error.

```
module array_of_inst(clk, d, q);
 input clk;
 input [7:0] d;
 output [7:0] q;
 my_dff r[8:0] (clk, d, q);
endmodule

module my_dff(clk, d, q);
 input clk, d;
 output q;
 reg q;

 always @(posedge clk) q<= d;
endmodule
```

### Action

The above test case would be expanded nine times as

```
my_dff r0 (clk, d[0], q[0]);
.
.
.
my_dff r8 (clk, d[8], q[8]);
```

Correct the mismatch between ports widths and the specified array by changing the input/output widths as shown in the corrected test case below.

```
module array_of_inst(clk, d, q);
 input clk;
 input [8:0] d;
 output [8:0] q;
 my_dff r[8:0] (clk, d, q);
endmodule

module my_dff(clk, d, q);
 input clk, d;
 output q;
 reg q;

 always @(posedge clk) q<= d;
endmodule
```

## CS168

### @E: Port <out> does not exist

An attempt was made to connect a signal from the instantiating module to a nonexistent port in the instantiated module. When instantiating a module using a named connection list, the port names must exist in the module definition. In the following test case, port out does not exist in instantiated module fad1 which results in the error.

```
module fad1 (output c,
 input a, b);
 assign c = a & b;
endmodule

module my_reg (input[3:0] i, j,
 input clk,
 output[3:0] out, out2);
 reg[3:0] temp;

 always@(posedge clk)
 begin
 temp = j;
 end
```

```
assign out2 = temp;
fad1 my_inst (.out(out[3]), .a(i[1]), .b(i[0]));
endmodule
```

## Action

Use the correct port names in the instantiating module as shown in the corrected test case below.

```
module fad1 (output c,
 input a, b);
 assign c = a & b;
endmodule

module my_reg (input[3:0] i, j,
 input clk,
 output[3:0] out, out2);
 reg[3:0] temp;
 always@(posedge clk)
 begin
 temp = j;
 end
 assign out2 = temp;
 fad1 my_inst (.c(out[3]), .a(i[1]), .b(i[0]));
endmodule
```

## CS169

### @E: Duplicate connection to named port <a>

When instantiating a Verilog module with ports connected by name, only one connection is allowed per port. In the following test case, port a is connected twice which results in the error.

```
module gate (
 output c,
 input a, b);
 assign c = a & b;
endmodule
```

```
module error (
 input[3:0] i, j,
 input clk,
 output[3:0] out, out2);
 reg[3:0] temp;

 always@(posedge clk)
 begin
 temp = j;
 end

 assign out2 = temp;
 gate my_gate (.c(out[3]), .a(i[1]), .b(i[0]), .a(i[2]));
endmodule
```

## Action

Remove any duplicate or multiple connections to the same port as shown in the corrected test case below.

```
module gate (
 output c,
 input a, b);
 assign c = a & b;
endmodule

module error (
 input[3:0] i, j,
 input clk,
 output[3:0] out, out2);
 reg[3:0] temp;

 always@(posedge clk)
 begin
 temp = j;
 end

 assign out2 = temp;
 gate my_gate (.c(out[3]), .a(i[1]), .b(i[0]));
endmodule
```

# CS170

## @E: Parameters must be integers

A non-integer literal was found in a parameter map. In the following test case, a real value is assigned to the parameter `adder_width` which causes the error.

### Example 1

```
module adder(cout, sum, a, b, cin);
 parameter adder_width = 1.0; // declare a parameter. Default
 // required
 output cout;
 output [adder_width -1:0] sum; // sum uses the size parameter
 input cin;
 input [adder_width -1 :0] a, b;
 assign {cout, sum} = a + b + cin;
endmodule

module adder16(cout, sum, a, b, cin);
 output cout;
 /* We are also using a parameter at this level of hierarchy */
 parameter size = 16; // We want a 16-bit adder
 output [size - 1: 0] sum;
 input [size - 1: 0] a, b;
 input cin;
 adder #(.adder_width(16)) my_adder (cout, sum, a, b, cin);
endmodule
```

This error can also occur if you are using the explicit parameter passing feature of Verilog 2001 and you do not enable the Verilog 2001 switch.

### Example 2

#### Verilog 2001 Explicit Parameter Passing Without Enabling Verilog 2001 Switch

```
// adder.v
module adder(cout, sum, a, b, cin);
 parameter adder_width = 1; // declare a parameter. Default
 // required
 output cout;
 output [adder_width -1:0] sum; // sum uses the size parameter
```

```

input cin;
input [adder_width -1 :0] a, b; // 'a' and 'b' use the
 // size parameter
assign {cout, sum} = a + b + cin;
endmodule

// adder_16.v
`include "adder.v"
module adder16(cout, sum, a, b, cin);
output cout;
/* We are also using a parameter at this level of hierarchy */
parameter size = 16; // We want a 16-bit adder
output [size - 1: 0] sum;
input [size - 1: 0] a, b;
input cin;
adder #(adder_width(16)) my_adder (cout, sum, a, b, cin);
endmodule

```

## Action

Make sure that the parameters are assigned integer values. To correct the first test case, change the parameter value to an integer as shown in the corrected test case below:

```

module adder(cout, sum, a, b, cin);
parameter adder_width = 1; // declare a parameter. Default
 // required
output cout;
output [adder_width -1:0] sum; // sum uses the size parameter
input cin;
input [adder_width -1 :0] a, b;
assign {cout, sum} = a + b + cin;
endmodule

module adder16(cout, sum, a, b, cin);
output cout;
/* We are also using a parameter at this level of hierarchy */
parameter size = 16; // We want a 16-bit adder
output [size - 1: 0] sum;
input [size - 1: 0] a, b;
input cin;
adder #(adder_width(16)) my_adder (cout, sum, a, b, cin);
endmodule

```

If explicit parameter passing is used, then make sure the Verilog 2001 switch is on.

## Setting the Verilog 2001 Switch

To set the Verilog 2001 switch, do one of the following:

- include the Tcl command `set_option -vlog_std v2001` in your project file
- in the interactive mode, check the small box associated with Verilog 2001 on the Verilog tab of the Implementation options accessible from the project view in the user interface.

## Using Verilog 95 Syntax

If you are using the Verilog 95 syntax, change the adder # line in the adder16 module as shown in the test case below.

```
module adder(cout, sum, a, b, cin);
parameter adder_width = 1; // declare a parameter. Default
 // required
output cout;
output [adder_width -1:0] sum; // sum uses the size parameter
input cin;
input [adder_width -1 :0] a, b;
assign {cout, sum} = a + b + cin;
endmodule

module adder16(cout, sum, a, b, cin);
output cout;
/* We are also using a parameter at this level of hierarchy */
parameter size = 16; // We want a 16-bit adder
output [size - 1: 0] sum;
input [size - 1: 0] a, b;
input cin;
adder #(16) my_adder (cout, sum, a, b, cin);
endmodule
```

# CS172

**@E: The number of ports in the instantiation does not match the number of ports in the module definition for instance <instanceName>**

A mismatch in the number of ports exists between the instantiation and the actual instance. The following test case generates the error because the instantiation of submod has two ports, but the actual instance has three.

```
module error_msg (a, b, c);
 input a, b;
 output c;
 submod U1(a,b);
endmodule

module submod(a, b, c);
 input a, b;
 output c;
 assign c = a & b;
endmodule
```

## Action

This is a syntax error. The module has three ports, and therefore any instantiation of the module must contain the three ports. To eliminate the error in the above test case, change the number of ports in the instantiation as shown in the corrected test case below.

```
module error_msg (a, b, c);
 input a, b;
 output c;
 submod U1(a,b,c);
endmodule

module submod(a, b, c);
 input a, b;
 output c;
 assign c = a & b;
endmodule
```

If you need to keep a port open in Verilog, you can either specify it through named association or positional association.

```
submod U1(.a(a),.b(b),.c()); else //named association
```

```
submod U1(a,b,); //positional association
```

Outputs and inputs (except for top-level ports) that are not used can be removed from the design by the synthesis tool to give optimal results.

## CS176

### **@E: port declaration conflict over <data\_bus>**

Duplicate definitions were found for the typedefs in a SystemVerilog design. In the test case below, the Multiple File Compilation Unit switch is enabled, which instructs the compiler to treat all files in a synthesis project as a single compilation unit. By default, the scope of the compilation unit is a single Verilog file. Enabling the Multiple File Compilation Unit switch overrides the default and creates a common compilation unit for all files. When the switch is enabled for the following design, the 'include directives are in other files that use the common file with all the parameters, data types (typedefs), and functions which results in duplicate definitions for the typedefs.

```
// Defines.vh
// defines "data_bus" one time only

typedef struct packed {
 logic [7:0] data;
 logic par; }
data_bus;

// Transmit.v
`include "defines.vh"
module transmit (
 input wire clk, reset,
 input data_bus din,
 output reg [8:0] dout);
 always_ff @(posedge clk)
 if (reset) dout <= '0;
 else dout <= din + 1;
endmodule
```

```
// Receive.v
`include "defines.vh"
module receive (
 input wire clk,
 input data_bus din,
 output reg [8:0] dout);
always_ff @(posedge clk)
 dout <= din;
endmodule

// Test.v
`include "defines.vh"
module test (
 input wire clk, reset,
 input data_bus din,
 output reg [8:0] dout);
data_bus data_n_par;

transmit tx (
 .clk (clk),
 .reset (reset),
 .din (din),
 .dout (data_n_par));

receive rx (
 .clk (clk),
 .din (data_n_par),
 .dout (dout));
endmodule
```

## Action

Embed the `typedef` constructs within the `'ifndef` and `'endif` constructs, as shown below.

```
`ifndef DEFINE_ALREADY
`define DEFINE_ALREADY
 typedef definitions
`endif
```

# CS177

## @E: Expecting net name

A signal or port name was missing from a statement that required a net name entry. In the following test case, a net assignment is missing after an assign statement which results in the error.

```
module error(input[3:0] i, j,
 input clk,
 output[3:0] out, out2);
 reg[3:0] temp;

 always@(posedge clk)
 begin
 temp = j;
 end

 assign out2 = temp;
 assign // incomplete assignment
 and and_inst1 (out[0], i[3], j[0]);
endmodule
```

## Action

Be sure to complete the signal assignment to a declared net as shown in the corrected test case below.

```
module error (
 input[3:0] i, j,
 input clk,
 output[3:0] out, out2);
 reg[3:0] temp;

 always@(posedge clk)
 begin
 temp = j;
 end

 assign out2 = temp;
 assign out[1] = temp[1];
 and and_inst1 (out[0], i[3], j[0]);
endmodule
```

# CS178

## @E: Unknown assignment target

Earlier versions of the Verilog compiler required that all signals and ports be declared before any signal assignments were made to them (in current versions, undefined signals or ports are automatically assigned to wires and the above message is no longer generated). In the following test case, an output port is not declared. Running the test case on an earlier version of the compiler results in the error.

```
module error(input i, j,
 input clk,
// signal "out" not declared in port list
 output reg q);

 always@(posedge clk)
 begin
 q <= j;
 end

 assign out = i;
endmodule
```

### Action

Declare signal out as an output port in the module as shown in the corrected test case below.

```
module error(
 input i, j,
 input clk,
 output out,
 output reg q);

 always@(posedge clk)
 begin
 q <= j;
 end

 assign out = i;
endmodule
```

# CS179

## @E: Assignment target <t> should be a net type

A signal in the design was not used as the specified type required by the design. For example, when a signal is declared as a `reg`, but is used as a `wire` or a `wire` in a continuous assignment statement. In the test case below, signal `t` is declared as type `reg`, although the code uses it as type `wire`, which results in the error.

```
module dff1 (x, y, z, q, d, clk, rst);
 input x, y, z, d, rst, clk;
 output q;
 reg t; // This signal should be declared as type wire t;
 assign t = x & y & z | d;
 reg q;

 always @(posedge clk or posedge rst)
 begin
 if (rst)
 q <= 0;
 else
 q <= t;
 end
endmodule
```

### Action

Make sure that nets are declared in the Verilog language when they are assigned using continuous assignments (as in the case above) or when they are gate outputs. Modify the code to declare the signal `t` as a `wire` by making the changes shown in the corrected test case below.

```
module dff1 (x, y, z, q, d, clk, rst);
 input x, y, z, d, rst, clk;
 output q;
 wire t;
 assign t = x & y & z | d;
 reg q;
```

```
always @(posedge clk or posedge rst)
begin
if (rst)
 q <= 0;
else
 q <= t;
end

endmodule
```

## CS185

### **@E: <out> is not a parameter name**

A non-parameter was used as a parameter. In the following test case, out in the assign statement, because it follows the pound sign (#), is incorrectly interpreted as a parameter which results in the error.

```
module test (a, b, out);
input a, b;
output out;
parameter temp = 10;
assign # out = a & b;
endmodule
```

### Action

Be sure to use parameters that are declared in the design. In the above test case, temp is a parameter. Edit the assign statement to use temp as shown in the corrected test case below.

```
module test (a, b, out);
input a, b;
output out;
parameter temp = 10;
assign #temp out = a & b;
endmodule
```

# CS187

## **@E: Expecting <symbol or word>**

Syntax error; the compiler expects a specific symbol or word. Click on an error in the following table for details about how to correct the syntax.

[@E: Expecting \(](#)

[@E: Expecting \) \(Verilog\)](#)

[@E: Expecting \) \(VHDL\)](#)

[@E: Expecting :](#)

[@E: Expecting ;](#)

[@E: Expecting \]](#)

[@E: Expecting => \(VHDL\)](#)

[@E: Expecting endmodule](#)

### **@E: Expecting (**

An expected opening parenthesis was not found. For example, the Verilog language requires the sensitivity list of an always block to be enclosed in parentheses. In the following test case, the missing opening parenthesis in the always block causes the error.

```
module dff(q, data, clk);
 output q;
 input data, clk;
 reg q;

 always @ posedge clk
 begin
 q <= data;
 end

endmodule
```

### Action

Add the opening parenthesis to the always block sensitivity list as shown in the corrected test case below.

```
moduledff(q, data, clk);
output q;
input data, clk;
reg q;

always @(posedge clk)
begin
 q <= data;
end

endmodule
```

## @E: Expecting )

The compiler expects a closing parenthesis for a particular syntax. For example, the Verilog language requires the sensitivity list of an always block to be enclosed in parentheses. In the following test case, the missing closing parenthesis in the always block causes the error.

```
moduledff(q, data, clk);
output q;
input data, clk;
reg q;

always @(posedge clk
begin
 q <= data;
end

endmodule
```

## Action

Add the closing parenthesis to the always block sensitivity list as shown in the corrected test case below.

```
moduledff(q, data, clk);
output q;
input data, clk;
reg q;

always @(posedge clk)
begin
 q <= data;
end
```

```
endmodule
```

### @E: Expecting :

The compiler expects a colon for a particular syntax. For example, the Verilog language requires a 2-dimensional array element to be written out as:

```
[index1:index2] mem [index3:index4]
```

In the following test case, the missing colon in the mem array statement causes the error.

```
module singleportram(q, a2, d, we, clk, en);
output [7:0] q;
input [7:0] d;
input [5:0] a2;
input clk, we, en;
reg [6:0] read_addr;
reg [6:0] mem [127
wire temp;
assign q = mem[read_addr];

always @(posedge clk) begin
if (we)
 mem[a2] <= d;
 read_addr <= a2;
end

endmodule
```

### Action

Complete the right member of the array as shown in the corrected test case below.

```
module singleportram(q, a2, d, we, clk, en);
output [7:0] q;
input [7:0] d;
input [5:0] a2;
input clk, we, en;
reg [6:0] read_addr;
reg [6:0] mem [127:0];
wire temp;
assign q = mem[read_addr];
```

```
always @(posedge clk) begin
 if (we)
 mem[a2] <= d;
 read_addr <= a2;
 end
endmodule
```

### @E: Expecting ;

The Verilog language requires all statements such as procedural, continuous, and declaration statements to end in a semicolon. In the following test case, the compiler errors out because a statement in the `always` block is not terminated with a semicolon.

```
moduledff(q, data, clk, rst);
output [7:0]q;
input [7:0]data;
input rst, clk;
reg [7:0]q;

always @(posedge clk)
begin
if (rst)
 q = 8'b01011011
else
 q = data;
end
endmodule
```

### Action

Make sure that all statements in Verilog end in a semicolon. To eliminate the error in the above test case, add a semicolon at the end of the statement as shown in the corrected test case below.

```
moduledff(q, data, clk, rst);
output [7:0]q;
input [7:0]data;
input rst, clk;
reg [7:0]q;
```

```
always @(posedge clk)
begin
if (rst)
 q = 8'b01011011;
else
 q = data;
end

endmodule
```

### @E: Expecting ]

The Verilog language requires an array element to have opening and closing square brackets as shown in the test case below.

```
[index1:index2] mem [index3:index4]
```

In the following test case, the missing closing square bracket in the mem assignment causes the error.

```
module singleportram(q, a2, d, we, clk, en);
output [7:0] q;
input [7:0] d;
input [6:0] a2;
input clk, we, en;
reg [6:0] read_addr;
reg[7:0] mem [127:0];
wire temp;
assign q = mem[read_addr];

always @(posedge clk) begin
if (we)
 mem[a2 <= d;
 read_addr <= a2;
end

endmodule
```

### Action

Add the closing bracket to the mem assignment as shown in the corrected test case below.

```
module singleportram(q, a2, d, we, clk, en);
output [7:0] q;
input [7:0] d;
input [6:0] a2;
input clk, we, en;
reg [6:0] read_addr;
reg[7:0] mem [127:0];
wire temp;
assign q = mem[read_addr];

always @(posedge clk) begin
if (we)
 mem[a2] <= d;
 read_addr <= a2;
end

endmodule
```

### @E: Expecting endmodule

The compiler parsed a Verilog module and did not encounter a corresponding `endmodule` statement. In the following test case, the `endmodule` statement is commented out which causes the error.

```
module counter1(out, cout, data, load, cin, clk);
output [7:0] out;
output cout;
input [7:0] data;
input load, cin, clk;
reg [7:0] out;

always @(posedge clk)
begin
 if (load)
 out = data;
 else
 out = out + cin;
end

assign cout = &out & cin;
//endmodule
```

## Action

This error typically occurs as a result of a user's copy and paste operations when the complete test case is not copied over. This error can also occur if incorrect syntax is present in the line previous to `endmodule` statement. To eliminate the error in the above test case, uncomment the `endmodule` statement.

```
module counter1(out, cout, data, load, cin, clk);
 output [7:0] out;
 output cout;
 input [7:0] data;
 input load, cin, clk;
 reg [7:0] out;

 always @(posedge clk)
 begin
 if (load)
 out = data;
 else
 out = out + cin;
 end

 assign cout = &out & cin;
endmodule
```

## @E: Expecting )

This error occurs when a closing parenthesis is missing in a VHDL statement. In the following test case, while declaring the size of the constant, the closing parenthesis was missing in the type declaration.

```
library ieee;
use ieee.std_logic_1164.all;

entity adder is
 port (a, b, cin: in std_logic;
 sum, cout:out std_logic);
end adder;
```

```

architecture behave of adder is
signal temp:std_logic;
constant temp2: std_logic_vector (3 downto 0 := "0000";
begin
 sum <= (a xor b) xor cin;
 cout <= (a and b) or (a and cin) or (b and cin);
end behave;
```

## Action

Make sure that all parenthesis pairs match up in the HDL code by adding the missing closing parenthesis.

```

library ieee;
use ieee.std_logic_1164.all;

entity adder is
 port (a, b, cin: in std_logic;
 sum, cout:out std_logic);
end adder;

architecture behave of adder is
signal temp:std_logic;
constant temp2: std_logic_vector (3 downto 0) := "0000";
begin
 sum <= (a xor b) xor cin;
 cout <= (a and b) or (a and cin) or (b and cin);
end behave;
```

## @E: Expecting =>

An equal to or greater than symbol ( $=>$ ) is missing from a port map statement. Port mapping using named association requires a  $=>$  between the signals of the component (formal) and the signals in the current level (actual). In the following test case, the U2 port map statement is missing a  $=>$  between cin and cin2 which results in the error.

```

library ieee;
use ieee.std_logic_1164.all;

entity adder_top is
 port (a, b, cin1, c, d, cin2: in std_logic;
 sum1, cout1, sum2, cout2: out std_logic);
end adder_top;
```

```
architecture behave of adder_top is
component adder is
 port (a, b, cin: in std_logic;
 sum, cout: out std_logic);
end component;

begin
 U1: adder port map (a=>a, b=>b, cin=>cin1, sum=>sum1,
 cout=>cout1);
 U2: adder port map (a=>c, b=>d, cin cin2, sum=>sum2,
 cout=>cout2);
end behave;
```

## Action

When port mapping using named association, place a => between the signals of the component and the signals in the current level. To eliminate the error in the above test case, place a => between cin and cin2 in the U2 port map statement.

```
library ieee;
use ieee.std_logic_1164.all;

entity adder_top is
 port (a, b, cin1, c, d, cin2: in std_logic;
 sum1, cout1, sum2, cout2: out std_logic);
end adder_top;

architecture behave of adder_top is
component adder is
 port (a, b, cin: in std_logic;
 sum, cout: out std_logic);
end component;

begin
 U1: adder port map (a=>a, b=>b, cin=>cin1, sum=>sum1,
 cout=>cout1);
 U2: adder port map (a=>c, b=>d, cin=>cin2, sum=>sum2,
 cout=>cout2);
end behave;
```

# CS188

## @E: Expecting delay specification

An incomplete or incorrect delay specification was encountered. In the test case below, the delay specification for the assign statement is incomplete which causes the error.

```
module error (
 input[3:0] i,
 input clk,
 output[3:0] out2);
 reg[3:0] temp;

 always@(posedge clk)
 begin
 temp = i;
 end

 assign #) out2 = temp;
endmodule
```

### Action

Either remove the delay specification or fully specify the delay as shown in the corrected test case below.

```
module test1 (
 input[3:0] i,
 input clk,
 output[3:0] out2);
 reg[3:0] temp;

 always@(posedge clk)
 begin
 temp = i;
 end

 assign #(10) out2 = temp;
endmodule
```

## CS189

### @E: capacitance is only valid for trireg nets

Charge strengths are specified only on trireg nets. In this case, a charge strength, such as small, medium, or large is specified for a data type other than trireg. For example, you cannot specify charge strength for a wire, reg, or other data type. In the following test case, a charge strength of small is specified on net out, which is an invalid capacitance specification.

```
module test (a, b, out);
 input a, b;
 output out;
 wire (small) out = a & b;
endmodule
```

#### Action

Replace wire with the trireg data type as shown in the corrected test case below.

```
module test (a, b, out);
 input a, b;
 output out;
 trireg (small) out = a & b;
endmodule
```

## CS190

### @E: drive specification is invalid

The table below outlines possible drive strength assignments based on the drive value (1 or 0). In this case, both drive strength assignments are defined by the same drive value.

Drive Value	Drive Strength Assignments				
1	supply1	strong1	pull1	weak1	highz1
0	supply0	strong0	pull0	weak0	highz0

In the test case below, the drive strength assignments are both specified by a drive value of 1 (weak1 and pull1).

```
module test (a, b, out);
 input a, b;
 output out;
 wire (pull1, weak1) out = a & b; //no drive value 0 definition
endmodule
```

## Action

To correct this syntax error, specify correct drive strengths. Although drive strengths are ignored during synthesis, the syntax error must be corrected before synthesis can proceed. The above test case must be modified to correct the conflict in drive strength specifications. To correct the error, weak0 is specified as the drive strength when the net is assigned a value of 0 as shown in the corrected test case below.

```
module test (a, b, out);
 input a, b;
 output out;
 wire (pull1, weak0) out = a & b;
endmodule
```

# CS191

## @E: Expecting drive strength

A drive strength assignment is incomplete. The table below outlines possible drive strength assignments based on the drive value (1 or 0).

Net Value	Drive Strength Assignments				
1	supply1	strong1	pull1	weak1	highz1
0	supply0	strong0	pull0	weak0	highz0

In the following test case, an incomplete drive strength assignment exists which causes the error.

```
module test (a, b, out);
 input a, b;
 output out;
 wire (strong1,) out = a & b;
endmodule
```

## Action

Make sure both drive strengths are specified correctly. Drive strengths are ignored during synthesis. In the corrected test case below, the drive strengths strong0 and strong1 are specified.

```
module test (a, b, out);
 input a, b;
 output out;
 wire (strong0, strong1) out = a & b;
endmodule
```

# CS192

### **@E: Expecting one non-highz strength**

The table below outlines possible drive strength assignments based on the drive value (1 or 0). In this case, both strength values are highz (high impedance). Only one of the strength values (1 or 0) can be assigned high impedance, the other must be a non-highz strength.

Net Value		Drive Strength Assignments			
1	supply1	strong1	pull1	weak1	highz1
0	supply0	strong0	pull0	weak0	highz0

In the following test case, highz0 and highz1 are specified as drive strengths of out.

```
module test (a, b, out);
 input a, b;
 output out;
 wire (highz0, highz1) out = a & b;
endmodule
```

## Action

Both net strength assignments cannot be high impedance. To correct the error, one of the drive strengths must be reassigned a non-high impedance strength. Drive strengths are not taken into account during synthesis and are ignored. In the corrected test case below, the highz1 assignment is changed to strong1.

```
module test (a, b, out);
 input a, b;
 output out;
 wire (strong1, highz0) out = a & b;
endmodule
```

# CS193

## @E: Expecting valid net name

An illegal literal was used as a net name such as Verilog reserved word. Net names must be user-defined words. In the following test case, output is used as a net name which results in the error.

```
module test (a, b, clk, out);
 input a, b;
 input clk;
 output out;
 reg out;
 wire output;
 assign output = (a == b) ? 1 : 0;

 always @(posedge clk)
 begin
 if (output)
 out <= a & b;
 end
```

```
endmodule
```

## Action

Use a user-defined net name instead of a reserved keyword for identifiers. All reserved keywords appear as blue in the HDL editor. The following test case shows how to eliminate the error. Every occurrence of signal output is changed to the valid net name out1.

```
module test (a, b, clk, out);
 input a, b;
 input clk;
 output out;
 reg out;
 wire out1;
 assign out1 = (a == b) ? 1 : 0;

 always @(posedge clk)
 begin
 if (out1)
 out <= a & b;
 end

endmodule
```

## CS204

### **@E: Signal <q> is missing from port list**

The Verilog language requires all declared ports to be listed in the port list. In the following test case, port q is declared in the module, but not in the port list which results in the error.

```
module dff(data, clk, rst);
 output [7:0]q;
 input [7:0]data;
 input rst, clk;
 reg [7:0]q;
```

```
always @(posedge clk)
begin
if (rst)
 q <= 8'b01011011;
else
 q <= data;
end

endmodule
```

## Action

Make sure that all declared ports are listed in the port list. To eliminate the error in the above test case, add port q to the port list as shown in the corrected test case below.

```
moduledff(q, data, clk, rst);
output [7:0]q;
input [7:0]data;
input rst, clk;
reg [7:0]q;

always @(posedge clk)
begin
if (rst)
 q <= 8'b01011011;
else
 q <= data;
end

endmodule
```

# CS215

## @E: Expecting module name

Verilog requires that an identifier or a module name follows the module keyword. This identifier or name cannot be a reserved word. The compiler errors out if a module name is missing or if one of the reserved keywords is used. In the following test case, the keyword reg is used as the module name which results in error.

```
module reg (input[3:0] i, j, // reg is not a valid module name
 input clk,
 output[3:0] out, out2);
reg[3:0] temp;
always@(posedge clk)
begin
 temp = j;
end

assign out2 = temp;
and and_inst1 (out[0], i[3], j[0]);
endmodule
```

## Action

Be sure to use a module name that is unique to the design as shown in the corrected test case below.

```
module my_reg (input[3:0] i, j,
 input clk,
 output[3:0] out, out2);
reg[3:0] temp;
always@(posedge clk)
begin
 temp = j;
end

assign out2 = temp;
and and_inst1 (out[0], i[3], j[0]);
endmodule
```

# CS216

## @E: Duplicate module name <adder>

The compiler found two definitions for the same module in the project. The duplication of module definitions usually occurs when a file containing the module description is explicitly added twice to a project and/or an ‘include

statement also references a file containing the same module description as the added file. Essentially, the user, by specifying the `include statement, has inadvertently added the same module description a second time.

This error is accompanied by the following warning:

```
@W: Previous module with same name adder
```

In the following test case, the project file indicates that the adder.v file, which contains the module description of module adder, is included in the project. The adder16.v file, which is also included in the project file, has an `include adder.v command. The compiler finds a duplicate description for the same module.

### **Project File Excerpt:**

```
#add_file options
add_file -verilog "adder.v"
add_file -verilog "adder16.v"
```

### **adder.v File**

```
module adder (cout, sum, a, b, cin);
parameter size = 1; /* declare a parameter. Default required */
output cout;
output [size-1:0] sum; // sum uses the size parameter
input cin;
input [size-1:0] a, b; // 'a' and 'b' use the size parameter
assign {cout, sum} = a + b + cin;
endmodule
```

### **adder16.v File**

```
`include "adder.v"
module adder16 (cout, sum, a, b, cin);
output cout;
/* We are also using a parameter at this level of hierarchy */

parameter my_size = 16; // We want a 16-bit adder
output [my_size - 1: 0] sum;
input [my_size - 1: 0] a, b;
input cin;

/* my_size overwrites size inside instance my_adder of adder */
adder #(my_size) my_adder (cout, sum, a, b, cin);
endmodule
```

## Action

For version 8.0 and later, duplicate module names can be used in the project when you select the Allow Duplicate Module option in the GUI or by using the equivalent command:

```
set_option -dup 1.
```

For Synplify Pro versions earlier than 8.0. Define all modules that are instantiated in your RTL code and add them to the project file. Alternatively, use `include *filename* in the HDL code to add the lower-level module files. When adding modules, make sure that all modules added to your project, including modules added directly and modules added through an `include statement, are unique. To eliminate the error in the above test case, either:

- Add the module definition of adder into the project file with the add file option and comment out the `include adder.v in the adder16.v file.
- Use `include adder.v to add the lower-level module files, but remove the following line from the project file:

```
add_file -verilog "adder.v"
```

# CS219

## @E: Expecting one of the keywords module, primitive or macromodule

The Verilog language requires the module description to start with the keyword module, primitive (for UDPs), or macromodule. The compiler errors out with the above message if one of these keywords is not used. The reason for the error could be a VHDL file renamed as a Verilog (.v) file, or a typographical error on the keyword module. Here are two examples that cause this error.

### VHDL File Named as *filename.v* File

```
library ieee;
use ieee.std_logic_1164.all;

entity dff1 is
port (data, clk, reset, set : in std_logic;
 qrs: out std_logic);
end dff1;
```

```

architecture async_set_reset of dff1 is
begin
 setreset: process (clk, reset, set)
 begin
 if reset = '1' then
 qrs <= '0';
 elsif set = '1' then
 qrs <= '1';
 elsif rising_edge(clk) then
 qrs <= data;
 end if;
 end process setreset;
end async_set_reset;

```

### Miss typed Keyword Module

```

mod array (clk, d, q);
input clk;
input [7:0] d;
output [7:0] q;
my_dff r[7:0] (clk, d, q);
endmodule

module my_dff (clk, d, q);
input clk, d;
output q;
reg q;

always @(posedge clk) q<= d;
endmodule

```

### Action

In either case, there are syntax errors that result in message. Make sure only Verilog files are named as .v files. Also make sure that the module description starts with the full word, module as shown in the corrected test case below.

```

module array (clk, d, q);
input clk;
input [7:0] d;
output [7:0] q;
my_dff r[7:0] (clk, d, q);
endmodule

```

```
module my_dff (clk, d, q);
 input clk, d;
 output q;
 reg q;

 always @(posedge clk) q<= d;
endmodule
```

A UDP declaration must start with the keyword primitive as shown below.

```
primitive mux2x1 (z, hab, bay, sel);
```

---

**Note:** The Verilog language is case sensitive. All Verilog keywords are lower case.

---

## CS222

### @W: Ignoring specify block

A specify block in the code was encountered. Specify blocks are used to specify delays in the module and to perform timing checks for the module. The following test case produces this warning for this condition.

```
// Simple flip-flop example without set or reset
module dff (data, clk, q);
 input [15:0]data;
 input clk;
 output [15:0]q;
 reg [15:0]q;

 always @(posedge clk)
 begin
 q = data;
 end
 specify
 specparam tSETUP =20, tHOLD =25;
 specparam tclk_q = (5:4:6);
 (clk *> q) = tclk_q;
 (data *> q) = 12;
 endmodule
```

## Action

Make sure that the design used for synthesis does not have specify blocks. All timing information is ignored by the compiler which can cause RTL/post synthesis mismatches. For the above test case, simply comment out the specify block to prevent possible mismatches between RTL and post synthesis simulation.

# CS223

### **@W: Port declaration for <Q> specifies a range, but later declarations do not**

A port declaration specifies a range of a vector, but when redeclared within the code as a wire or a reg, the range is not specified. In the test case below, the range for Q is specified in the port declaration as an 8-bit port (output [7:0] Q), but within the module it is declared as reg Q.

```
module prep1(Q, CLK, RST, S_L, S0, d0, d1, d2, d3);
 output [7:0] Q;
 input CLK, RST, S_L;
 input S1, S0;
 input [7:0] d0, d1, d2, d3;
 reg Q;
 reg [7:0] Y, q_reg; // q_reg is output from 8-bit register

 always @(S1 or S0 or d0 or d1 or d2 or d3)
 begin
 case ({S1, S0})
 2'b00: Y = d0;
 2'b01: Y = d1;
 2'b10: Y = d2;
 default : Y = d3;
 endcase
 end

 always @(posedge CLK or posedge RST)
 begin
 if (RST) begin
 q_reg = 0; // reset register
 Q = 0; // reset shift register
 end else if (S_L) begin
 end
```

```

 Q[7:0] = {Q[6:0],Q[7]}; // shift register
 q_reg = Y;
 end else begin
 Q = q_reg; // load from register
 q_reg = Y; // load from mux
 end
end
endmodule

```

## Action

Make sure that the port declarations as well as the redeclarations of the port match in width. In the above test case, specify a range for `reg` as shown in the corrected test case below.

```

module prep1(Q, CLK, RST, S_L, S1, S0, d0, d1, d2, d3);
output [7:0] Q;
input CLK, RST, S_L;
input S1, S0;
input [7:0] d0, d1, d2, d3;
reg [7:0] Q;
reg [7:0] Y, q_reg; // q_reg is output from 8-bit register

always @(S1 or S0 or d0 or d1 or d2 or d3)
begin
 case ({S1, S0})
 2'b00: Y = d0;
 2'b01: Y = d1;
 2'b10: Y = d2;
 default : Y = d3;
 endcase
end

always @(posedge CLK or posedge RST)
begin
 if (RST) begin
 q_reg = 0; // reset register
 Q = 0; // reset shift register
 end else if (S_L) begin
 Q[7:0] = {Q[6:0],Q[7]}; // shift register
 q_reg = Y;
 end else begin

```

```
 Q = q_reg; // load from register
 q_reg = Y; // load from mux
end
end
endmodule
```

## CS228

### @W: Nested block comments.

The Verilog compiler encountered a nested block comment in a module (nested block comments are not supported according to the LRM and are ignored in the RTL code). The nested block comment in the example below results in the warning.

```
module my_and (c,a,b);
 input a,b;
 output c;
 wire temp;
 /*initial block comment
 * nested block comment */
 assign temp =!b;
 assign c= a & temp;
endmodule
```

### Action

To eliminate the warning, remove the nesting and include as a separate (non-nested) comment.

```
module my_and (c,a,b);
 input a,b;
 output c;
 wire temp;
 /* initial block comment /*
 /* second block comment */
 assign temp =!b;
 assign c= a & temp;
endmodule
```

# CS231

## @E: Unknown macro <temp>

The referenced macro is unknown or undefined. In the example below, macro S1 is undefined (the macro is commented out in the example) which results in the error.

```
// Mealy machine to detect two consecutive 1's
module seq_det11 (input x, clk, reset, output reg y);
reg next_state, state /* synthesis syn_encoding="gray" */ ;
`define S0 1'd0
// `define S1 1'd1

/*************************/
always @(state or x)
begin
 case (state)
 `S0 :
 begin
 if(x)
 begin
 next_state <= `S1;
 y <= 1'b0;
 end
 else
 begin
 next_state <= `S0;
 y <= 1'b0;
 end
 end
 `S1 :
 begin
 if(x)
 begin
 next_state <= `S1;
 y <= 1'b0;
 end
 else
 begin
 next_state <= `S0;
 end
 end
 endcase
end
```

```
 y <= 1'b0;
 end
end
endcase
end

/**/
always @(posedge clk or posedge reset)
begin
 if(reset)
 state <= `S0;
 else
 state <= next_state;
end

/**/
endmodule
```

## Action

Make sure that the referenced macro is defined in the RTL source code. Or, instead of modifying the macro definition directly in the source, you can override the RTL setting with an option added either in the project file or the GUI:

- Add the `-hdl_define` option to the project file to override the RTL definition and define a macro. For the above example, include the following option to correct the error without having to modify the source code:  
  
`set_option -hdl_define -set "S1=1"`
- Add the `-hdl_define` entry to the Compiler Directives field on the Verilog tab of the Implementation Options dialog box. Note that when using this method, the macro definition is temporary until the project is saved.

## CS234

### @E: expecting identifier immediately following back-quote (`)

Verilog language requires that all directives are defined by using `keyword. However, if the back-quote (`) is not immediately followed by an identifier (define, ifdef, timescale, include, etc.) in the source code, the compiler errors out. Here is an example that results in the above error.

```
// Mealy machine to detect two consecutive 1's
module seq_det11 (input x, clk, reset, output reg y);
reg next_state, state /* synthesis syn_encoding="gray" */ ;
` define S0 1'd0
` define S1 1'd1

/*****************/
always @(state or x)
begin
 case (state)
 `S0 :
 begin
 if(x)
 begin
 next_state <= `S1;
 y <= 1'b0;
 end
 else
 begin
 next_state <= `S0;
 y <= 1'b0;
 end
 end
 `S1 :
 begin
 if(x)
 begin
 next_state <= `S1;
 y <= 1'b0;
 end
 else
 begin
 next_state <= `S0;
```

```

 y <= 1'b0;
 end
end
endcase
end

/**/
always @(posedge clk or posedge reset)
begin
 if(reset)
 state <= `S0;
 else
 state <= next_state;
end

/**/
endmodule

```

## Action

Do not leave a space after the back-quote (`). In the above example, modifying the code to remove the spaces eliminates the error.

```

`define S0 1'd0
`define S1 1'd1

```

# CS236

### **@E: Expecting quoted file name**

The compiler encountered an `include statement without double quotes around the filename (`include statements are used to include the contents of Verilog files). The filename can be specified with either a full path name or a relative path, but must be enclosed in double quotes. In the following test case, the double quotes are omitted which results in the error.

```

`include adder.v
module adder16 (cout, sum, a, b, cin);
output cout;
/* We are also using a parameter at this level of hierarchy */
parameter my_size = 16; // We want a 16-bit adder
output [my_size - 1: 0] sum;

```

```
input [my_size - 1: 0] a, b;
input cin;
/* my_size overwrites size inside instance my_adder of adder */
adder #(my_size) my_adder (cout, sum, a, b, cin);
endmodule

//contents of adder.v
//module adder (cout, sum, a, b, cin);
//parameter size = 1; /* declare a parameter. Default required */
//output cout;
//output [size-1:0] sum; /* sum uses the size parameter */
//input cin;
//input [size-1:0] a, b; /* 'a' and 'b' use the size parameter */
//assign {cout, sum} = a + b + cin;
//endmodule
```

## Action

Make sure that the file name specified by the `include statement is in double quotes. To eliminate the error in the above test case, enclose the included filename in double quotes as shown in the corrected test case below.

```
`include "adder.v"
module adder16 (cout, sum, a, b, cin);
output cout;
/* We are also using a parameter at this level of hierarchy */
parameter my_size = 16; // We want a 16-bit adder
output [my_size - 1: 0] sum;
input [my_size - 1: 0] a, b;
input cin;
/* my_size overwrites size inside instance my_adder of adder */
adder #(my_size) my_adder (cout, sum, a, b, cin);
endmodule
```

# CS243

## **@E: Direction (input, output, inout) set multiple times for port <b>**

A module has multiple port declarations for the same port. In the following test case, port b is declared multiple times which results in the error.

```
// Comparator
module compare(equal, a, output b);
parameter size = 1;
output equal;
input [size-1:0] a, b; // declare inputs
assign equal = a == b;
endmodule
```

## Action

In the above test case, port b is used as an input which conflicts with its declaration as an output in the port list. Make sure that ports are declared only once with the correct mode. To eliminate the error, remove the output declaration in the port list as shown in the corrected test case below.

```
// Comparator
module compare(equal, a, b);
parameter size = 1;
output equal;
input [size-1:0] a, b; // declare inputs
assign equal = a == b;
endmodule
```

# CS266

## **@E: Failure while reading device library file <fileName>**

The structural Verilog compiler could not read the device library file. Device library files contain black box definitions of technology primitives in the selected technology.

## Action

You can specify all structural Verilog files as Verilog files instead and run the design again.

# CS267

## @E: defparam: Unhandled assignment for <parameterName>

The structural Verilog compiler could not evaluate the constant value in the RHS for defparam.

```
module sub (
 in1,
 in2,
 in3,
 in4,
 out1
)
;
input in1;
input in2;
input in3;
input in4;
output out1;
wire [15:0]temp = 16'h8000;

LUT4 out1_c (
 .I0(in1),
 .I1(in2),
 .I2(in3),
 .I3(in4),
 .O(out1)
);
defparam out1_c.INIT=temp;
endmodule
```

### Action

Check that the RHS defparam assignments are correct. Assignments are corrected as shown below.

```
module sub (
 in1,
 in2,
 in3,
 in4,
 out1
)
```

```
;
input in1;
input in2;
input in3;
input in4;
output out1;
wire [15:0]temp = 16'h8000;

LUT4 out1_c (
 .I0(in1),
 .I1(in2),
 .I2(in3),
 .I3(in4),
 .O(out1)
);
defparam out1_c.INIT=16'h8000;
endmodule
```

## CS268

### **@E: Unsupported assignment to net <netName>**

The structural Verilog compiler only supports simple assignments. In this test case, the compiler encountered a complex assignment for net out2.

```
module sub (
 in1,
 in2,
 in3,
 in4,
 in5,
 out1,
 out2,
 in6
)
;
input in1;
input in2;
input in3;
input in4;
output out1;
input [2:0]in5;
```

```
output out2;
input in6;

assign out2 = in5[in6];

LUT4 out1_c (
 .I0(in1),
 .I1(in2),
 .I2(in3),
 .I3(in4),
 .O(out1)
);
defparam out1_c.INIT=16'h8000;
endmodule
```

## Action

You can specify the structural Verilog file containing the complex assignment as a Verilog file and run the design again.



## CHAPTER 20

# DE Messages 100 – 110

---

All the messages described in this section can be downgraded from an error to a warning. By definition, an error stops the operation and requires the condition to be corrected and the operation repeated. Downgrading an error to a warning allows the operation to be continued beyond the reported condition. At the risk of producing incorrect silicon, the consequences of downgrading an error to a warning must be fully understood.

## DE102

### **@E: The high number of multi-hop paths found in this design can result in excessive runtimes during time budgeting.**

The above error is reported during partitioning when an internal evaluation determines that the number of recursive multi-hop paths in the design would result in an excessive amount of time required to perform the time-budget calculation.

#### Action

To address the error, make sure that the -effort level for the run partition command is set to low or medium and make sure that the multi-hop\_accuracy option is set to 0. If the error persists, try loosening the partitioning constraints.

Downgrading this error to a warning allows the time-budget calculation to continue to determine if a solution is ultimately possible.

## DE103

### **@E: Distributed TMR view <viewName> includes user-instantiated technology primitive <primitiveName> that cannot be scrubbed**

A module containing user-instantiated technology primitives has been marked for distributed TMR.

When using distributed TMR, voting logic is added on feedback paths to avoid the accumulation of errors introduced by an SEU (Single Event Upset). When a design includes user-instantiated technology primitives, the tool does not add voting logic to the feedback paths through these primitives which can result in improper design behavior.

Under certain conditions, this error can be downgraded to a warning which will allow the operation to continue in the presence of user-instantiated primitives.

### Action

To address the error, remove any user-instantiated technology primitives from the module.

Downgrading this error to a warning reports the user-instantiated technology primitives in the design and allows the operation to continue. The user is responsible for avoiding any feedback loops on paths that include user-instantiated technology primitives.

## DE108

### **@E: Safe state machine method specified on instance <name> is not applicable for this technology**

This error is displayed because the technology family includes TMR on registers and therefore does not support safe FSM specified through attributes or options.

### Action

Do not implement safe FSMs with any of the methods below, as it will cause the error message to display:

- Setting the `syn_encoding = safe` or `syn_safe_case` attributes
- Enabling these options on the High Reliability tab of the Implementation Options dialog box: Preserve and Decode Unreachable States, FSM Error Correction using Hamming Distance 3, FSM-DED and Recovery using Hamming Distance 3

Alternatively, use the `message_override -warning DE108` command to downgrade the error to a warning, continue with synthesis, and implement safe FSMs based on TMR registers.

## DE110

**@E: Subsystem <*subsystem*> did not link correctly. See linker.rpt for details.**

A module failed to link. This may happen when a port of the module was added, removed, or renamed, so copies of the block are no longer consistent.

### Action

Address the consistency issue or downgrade this message to a warning.

## CHAPTER 21

# FF Messages 100 – 112

---

## FF101

**@W: DFPC used: design must not allow both PRE and CLR to be asserted at the same time.**

A design that includes a DFPC primitive (D-flip-flop with both asynchronous preset and clear) indicated that for the selected technology that the DFPC preset and clear have equal priority and that if asserted simultaneously can give unpredictable results. The primitive is inferred with the above message.

```
module ff_async (d,clk,set,rst, q);
 input clk,d,set,rst;
 output reg q;

 always@(posedge clk or posedge rst or posedge set)
 if(rst)
 q<= 1'b0;
 else if(set)
 q<=1'b1;
 else if(clk)
 q<=d;
endmodule
```

## Action

If you are confident that both the preset and clear cannot be asserted simultaneously, you can ignore the above warning. If simultaneous assertion is possible, change the coding to infer a different primitive. In the modified test case below, the set condition is synchronous to the clock which infers a DFC1 primitive without the ambiguity.

```
module ff_async (d,clk,set,rst, q);
 input clk,d,set,rst;
 output reg q;

 always@(posedge clk or posedge rst)
 if(rst)
 q<= 1'b0;
 else if(clk)
 begin
 if (set)
 q<=1'b1;
 else
 q<=d;
 end
 endmodule
```

## FF102

### **@W: DFPCA used: design must not allow both PRE and CLR to be asserted at the same time**

A design that includes a DFPCA primitive (D-flip-flop with active high preset and active low clear) indicated that for the selected technology that the DFPCA preset and clear have equal priority and that if asserted simultaneously can give unpredictable results. The primitive is inferred with the above message.

```
module ff_async (d,clk,set,rst, q);
 input clk,d,set,rst;
 output reg q;
```

```
always@(negedge clk or negedge rst or posedge set)
 if(!rst)
 q<= 1'b0;
 else if(set)
 q<=1'b1;
 else if(!clk)
 q<=d;
endmodule
```

## Action

If you are confident that both the preset and clear cannot be asserted simultaneously, you can ignore the above warning. If simultaneous assertion is possible, change the coding to infer a different primitive. In the modified test case below, the set condition is synchronous to the clock which infers a DFC1D primitive without the ambiguity.

```
module ff_async (d,clk,set,rst, q);
 input clk,d,set,rst;
 output reg q;

 always@(negedge clk or negedge rst)
 if(!rst)
 q<= 1'b0;
 else if(!clk)
 begin
 if (set)
 q<=1'b1;
 else
 q<=d;
 end
 endmodule
```

# FF106

## @E: Port count mismatch between black-box/tech primitive <AND2>.

A technology primitive was declared as a black box and the port count does not match the original technology primitive definition. In the example below, a 2-input AND2 primitive is used. The black-box definition for the primitive in the code has three inputs and one output which causes the mapper to error out with the above message.

```
module AND2(Y,A,B,C)/* synthesis syn_black_box*/;
output Y;
input A;
input B,C;
endmodule

module top (in1,in2,in3,out);
output out;
input in1,in2,in3;
AND2 u1 (out,in1,in2,in3);
endmodule
```

### Action

Either make sure that the primitive definition has the same port count as the original technology primitive or change the black-box name if a different primitive implementation is used. In the corrected test case below, the primitive definition in the code is modified to match the original technology primitive definition.

```
module AND2(Y,A,B)/* synthesis syn_black_box*/;
output Y;
input A,B;
endmodule

module top (in1,in2,out);
output out;
input in1,in2;
AND2 u1 (out,in1,in2);
endmodule
```

## FF112

### **@W: Unable to map all tristates. Make sure there are usable tristate cells in the library**

A design includes tristate drivers, but the library/device selected had no available tristate cells (the tristates in the design cannot be mapped and will cause place and route to fail). Running the following test case results in the above message.

```
module noiotri(a, b, c, d, e, f);
 output [2:0] f;
 inout [3:0] e;
 input [2:0] a, b, c, d;
 assign e = {a[2], (a & b)} ? {c[2], (c & d)} : 4'bz;
 assign f = e[2:0] & d;
endmodule
```

### Action

Make sure that either the design has no tristates if the device library has no tristates or use a library that supports tristates.



## CHAPTER 22

# FL Messages 100 – 105

---

## FL104

### **@E: Port name <A> mismatch on primitive <GCLKBUF>**

A primitive was instantiated and there was a port name mismatch between the standard primitive definition and the port names defined in design. In the test case below, primitive GCLKBUF is instantiated with port names a and b. Because these names do not match the standard primitive port names (ci and o), the mapper errors out with the above message.

```
module test_ff (q, clk, din, en, rst);
 output [1:0] q ;
 input [1:0]din;
 input clk;
 input en,rst;
 reg [1:0] q;
 wire temp;
 GCLKBUF (clk,temp);

 always @(posedge temp or posedge rst)
 if(rst)
 q<=1'b0;
 else
 if(en)
 q<=din;
endmodule
```

```
module GCLKBUF (input a, output b) /*synthesis syn_black_box */
*/;
endmodule
```

## Action

If a standard primitive is to be instantiated in a design, the port definitions in the design must match the standard primitive names. In the corrected test case below, the standard GCLKBUF primitive port names (ci and o) are used to eliminate the error.

```
module test_ff (q,clk,din, en, rst);
output [1:0] q ;
input [1:0]din;
input clk;
input en,rst;
reg [1:0] q;
wire temp;
GCLKBUF (clk,temp);

always @(posedge temp or posedge rst)
 if(rst)
 q<=1'b0;
 else
 if(en)
 q<=din;
endmodule

module GCLKBUF (input ci, output o) /*synthesis syn_black_box */;
endmodule
```

# FL105

## **@E: Set-reset primitive not supported for instance <q>.**

An asynchronous set/reset primitive was encountered in a design and the device vendor does not support functional primitives with asynchronous set/reset functionality. In the test case below, a flip-flop is implemented with a set/reset primitive which causes the mapper to error out with the above message.

```
module d_ff_async (d,clk,set,rst,q);
input clk,d,set,rst;
output reg q;
always@(posedge clk or posedge rst or posedge set)
 if(rst)
 q<= 1'b0;
 else if (set)
 q<=1'b1;
 else
 q<=d;
endmodule
```

## Action

The HDL code must be modified to correspond to the targeting architecture (avoid describing a register in the HDL code that is not supported). In the corrected test case below, the asynchronous reset is maintained, and the set operation is made synchronous to the clock to eliminate the error.

```
module d_ff_async (d,clk,set,rst,q);
input clk,d,set,rst;
output reg q;
always@(posedge clk or posedge rst) // or posedge set
 if(rst)
 q<= 1'b0;
 else if (clk)
 if(set)
 q<=1'b1;
 else
 q<=d;
endmodule
```



## CHAPTER 23

# FO Messages 100 – 126

---

## FO100

### **@N: Reset <1>: <"RST0" in work.gsr\_test(verilog)>**

By default, if there is a single reset/preset used in the design, the mapper connects that reset/preset signal to a GSR instance. When there are multiple reset/preset signals, the mapper cannot determine which signal to connect to the GSR instance and generates the above note indicating the number of reset/preset signals found and to which module they belong. For the test case below, there are two resets in the design and two notes are displayed.

```
`define width 4'd4
module gsr_test (CLK, RST0, RST1, a, q);
 input CLK, RST0, RST1;
 input [`width-1:0] a;
 output [`width -1:0] q;
 reg [`width -1:0] q,temp;

 always @(posedge CLK or posedge RST0)
 begin
 if (RST0)
 temp <= 4'b0110;
 else
 temp <= a;
 end
```

```
always @(posedge CLK or negedge RST1)
begin
 if (!RST1)
 q <= 4'b0110;
 else
 q <= temp;
end
endmodule
```

## Action

When you want a particular reset/preset signal to use the GSR instance, instantiate the GSR instance in the HDL code. In the modified test case below, the mapper uses the GSR instance for reset signal RST0.

```
`define width 4'd4
module gsr_test (CLK, RST0, RST1, a, q);
 input CLK, RST0, RST1;
 input [`width-1:0] a;
 output [`width -1:0] q;
 reg [`width -1:0] q,temp;
 GSR u1(RST0);

 always @(posedge CLK or posedge RST0)
 begin
 if (RST0)
 temp <= 4'b0110;
 else
 temp <= a;
 end

 always @(posedge CLK or negedge RST1)
 begin
 if (!RST1)
 q <= 4'b0110;
 else
 q <= temp;
 end
endmodule
```

## FO104

### @E: Found multiple GSR instances <work.top(verilog)-u2> and <work.top(verilog)-u1>

The Global Set/Reset (GSR) available in some device technologies, is used when there is only set/reset. When a design includes more than one set/reset, the mapper does not use GSR. When GSR is instantiated more than once in a design, the mapper errors out with the above message. In the test case below, two GSR primitives are instantiated which results in the above error.

```
`define width 4'd4
module top (CLK, RST0, RST1, a, q);
 input CLK, RST0, RST1;
 input [`width-1:0] a;
 output [`width -1:0] q;
 reg [`width -1:0] q,temp;
 GSR u1 (RST0);
 GSR u2 (RST1);

 always @(posedge CLK or negedge RST0)
 begin
 if (!RST0)
 temp <= 4'b0110;
 else
 temp <= a;
 end

 always @(posedge CLK or negedge RST1)
 begin
 if (!RST1)
 q <= 4'b0110;
 else
 q <= temp;
 end
endmodule

module GSR(GSR); // synthesis syn_black_box syn_noprune=1
 input GSR;
endmodule
```

## Action

To eliminate in the presence of multiple set/reset signals, use the GSR for the global set/reset by instantiating it only once as shown in the corrected test case below.

```
`define width 4'd4
module top (CLK, RST0, RST1, a, q);
 input CLK, RST0, RST1;
 input [`width-1:0] a;
 output [`width -1:0] q;
 reg [`width -1:0] q,temp;
 GSR u2 (RST1);

 always @(posedge CLK or negedge RST0)
 begin
 if (!RST0)
 temp <= 4'b0110;
 else
 temp <= a;
 end

 always @(posedge CLK or negedge RST1)
 begin
 if (!RST1)
 q <= 4'b0110;
 else
 q <= temp;
 end
endmodule

module GSR(GSR); // synthesis syn_black_box syn_noprune=1
 input GSR;
endmodule
```

## FO105

**@N: Propagated <1> black\_box\_pad\_pin properties upward to ports of <top>.**

A black\_box\_pad\_pin property was applied on a black-box I/O pin and it was successfully instantiated. The note also indicates that the corresponding top-level port will not have any pads. In the following test case, output pin q of the black box is assigned the directive black\_box\_pad\_pin, and the black box is instantiated in top module top as indicated in the above message.

```
module test (d,clk,q)
 /* synthesis syn_black_box black_box_pad_pin = "q" */
 input d,clk;
 output q;
endmodule

module top (d1,clk1,q1);
 input d1,clk1;
 output q1;
 test u1 (.d(d1),.clk(clk1),.q(q1));
endmodule
```

### Action

Informative message; no action required.

## FO109

**@W: Instantiated black-box <II\_0> () is in ORCA3 library, not ORCA4E library.**

A library component from one library was used in a design targeted for a device from another library. In the test case below, black box CLKCNTHB\_PUR, which is unique to one library, is instantiated in a design based on another library which results in the warning message.

```
module CLKCNTHB_PUR_top (CLKIN, SHUTOFF, PUR, CLKOUT);
 input CLKIN;
 input SHUTOFF;
 input PUR;
 output CLKOUT;
 CLKCNTHB_PUR(CLKIN, SHUTOFF, PUR, CLKOUT);
endmodule

module CLKCNTHB_PUR(CLKIN, SHUTOFF, PUR, CLKOUT);
 input CLKIN;
 input SHUTOFF;
 input PUR;
 output CLKOUT;
endmodule
```

## Action

Use device-specific library components in the design. If any other component is used, its functionality must be defined explicitly.

# FO110

## **@W: Instantiated black-box <u1> () is in ORCA4 library, not ORCA3 library.**

A library component from one library was used in a design targeted for a device from another library. In the test case below, black box MUX81, which is unique to one library, is instantiated in a design based on another library which results in the warning message.

```
module MUX81_top (D0, D1, D2, D3, D4, D5, D6, D7, SD1, SD2, SD3,
 Z);
 input D0, D1, D2, D3, D4, D5, D6, D7, SD1, SD2, SD3;
 output Z;
 MUX81 u1 (D0, D1, D2, D3, D4, D5, D6, D7, SD1, SD2, SD3, Z);
endmodule

module MUX81 (D0, D1, D2, D3, D4, D5, D6, D7, SD1, SD2, SD3, Z);
 input D0, D1, D2, D3, D4, D5, D6, D7, SD1, SD2, SD3;
 output Z;
endmodule
```

### Action

Use device-specific library components in the design. If any other component is used, its functionality must be defined explicitly.

## FO111

### **@W: Cannot propagate black\_box\_pad\_pin property from pin <u1/q>. Net should connect only to ports.**

The `black_box_pad_pin` directive specifies pins on a user-defined black-box component as I/O pads. The net connecting a black-box pin and a top-level port was also used in an intermediate operation. In the following test case, output port `q` of the black box is passed as `black_box_pad_pin` to the top-level module and is also assigned to AND gate input which results in the warning message.

```
module test (d,clk,q)
 /* synthesis syn_black_box black_box_pad_pin = "q" */
 input d,clk;
 output q;
endmodule

module top (d1,in1,clk1,q1,q2);
 input d1,clk1,in1;
 output q1,q2;
 test u1 (.d(d1),.clk(clk1),.q(q1));
 assign q2 = in1 & q1;
endmodule
```

### Action

Make sure that any black-box pins defined by the `black_box_pad_pin` directive only drive top-level ports.

```
module test (d,clk,q)
 /* synthesis syn_black_box black_box_pad_pin = "q" */
 input d,clk;
 output q;
endmodule
```

```
module top (d1,in1,clk1,q1,q2);
 input d1,clk1,in1;
 output q1,q2;
 test u1 (.d(d1),.clk(clk1),.q(q1));
 assign q2 = in1 & d1;
endmodule
```

## FO112

### **@W: Cannot propagate black\_box\_pad\_pin property from pin <u1/q>. Net must connect only to ports.**

The `black_box_pad_pin` directive specifies pins on a user-defined black-box component as I/O pads. The above warning occurs when the net connecting a black-box pin and a top-level port is also used in an intermediate operation. In the following test case, output port `q` of the black box is passed as `black_box_pad_pin` to the top-level module and is also assigned to AND gate input which results in the warning message.

```
module test (d,clk,q)
 /* synthesis syn_black_box black_box_pad_pin = "q" */ ;
 input d,clk;
 output q;
endmodule

module top (d1,clk1,q1);
 input d1,clk1;
 wire temp;
 output q1;
 test u1 (.d(d1),.clk(clk1),.q(temp));
 assign q1 = d1 & temp;
endmodule
```

### Action

Make sure that any black-box pins defined by the `black_box_pad_pin` directive only drive top-level ports.

```
module test (d,clk,q)
 /* synthesis syn_black_box black_box_pad_pin = "q" */
 input d,clk;
 output q;
endmodule

module top (d1,clk1,q1);
 input d1,clk1;
 output q1;
 test u1 (.d(d1),.clk(clk1),.q(q1));
endmodule
```

## FO114

### **@E: Could not bind `black_box` to library component.**

The mapper cannot bind a black box because its name is the same as a technology primitive.

The port name of the standard GSR technology primitive is GSR. In the test case below, however, the port name for the black-box GSR is rst which causes the mapper to error out with the above message.

The GSR technology primitive is used for the following test case.

```
`define width 4'd4
module top (CLK, RST0, RST1, a, q);
 input CLK, RST0, RST1;
 input [`width-1:0] a;
 output [`width -1:0] q;
 reg [`width -1:0] q,temp;
 GSR u1 (RST0);

 always @(posedge CLK or negedge RST0)
 begin
 if (!RST0)
 temp <= 4'b0110;
 else
 temp <= a;
 end
```

```

always @(posedge CLK or negedge RST1)
begin
if (!RST1)
 q <= 4'b0110;
else
 q <= temp;
end
endmodule

module GSR(rst); // synthesis syn_black_box syn_noprune=1
input rst;
endmodule

```

## Action

Make sure that the port name of the standard technology primitive and the black-box name used in the design always match. To eliminate the error in the above test case, use standard port name GSR in the port definition of module GSR.

```

`define width 4'd4
module top (CLK, RST0, RST1, a, q);
input CLK, RST0, RST1;
input [`width-1:0] a;
output [`width -1:0] q;
reg [`width -1:0] q,temp;
GSR u1 (RST0);

always @(posedge CLK or negedge RST0)
begin
if (!RST0)
 temp <= 4'b0110;
else
 temp <= a;
end

always @(posedge CLK or negedge RST1)
begin
if (!RST1)
 q <= 4'b0110;
else
 q <= temp;
end
endmodule

```

```
module GSR(GSR); // synthesis syn_black_box syn_noprune=1
 input GSR;
endmodule
```

## FO117

### **@N: syn\_useioff discarded: cannot embed counter in output pads.**

A counter defined in the HDL code also has an associated syn\_useioff attribute set to pack the output registers in the I/O flip-flops. Because the I/O blocks do not include the hardware required to implement the counter logic, the syn\_useioff attribute is ignored.

In the test case below, the outputs of counters out and cout are forced to go into I/O blocks by the syn\_useioff attributes which results in two of the above notes.

```
module counter2(out, cout, data, load, cin, clk);
 input [7:0] data;
 input load;
 input cin, clk;
 output reg [7:0] out /* synthesis syn_useioff = 1 */;
 output reg cout /* synthesis syn_useioff = 1 */;
 reg [7:0] preout;

 always @ (posedge clk)
 begin
 out = preout;
 end

 always @(out or data or load or cin)
 begin
 {cout, preout} = out + cin;
 if (load) preout = data;
 end

endmodule
```

## Action

The registers of a counter cannot be packed into I/O blocks; the `syn_useioff` attributes must be set to 0 as shown in the modified test case below or simply deleted.

```
module counter2(out, cout, data, load, cin, clk);
 input [7:0] data;
 input load;
 input cin, clk;
 output reg [7:0] out /* synthesis syn_useioff = 0 */;
 output reg cout /* synthesis syn_useioff = 0 */;
 reg [7:0] preout;

 always @ (posedge clk)
 begin
 out = preout;
 end

 always @(out or data or load or cin)
 begin
 {cout, preout} = out + cin;
 if (load) preout = data;
 end

endmodule
```

## FO120

**@W: Attribute <`DELAYMODE`> cannot be set on instance <`q_pad`>, only on pads of type : <`Input`>**

An attribute property was specified on a pad type for which it was not applicable. In the test case below, the `orca_props` attribute sets property `DELAYMODE` to 0 for output port `q`. However, because the `DELAYMODE` property is legal only for ports of type `input`, the mapper issues the above warning.

```
module test_orca_props (q,clk,din, en, rst);
 output q /* synthesis orca_props = "DELAYMODE=0" */;
 input clk,din,en,rst;
 reg q;
```

---

```

always @(posedge clk or posedge rst)
if(rst)
 q<=1'b0;
else
 if(en)
 q<=din;
endmodule

```

## Action

Make sure that the property being applied on a port is legal for that port type. For the legal properties of an attribute and their values, please refer online help. In the modified test case below, the property is correctly applied on input port din.

```

module test_orca_props (q,clk,din, en, rst);
output q;
input clk,en,rst;
input din /* synthesis orca_props = "DELAYMODE=0" */;
reg q;

always @(posedge clk or posedge rst)
if(rst)
 q<=1'b0;
else
 if(en)
 q<=din;
endmodule

```

## FO122

**@W: Attribute <AMPSMODE> must be paired with another attribute like : <LEVELMODE> = <LVCMOS2>, it will be ignored in Place and Route**

Some attributes require a property to be specified in conjunction with other properties and, when such a property is specified without its associated properties, the mapper issues the above warning. For example, the AMPSMODE property of the orca\_props attribute must be set in conjunction

with the LEVELMODE property set to either LVTTL or LVCMOS2. In the test case below, the orca\_props attribute only sets AMPSMODE and, because LEVELMODE is not set, the mapper issues the above warning.

```
module test_orca_props (q,clk,din, en, rst);
output q /* synthesis orca_props = "AMPSMODE=24" */;
input clk,en,rst;
input din ;
reg q;

always @(posedge clk or posedge rst)
if(rst)
 q<=1'b0;
else
 if(en)
 q<=din;
endmodule
```

## Action

Make sure that any attribute required in conjunction with other properties or attributes is specified. In the corrected test case below, property AMPSMODE is set in conjunction with LEVELMODE set to LVTTL to eliminate the warning.

```
module test_orca_props (q,clk,din, en, rst);
output q /* synthesis orca_props = "AMPSMODE=24,LEVELMODE=LVTTL"
*/;
input clk,en,rst;
input din ;
reg q;

always @(posedge clk or posedge rst)
if(rst)
 q<=1'b0;
else
 if(en)
 q<=din;
endmodule
```

# FO126

## @N: Generating RAM <mem[4:0]>

The mapper has inferred a RAM. Running the following test case on a device from the MachXO library results in the above message.

```
module ram (d,clk,we,waddr,raddr);
 input clk,we;
 inout [4:0] d;
 input [9:0] waddr,raddr;
 reg [9:0] reg_raddr;
 reg [4:0] mem [1023:0];

 always@(posedge clk)
 begin
 reg_raddr <= raddr;
 if(we)
 mem[waddr] <= d;
 end

 assign d= we?5'bzz: mem[reg_raddr];
endmodule
```

### Action

Informative message; no user action required.



## CHAPTER 24

# FP Messages 100 – 139

---

## FP101

**@W: The design has <9> instantiated global buffers but allowed is only <6>**

More than an allowed number of global buffers were instantiated in a design. By default, the maximum number of buffers that can be instantiated is six. This number can be increased to a maximum of 18 using the syn\_global\_buffers attribute. In the test case below, nine CLKBUFFs are instantiated which exceeds the default and results in the warning message.

```
module ff_rs (d,rst,clk,q);
parameter k =8;
input [k:0] d;
input rst ;
input clk;
output reg [k:0] q;
wire [k:0] d_buf;
CLKBUF n1 [k:0] (d_buf,d);

always @(posedge clk)
if (rst)
 q <= 1'b0;
else if(clk)
 q <= d_buf;
endmodule
```

## Action

The number of instantiations must be either 6 (the default) or no more than 18 (specified with the `syn_global_buffers` attribute). In the corrected test case below, the `syn_global_buffers` attribute is set to 10

```
module ff_rs (d,rst,clk,q) /* synthesis syn_global_buffers = 10 */
/*
parameter k =8;
input [k:0] d;
input rst ;
input clk;
output reg [k:0] q;
wire [k:0] d_buf;
CLKBUF n1[k:0] (d ,d_buf);

always @(posedge clk)
 if (rst)
 q<=1'b0;
 else if(clk)
 q<= d_buf;
endmodule
```

## FP103

### **@W: User can use `syn_global_buffers` to increase the allowed global clock buffers to maximum 18**

More than an allowed number of global buffers were instantiated in a design. You can increase the number of global buffers available with the `syn_global_buffers` attribute; the maximum number is 18 and the default is 6. For the test case below, 9 global buffers are instantiated but because only 6 are available (the default), the mapper displays the above message.

```
module ff_rs (d,rst,clk,q);
parameter k =8;
input [k:0] d;
input rst ;
input clk;
output reg [k:0] q;
wire [k:0] d_buf;
CLKBUF n1 [k:0] (d_buf,d);
```

```
always @(posedge clk)
 if (rst)
 q <= 1'b0;
 else if(clk)
 q <= d_buf;
endmodule
```

## Action

The number of global buffer instantiations must be either 6 (the default) or no more than 18 (specified with the `syn_global_buffers` attribute). In the corrected test case below, the `syn_global_buffers` attribute is set to 10.

```
module ff_rs (d,rst,clk,q) /* synthesis syn_global_buffers = 10
*/;
parameter k =8;
input [k:0] d;
input rst ;
input clk;
output reg [k:0] q;
wire [k:0] d_buf;
CLKBUF n1[k:0] (d ,d_buf);

always @(posedge clk)
 if (rst)
 q <= 1'b0;
 else if(clk)
 q <= d_buf;
endmodule
```

# FP109

## **@W: Inferring tristate/bidirectional I/O buffer, with I/O insertion disabled, to compensate for device without internal tristate cell.**

The above warning occurs when:

- the RTL code includes internal tristates
- I/O insertion is disabled
- the specified device does not provide an internal tristate cell

When the above conditions occur, the mapper infers a tristate/bidirectional I/O buffer). Note that I/O insertion occurs only on the tristated ports. The test case below includes two tristates which, when I/O insertion is disabled, results in the above message.

```
##Project options##
set_option -disable_io_insertion 1
```

## Test Case

```
module test (input a,b,x,y,p,q,output out);
wire and_out,or_out;
wire en1,en2;
assign and_out = a & b;
assign or_out = a | b;
assign en1 = x | y;
assign en2 = p & q;
assign temp = en1 ? and_out : 1'bz;
assign temp = en2 ? or_out : 1'bz;
assign out = temp ;
endmodule
```

## Action

You can ignore the warning if expected or, if no I/O buffers are to be inserted, make sure that there are no internal tristates when I/O insertion is disabled.

# FP112

## **@W: User specified syn\_global\_buffers exceeds the allowed global limit of <6> for this part**

The value specified for the syn\_global\_buffers attribute exceeded the allowed global limit for the selected part. For the following test case, a device with a global buffer limit of 6 is used. The attribute value in the test case is set to 7 which results in the above message.

## Test Case

```
module top (clk1, clk2, clk3, clk4, clk5, clk6, clk7, d1, d2,
 d3, d4, d5, d6, d7, q1, q2, q3, q4, q5,q6, q7, reset)
 /* synthesis syn_global_buffers = 7 */;
 input clk1, clk2, clk3, clk4, clk5, clk6, clk7;
 input d1, d2, d3, d4, d5, d6, d7;
 output q1, q2, q3, q4, q5, q6, q7;
 input reset;
 reg q1, q2, q3, q4, q5, q6, q7;

 always @(posedge clk1 or posedge reset)
 if (reset)
 q1 <= 1'b0;
 else
 q1 <= d1;

 always @(posedge clk2 or posedge reset)
 if (reset)
 q2 <= 1'b0;
 else
 q2 <= d2;

 always @(posedge clk3 or posedge reset)
 if (reset)
 q3 <= 1'b0;
 else
 q3 <= d3;

 always @(posedge clk4 or posedge reset)
 if (reset)
 q4 <= 1'b0;
 else
 q4 <= d4;

 always @(posedge clk5 or posedge reset)
 if (reset)
 q5 <= 1'b0;
 else
 q5 <= d5;

 always @(posedge clk6 or posedge reset)
 if (reset)
 q6 <= 1'b0;
 else
 q6 <= d6;
```

---

```

always @(posedge clk7 or posedge reset)
 if (reset)
 q7 <= 1'b0;
 else
 q7 <= d7;

endmodule

```

## Action

Either make sure that the value specified by the `syn_global_buffers` attribute does not exceed the allowed global limit for the selected part as shown in the corrected test case below or select a part with more available global buffers.

```

module top (clk1, clk2, clk3, clk4, clk5, clk6, clk7, d1, d2, d3,
 d4, d5, d6, d7, q1, q2, q3, q4, q5, q6, q7, reset)
 /* synthesis syn_global_buffers =6 */;
 input clk1, clk2, clk3, clk4, clk5, clk6, clk7;
 input d1, d2, d3, d4, d5, d6, d7;
 output q1, q2, q3, q4, q5, q6, q7;
 input reset;
 reg q1, q2, q3, q4, q5, q6, q7;

 always @(posedge clk1 or posedge reset)
 if (reset)
 q1 <= 1'b0;
 else
 q1 <= d1;

 always @(posedge clk2 or posedge reset)
 if (reset)
 q2 <= 1'b0;
 else
 q2 <= d2;

 always @(posedge clk3 or posedge reset)
 if (reset)
 q3 <= 1'b0;
 else
 q3 <= d3;

 always @(posedge clk4 or posedge reset)
 if (reset)
 q4 <= 1'b0;
 else
 q4 <= d4;

```

```
always @(posedge clk5 or posedge reset)
 if (reset)
 q5 <= 1'b0;
 else
 q5 <= d5;

always @(posedge clk6 or posedge reset)
 if (reset)
 q6 <= 1'b0;
 else
 q6 <= d6;

always @(posedge clk1 or posedge reset)
 if (reset)
 q7 <= 1'b0;
 else
 q7 <= q1;

endmodule
```

## FP132

**@W: Unknown pad\_type <LVC MOS\_18> specified for design input pins. Supported types are LVTTL, LVC MOS\_33, LVC MOS\_25, PCI33**

An unknown pad type was specified on an input pin. In the test case below, an invalid pad type (LVC MOS\_18) is specified.

### Constraint File Contents

```

I/O Standards

define_io_standard -default_input
-delay_type input syn_pad_type {LVC MOS_18}
```

## Test Case

```
module encoder2(none_on, out2, out1, out0, h, g, f, e, d, c, b, a);
 input h, g, f, e, d, c, b, a;
 output none_on, out2, out1, out0;
 wire [3:0] outvec;

 assign outvec = h ? 4'b0111: g ? 4'b0110:
 f ? 4'b0101: e ? 4'b0100:
 d ? 4'b0011: c ? 4'b0010:
 b ? 4'b0001: a ? 4'b0000:
 4'b1000;
 assign none_on = outvec[3];
 assign out2 = outvec[2];
 assign out1 = outvec[1];
 assign out0 = outvec[0];
endmodule
```

## Action

Make sure that only I/O standard type LVTTL, LVCMOS\_25, LVCMOS\_33, or PCI33 is applied to an input port as shown in the constraint file entry below:

```
define_io_standard -default_input
 -delay_type input syn_pad_type {LVCMOS_25}
```

# FP133

**@W: Unknown pad\_type <LVCMOS\_18> specified for <work.encoder2(verilog)-h>. Supported types are LVTTL / LVCMOS\_33 / LVCMOS\_25**

An unknown pad type was specified on an I/O pin. In the test case below, an invalid pad type (LVCMOS\_18) is specified.

## Constraint File Contents

```

I/O Standards

define_io_standard {h} -delay_type input syn_pad_type
{LVCMOS_18}
```

## Test Case

```
module encoder2(none_on, out2, out1, out0, h, g, f, e, d, c, b, a);
 input h, g, f, e, d, c, b, a;
 output none_on, out2, out1, out0;
 wire [3:0] outvec;

 assign outvec = h ? 4'b0111: g ? 4'b0110:
 f ? 4'b0101: e ? 4'b0100:
 d ? 4'b0011: c ? 4'b0010:
 b ? 4'b0001: a ? 4'b0000:
 4'b1000;
 assign none_on = outvec[3];
 assign out2 = outvec[2];
 assign out1 = outvec[1];
 assign out0 = outvec[0];

endmodule
```

## Action

Make sure that only I/O standard type LVTTL, LVCMOS\_25, or LVCMOS33 is applied to an I/O port as shown in the constraint file entry below:

```
define_io_standard {h} -delay_type input syn_pad_type {LVCMOS_25}
```

## FP134

**@W: Unknown pad\_type <LVC MOS\_18> specified for bidirectional pins. Supported types are LVTTL, LVC MOS\_33, LVC MOS\_25, PCI33**

An unknown pad type was specified on a bidirectional pin. In the test case below, an invalid pad type (LVC MOS\_18) is specified.

Constraint File Content:

```
define_io_standard -default_bidir
-delay_type bidir syn_pad_type {LVC MOS_18}
```

Test Case

```
module test (d,waddr,raddr,clk,en);
 input clk,en;
 input [3:0] waddr,raddr;
 inout [1:0]d;
 reg [1:0] mem[15:0];

 always @(posedge clk)
 begin
 if (en)
 mem [waddr] <= d;
 end

 assign d = en ? 2'bZ : mem[raddr];
endmodule
```

Action

Make sure that only I/O standard type LVTTL, LVC MOS\_25, LVC MOS\_33, or PCI33 is applied to a bidirectional port as shown in the constraint file entry below:

```
define_io_standard {h} -delay_type input syn_pad_type {LVC MOS_25}
```

# FP135

**@W: Unknown pad\_type <LVC MOS\_18> specified for design output pins. Supported types are LVTTL, LVC MOS\_33, LVC MOS\_25, PCI33**

An unknown pad type was specified on an output pin. In the test case below, an invalid pad type (LVC MOS\_18) is specified.

Constraint File Content:

```
define_io_standard -default_output
-delay_type output syn_pad_type {LVC MOS_18}
```

Test Case

```
module encoder2(none_on, out2, out1, out0, h, g, f, e, d, c, b, a);
output h, g, f, e, d, c, b, a;
output none_on, out2, out1, out0;
wire [3:0] outvec;
assign outvec = h ? 4'b0111: g ? 4'b0110:
f ? 4'b0101: e ? 4'b0100:
d ? 4'b0011: c ? 4'b0010:
b ? 4'b0001: a ? 4'b0000:
4'b1000;
assign none_on = outvec[3];
assign out2 = outvec[2];
assign out1 = outvec[1];
assign out0 = outvec[0];
endmodule
```

Action

Make sure that only I/O standard type LV TTL, LVC MOS\_25, or LVC MOS33 is applied to an output port as shown in the constraint file entry below:

```
define_io_standard -default_output
-delay_type output syn_pad_type {LVC MOS_25}
```

# FP139

## @E: Number of instantiated global buffers exceeds allowed global clock buffers

The number of global buffers required exceeded the allowed number of global clock buffers available for the device (default 6).

```
module ff_rs (d,rst,clk,q);
parameter k =7;
input [k:0] d;
input rst ;
input clk;
output reg [k:0] q;
wire [k:0] d_buf;
CLKBUF n1 [k:0] (d_buf,d);

always @(posedge clk)
 if (rst)
 q <= 1'b0;
 else if(clk)
 q <= d_buf;
endmodule
```

### Action

Make sure that the number of global buffers required does not exceed the default number of global clock buffers available or, as shown in the corrected case below, increasing the number of designated global buffers using the `syn_global_buffers` attribute.

```
module ff_rs (d,rst,clk,q) /* synthesis syn_global_buffers = 8 */ ;
parameter k =7;
input [k:0] d;
input rst ;
input clk;
output reg [k:0] q;
wire [k:0] d_buf;
CLKBUF n1 [k:0] (d_buf,d);
```

```
always @(posedge clk)
 if (rst)
 q <= 1'b0;
 else if(clk)
 q <= d_buf;
endmodule
```

The number of global clock buffers is device-specific; consult the vendor documentation for the device/family you are using.



## CHAPTER 25

# FX Messages 103 – 391

---

## FX103

**@N: Instance "<en>" with "<6>" loads has been replicated "<1>" time(s) due to a <hard> fanout limit of "<5>."**

The syn\_maxfan attribute overrides the default (global) fanout guide for an individual input port, net, or register output. A syn\_maxfan attribute has been applied and that to meet the fanout requirement, the associated instance has been replicated *n* times.

For the test case below, the syn\_maxfan attribute sets a fanout limit of 5 on instance en. Because the instance must drive six loads, the mapper replicates the instance one time to meet the fanout requirement.

```
module test(a, clk, rst, din, q);
 input [3:0] a;
 input [5:0] din;
 input clk, rst;
 output reg [5:0] q;
 reg en /* synthesis syn_maxfan = 5 */;

 always @ (posedge clk or posedge rst)
 begin
 if (rst) begin
 en <= 0;
 q <= 0;
```

```
 end
 else begin
 en <= &a;
 if (en)
 q <= din;
 end
 end
endmodule
```

## Action

Informative message; no action required. The synthesis tool does not respect either:

- the global Fanout Guide limit – The Fanout Guide set from the Device tab of the Implementation Options panel places a soft limit for the design.
- the syn\_maxfan limit – The attribute value places a hard limit when it is attached to nets, ports, primitive instances, and registers in certain technologies.

## FX104

**@N: Inserting <integer> buffers on net <netname> (with fanout of <integer>) because of a <limitType> fanout limit of <integer>.**

The syn\_maxfan attribute overrides the default (global) fanout guide for an individual input port, net, or register output. When fanout for a net is limited, additional buffers are used to drive the loads. The above note indicates that a net has been buffered.

For the test case below, the syn\_maxfan attribute sets a fanout limit of 1 on net b\_c. Because the net must drive two loads, the mapper buffers the net to meet the fanout requirement.

```
module test(a,b,q1,q2);
 input a;
 input b /*synthesis syn_maxfan = 1 */;
 output q1,q2;
 assign q1 = a +b;
 assign q2 = a & b;
endmodule
```

## Action

Always use a realistic value for the soft or hard fanout limit so that the tool respects the value.

The synthesis tool does not respect either:

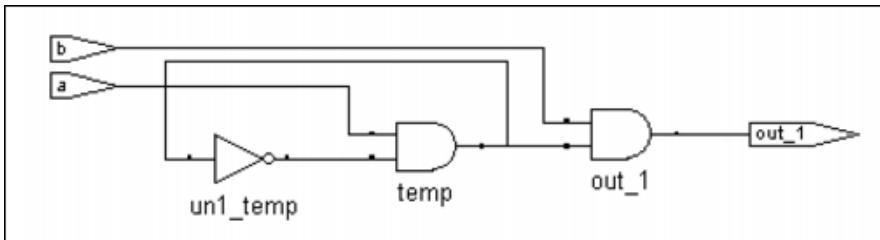
- the global Fanout Guide limit – The Fanout Guide set from the Device tab of the Implementation Options panel places a soft limit for the design.
- the syn\_maxfan limit – The attribute value places a hard limit when it is attached to nets, ports, primitive instances, and registers in certain technologies.

# FX105

## @W: Found combinational loop at <un1\_temp>

A combinational loop (a loop where the output of the combinational gates drives the inputs) was found in the HDL code (combinational loops can cause unexpected behavior and mismatches in the RTL to post-synthesis simulation). In the following test case, temp is an output of an assignment that uses itself as one of the inputs as shown in the schematic below.

```
module fx105_v (input a,b,output reg out_1);
 reg temp;
 always @(a,b,temp)
 begin
 temp <= (!temp) & a;
 out_1 <= (temp & b);
 end
 endmodule
```



## Action

To avoid the above warning, make sure that there are no combinational loops in the design.

```
module fx105_v (input a,b,output reg out_1);
reg temp;
always @(a,b,temp)
begin
 temp <= a;
 out_1 <= (temp & b);
end
endmodule
```

## FX106

### **@W: Using block\_RAM for single port RAM with syn\_ramstyle = no\_rw\_check**

To infer dual-port RAMs, which are fully synchronous, the following conditions must exist:

- Either the read address or the output is registered.
- Both the read address and the output are registered.

In all cases, the clock registering the address and the RAM must be the same. If either one of the following criteria is present, a dual-port RAM is inferred.

- Either the read and write addresses are different.
- Either the read and write clocks are different.

- Enable signals can be different.
- In a dual-port Block RAM implementation, there is conflict resolution behavior such as:
  - Both ports write to the same memory cell violating the clock-to-clock setup requirement so stored data is considered invalid.
  - One port attempts a read while the other port attempts a write on the same memory cell making the data out on the reading port invalid.

The synthesis tool creates glue by-pass logic to ensure pre- and post-synthesis simulation results. You can disable this by using “no\_rw\_check” option with the attribute syn\_ramstyle. These conflicts occur only in dual-port RAMs, hence if the option “no\_rw\_check” is used in a single-port RAM implementation, it assumes a block RAM is being requested. In the following test case, a blockRAM is implemented.

```
module ram_test (q, a, d, we, clk);
 output [7:0] q;
 input [7:0] d;
 input [6:0] a;
 input clk, we;
 reg [6:0] read_add;
 /* The array of an array register ("mem") the RAM will be
 inferred from */
 reg [7:0] mem [127:0] /* synthesis syn_ramstyle =
 "no_rw_check" */;
 assign q = mem[read_add];

 always @ (posedge clk) begin
 read_add <= a;
 if (we)
 /* Register RAM Data */
 mem[a] <= d;
 end
endmodule
```

## Action

Make sure that the clock registering the address and the RAM are the same. Remove the no\_rw\_check option on the attribute so that the glue logic is created.

```

module ram_test (q, a, d, we, clk);
output [7:0] q;
input [7:0] d;
input [6:0] a;
input clk, we;
reg [6:0] read_add;
/* The array of an array register ("mem") the RAM will be
 inferred from */
reg[7:0] mem [127:0] /* synthesis syn_ramstyle = "block_ram" */;
assign q = mem[read_add];

always @(posedge clk) begin
read_add <= a;
 if(we)
 /* Register RAM Data */
 mem[a] <= d;
end
endmodule

```

## FX107

**@W: RAM <instanceName> does not have a read/write conflict check.**  
**Possible simulation mismatch. To resolve a read/write conflict,**  
**either set syn\_ramstyle = rw\_check, or enable the "Read Write**  
**Check on RAM" Implementation Option. For more information,**  
**search for "read/write conflict check" in Online Help.**

To infer Block Select RAMs, which are fully synchronous, the following conditions must exist:

- Either the read address or the output is registered.
- Both the read address and the output are registered.

In both cases, the clock registering address and the output of the RAM must be the same.

If either one of the following criteria is present, a dual-port RAM is inferred:

- The read and write addresses are different.
- The read and write clocks are different.

- Enable signals are different.
- In the Block RAM implementation, there is conflict resolution behavior such as:
  - Both ports write to the same memory cell violating the clock-to-clock setup requirement so stored data is considered invalid.
  - One port attempts a read while the other port attempts a write on the same memory cell making the data out on the reading port invalid.

The synthesis tool creates glue by-pass logic to ensure pre- and post-synthesis simulation results. You can disable this by using `no_rw_check` option with the attribute `syn_ramstyle`. The above warning appears indicating that no glue logic has been created for this conflict as in the following test case.

```
module dualportram (q, a1, a2, d, we, clk, en);
output [7:0] q;
input [7:0] d;
input [6:0] a1, a2;
input clk, we, en;
reg [6:0] read_addr;
reg[7:0] mem [127:0] /* synthesis syn_ramstyle = "no_rw_check" */;
assign q = mem [read_addr];

always @ (posedge clk) begin
 if (we)
 mem[a2] <= d;
 read_addr <= a1;
 end
endmodule
```

## Action

Make sure that the clock registering address and the output of the RAM are the same. Remove the `no_rw_check` option on the attribute so that the glue logic is created. For more information, search for “`syn_ramstyle`” in Online Help.

```

module dualportram (q, a1, a2, d, we, clk, en);
output [7:0] q;
input [7:0] d;
input [6:0] a1, a2;
input clk, we, en;
reg [6:0] read_addr;
reg[7:0] mem [127:0] /* synthesis syn_ramstyle = "block_ram" */;
assign q = mem [read_addr];

always @ (posedge clk) begin
if (we)
mem[a2] <= d;
read_addr <= a1;
end
endmodule

```

## FX108

**@W: Block SelectRAM for specified technology cannot be inferred for RAM <mem[7:0]> with given coding style (flip-flop present at RAM output)**

To infer block select RAMs, which are fully synchronous, the following conditions must exist:

- Either the read address or the output is registered.
- Both the read address and the output are registered.

When the device family does not support block RAMs where the register is at its output, the RAM is implemented as select RAM as in the following test case.

```

module singleportram (q, a2, d, we, clk);
output [7:0] q;
input [7:0] d;
input[6:0] a2;
input clk, we;
reg [6:0] read_addr;
reg[7:0] mem [127:0]/* synthesis syn_ramstyle ="block_ram" */;
reg [7:0] q;

```

```
always @(posedge clk)
begin
 if (we)
 mem[a2] <= d;
 q = mem[a2];
 end
endmodule
```

## Action

Make sure that the output of the RAM is not registered when the device family does not support block RAMs. Implement the block RAM with the address registered.

```
module singleportram (q, a2, d, we, clk);
output [7:0] q;
input [7:0] d;
input[6:0] a2;
input clk, we;
reg [6:0] read_addr;
reg [7:0] mem [127:0]/* synthesis syn_ramstyle ="block_ram" */;
wire [7:0] q;
assign q = mem[read_addr];

always @(posedge clk)
begin
 if (we)
 mem[a2] <= d;
 read_addr <= a2;
// q = mem[read_addr];
 end
endmodule
```

**@N: Please copy dffrs.xnf to your design directory.**

With certain families, the mapper infers a flip-flop with asynchronous set and reset.

The EDIF file contains only the DFFRS primitive, however, to place and route this design, the dffrs.xnf file must be copied from directory *tool\_install\_directory/lib/technology* into the design directory. This xnf file contains the functionality of the DFFRS primitive. The following test case produces the above note.

```
module dff1 (q, qb, d, clk, set, reset);
 input d, clk, set, reset;
 output q, qb;
 // declare q and qb to be reg, because assigned inside always
 reg q, qb;

 always @(posedge clk or posedge set or posedge reset)
 begin
 if (reset) begin
 q = 0;
 qb = 1;
 end else if (set) begin
 q = 1;
 qb = 0;
 end else begin
 q = d;
 qb = ~d;
 end
 end
endmodule
```

The following warning appears along with this note:

@W: Your design contains FFs with both set and reset, they have been mapped to DFFRS.

## Action

Make sure that the xnf file is copied into the implementation directory before performing place and route. The xnf file has the functionality of the DFFRS (asynchronous set and reset register). This is also true for primitives DFFRSE, LD, LD\_1, LDC, LDC\_1, LDCP, LDCP\_1, LDP, LDP\_1, SLDCE\_1 and SLDE\_1 if inferred from the design.

# FX131

## @W: I/O pads should not be instantiated inside a compile point

User instantiated I/O pads should not be present inside a compile-point region when using the compile-point flow. In the following test case, the presence of an I/O pad inside the compile point results in the above warning.

```
module cp (// compile point module
 input c, d,
 output reg q);
 wire temp;
 IBUF I1 (.O(temp), .I(d));

 always@(posedge c)
 begin
 q <= temp;
 end

 endmodule

module top (// Top module
 input c, d,
 output q);
 cp U1 (.c(c), .d(d), .q(q));
endmodule
```

## Action

Instantiate the I/O pad in the top-level module of the design. To eliminate the warning in the above test case, edit the code as shown in the corrected test case below.

```
module cp (// compile point module
 input c, d,
 output reg q);

 always@(posedge c)
 begin
 q <= d;
 end

 endmodule
```

```
module top (// Top module
 input c, d,
 output q);
 wire temp;
 IBUF I1 (.O(temp), .I(d));
 cp U1 (.c(c), .d(temp), .q(q));
endmodule
```

## FX132

### **@W: Clock buffers should not be instantiated inside a compile point**

User-instantiated clock buffers cannot be present inside the compile point region when using the compile point flow. In the following test case, the presence of the clock buffer inside the compile point results in the above warning.

```
module cp (// Compile point module
 input c, d,
 output reg q);
 wire temp;
 BUFG C1 (.O(clk), .I(c));

 always@(posedge clk)
 begin
 q <= d;
 end
endmodule

module top (// Top module
 input c, d,
 output q);
 cp U1 (.c(c), .d(d), .q(q));
endmodule
```

### Action

Instantiate clock buffers in the top-level module of the design. To eliminate the warning in the above test case, edit the code as shown in the corrected test case below.

```
module cp (// Compile point module
 input c, d,
 output reg q);
begin
 q <= d;
end
endmodule

module top (// Top module
 input c, d,
 output q);
 wire temp;
 BUFG C1 (.O(temp), .I(c));
 cp U1 (.c(temp), .d(d), .q(q));
endmodule
```

## FX136

### **@A: Found internal transparent latches mapped to combinational logic.**

The mapper inferred a latch with an asynchronous clear and preset in specific technologies. Because the EDIF file contains only the LDCP\_1 primitive, you must copy the corresponding xnf file from directory *tool\_install\_directory/lib/technology* into the design directory before you can place and route the design. The xnf file contains the functionality of the inferred latch primitive. The following test case produces the above advisory. Note that the inferred primitive is present in the advisory. In the test case below, primitive LDCP\_1 is inferred.

```
// Level sensitive latch example 2, with set and reset
module latch2(q, data, clk, set, reset);
 output q;
 input data, clk, set, reset;
 assign q = reset ? 0 : (set ? 1 : (clk ? data : q));
endmodule
```

## Action

Make sure that the corresponding xnf file is copied into the implementation directory before performing place and route. The xnf file has the functionality of the inferred latch. This is also true for primitives DFFRS, DFFRSE, LD, LD\_1, LDC, LDC\_1, LDCP, LDCP\_1, LDP, LDP\_1, SLDCE\_1, and SLDE\_1 if inferred from the design.

# FX164

**@N: The option to pack registers in the IOB has not been specified.  
Please set syn\_useioff attribute.**

The mapper, by default, packs the register in the IOBs only to satisfy timing requirements. The registers along the critical path may or may not be packed into the IOBs. The packing of registers to the I/O buffers is controlled by the use of the syn\_useioff directive. To override the default behavior, use this directive locally on a flip-flop

```
define_attribute { port } syn_useioff {1}
```

or globally

```
define_global_attribute syn_useioff {1}
```

A test case which produces the above note has been added for reference.

```
module adder(out, a, b, c, cin, sel, reset, clk);
parameter size = 16; /* declare a parameter. Default required */
output [size-1:0] out;
wire [size-1:0] sum,out1;
reg [size-1:0] out;
// sum uses the size parameter
input cin, sel, reset, clk;
input [size-1:0] a, b, c; // 'a' and 'b' use the size parameter
reg [size-1:0] a_reg, c_reg,b_reg;
assign sum = a_reg + b_reg + cin;
assign out1 = sel ? sum : c_reg;
```

```

always @(posedge clk)
begin
 a_reg <= a;
 b_reg <= b;
 c_reg <= c;
 if (reset)
 out <= 16'h0000;
 else
 out <= out1;
end
endmodule

```

## Action

The registers are not packed in the I/O buffers by default. A global attribute can be set to pack all registers in the ports to the IOBs. If implemented locally or globally, the registers will have an IOB=true property on them which is visible in the technology view by positioning the cursor over the register in question.

You can also edit the EDIF file as follows:

```

(instance (rename out_oreg_0 "out_oreg[0]")
(viewRef PRIM (cellRef FD (libraryRef UNILIB)))
(property IOB (string "TRUE"))

```

# FX173

## **@E: Bidirectional pin <D[0]> feeds another bidirectional pin. Register or buffer must be added between pins.**

If a bidir bus port is driving another bidir bus port, the mapper does not honor the bit-slice assignments and splits the ports accordingly. In the following test case, bus ports C and D function as follows: Inputs: D(0), C(1), D(2); Outputs: C(0), D(1), C(2). However, the mapper selects both of these ports as IOBUFs and ties them to each other.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

```

```

entity b1_fpgal_top is
 port (A: inout std_logic;
 B: inout std_logic;
 C: inout std_logic_vector (2 downto 0);
 D: inout std_logic_vector (2 downto 0));
end b1_fpgal_top;

architecture struct of b1_fpgal_top is
begin
 A <= B;
 C(0) <= D(0);
 D(1) <= C(1);
 C(2) <= D(2);
end struct;

```

## Action

This error occurs only with bus ports. Examining ports A and B, B is assigned an IBUF and A is assigned an OBUF honoring the assignment A <= B. To eliminate the error in the above test case, make sure that the ports are registered or buffered as shown in the corrected test case below.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity b1_fpgal_top is
 port (A: inout std_logic;
 B: inout std_logic;
 C: inout std_logic_vector (2 downto 0);
 D: inout std_logic_vector (2 downto 0);
 CLK: in std_logic);
end b1_fpgal_top;

architecture struct of b1_fpgal_top is
begin
 process (CLK)
 begin
 if (CLK = '1' and CLK'event) then
 A <= B;
 C(0) <= D(0);
 D(1) <= C(1);
 C(2) <= D(2);
 end if;
 end process;
end struct;

```

## FX210

### **@E: Technology environment variable not set**

The above error is reported when place and route is invoked to do initial placement and the corresponding technology environment variable is not set. In the Synplify tools, this error occurs when place and route is invoked without first setting the environment variable.

#### Action

Make sure that the technology environment variable is set before running place and route.

## FX211

### **@N: Packed ROM <out1\_0[7:0] (9 input, 8 output)> to Block SelectRAM**

The synthesis tool supports the mapping of ROM into block RAM in certain architectures, provided that the conditions listed below are met.

1. Place a dff register in front of the ROM, or place one of the following after the ROM:

Asynchronous	Synchronous
dff, dffe	
dffr, dffre	sdffr, sdffre
dffs, dffse	sdffs, sdffse
dffpatr, dffpatre	sdffpatr, sdffpatre

In the above table, dffe is an enabled flip-flop, dffre is an enabled flip-flop with asynchronous reset, dffse is an enabled flip-flop with asynchronous set, and dffpatre is an enabled, vectored flip-flop with asynchronous reset pattern.

2. Make sure that registers and ROMs are within the same hierarchy.
3. Make sure that the number of outputs of the candidate ROM is 64 or fewer.
4. Assign values to at least half of the addresses. For example, for a ROM with ten address bits (1024 unique addresses), at least 512 of those unique addresses must be assigned values.

A note is generated for each ROM that is detected and mapped to a blockRAM, as in the following test case.

```
library ieee;
use ieee.std_logic_1164.all;

entity romtest is port(
addr : in std_logic_vector(6 downto 0);
set, clk : in std_logic;
out1_out : out std_logic_vector(3 downto 0));
end romtest;

architecture beh of romtest is
signal out1 : std_logic_vector(3 downto 0);
signal addr_temp : std_logic_vector(6 downto 0);
begin
out1_out <= out1;
process(clk)
begin
if clk'event and clk='1' then
 addr_temp <= addr;
end if;
end process;

process(addr_temp)
begin
case addr_temp is
when "0000000" => out1 <= X"1";
when "0000001" => out1 <= X"0";
when "0000010" => out1 <= X"3";
when "0000011" => out1 <= X"0";
when "0000100" => out1 <= X"5";
when "0000101" => out1 <= X"0";
when "0000110" => out1 <= X"7";
when "0000111" => out1 <= X"0";
when "0001000" => out1 <= X"9";
when "0001001" => out1 <= X"0";
when "0001010" => out1 <= X"b";
end case;
end process;
```

```
when "0001011" => out1 <= X"0";
when "0001100" => out1 <= X"d";
when "0001101" => out1 <= X"0";
when "0001110" => out1 <= X"f";
when "0001111" => out1 <= X"0";
when "0010000" => out1 <= X"0";
when "0010001" => out1 <= X"0";
when "0010010" => out1 <= X"a";
when "0010011" => out1 <= X"b";
when "0010100" => out1 <= X"8";
when "0010101" => out1 <= X"9";
when "0010110" => out1 <= X"0";
when "0010111" => out1 <= X"0";
when "0011000" => out1 <= X"0";
when "0011001" => out1 <= X"0";
when "0011010" => out1 <= X"2";
when "0011011" => out1 <= X"3";
when "0011100" => out1 <= X"5";
when "0011101" => out1 <= X"7";
when "0011110" => out1 <= X"9";
when "0011111" => out1 <= X"1";
when "0100000" => out1 <= X"a";
when "0100001" => out1 <= X"c";
when "0100010" => out1 <= X"e";
when "0100011" => out1 <= X"0";
when "0100100" => out1 <= X"0";
when "0100101" => out1 <= X"0";
when "0100110" => out1 <= X"0";
when "0100111" => out1 <= X"0";
when "0101000" => out1 <= X"0";
when "0101001" => out1 <= X"0";
when "0101010" => out1 <= X"0";
when "0101011" => out1 <= X"0";
when "0101100" => out1 <= X"0";
when "0101101" => out1 <= X"0";
when "0101110" => out1 <= X"0";
when "0101111" => out1 <= X"6";
when "0110000" => out1 <= X"0";
when "0110001" => out1 <= X"0";
when "0110010" => out1 <= X"9";
when "0110011" => out1 <= X"0";
when "0110100" => out1 <= X"0";
when "0110101" => out1 <= X"0";
when "0110110" => out1 <= X"a";
when "0110111" => out1 <= X"0";
when "0111000" => out1 <= X"0";
```

```
when "0111001" => out1 <= X"0";
when "0111010" => out1 <= X"f";
when "0111011" => out1 <= X"0";
when "0111100" => out1 <= X"0";
when "0111101" => out1 <= X"0";
when "0111110" => out1 <= X"a";
when "0111111" => out1 <= X"0";
when "1000000" => out1 <= X"0";
when "1000001" => out1 <= X"9";
when "1000010" => out1 <= X"0";
when "1000011" => out1 <= X"0";
when "1000100" => out1 <= X"0";
when "1000101" => out1 <= X"0";
when "1000110" => out1 <= X"0";
when "1000111" => out1 <= X"f";
when "1001000" => out1 <= X"0";
when "1001001" => out1 <= X"0";
when "1001010" => out1 <= X"0";
when "1001011" => out1 <= X"0";
when "1001100" => out1 <= X"c";
when "1001101" => out1 <= X"0";
when "1001110" => out1 <= X"0";
when "1001111" => out1 <= X"0";
when "1010000" => out1 <= X"d";
when "1010001" => out1 <= X"0";
when "1010010" => out1 <= X"0";
when "1010011" => out1 <= X"0";
when "1010100" => out1 <= X"0";
when "1010101" => out1 <= X"0";
when "1010110" => out1 <= X"0";
when "1010111" => out1 <= X"0";
when "1011000" => out1 <= X"0";
when "1011001" => out1 <= X"0";
when "1011010" => out1 <= X"0";
when "1011011" => out1 <= X"0";
when "1011100" => out1 <= X"0";
when "1011101" => out1 <= X"0";
when "1011110" => out1 <= X"0";
when "1011111" => out1 <= X"0";
when "1100000" => out1 <= X"0";
when "1100001" => out1 <= X"0";
when "1100010" => out1 <= X"0";
when "1100011" => out1 <= X"0";
when "1100100" => out1 <= X"0";
when "1100101" => out1 <= X"0";
when "1100110" => out1 <= X"0";
```

```
when "1100111" => out1 <= X"0";
when "1101000" => out1 <= X"0";
when "1101001" => out1 <= X"0";
when "1101010" => out1 <= X"0";
when "1101011" => out1 <= X"0";
when "1101100" => out1 <= X"0";
when "1101101" => out1 <= X"0";
when "1101110" => out1 <= X"0";
when "1101111" => out1 <= X"9";
when "1110000" => out1 <= X"1";
when "1110001" => out1 <= X"2";
when "1110010" => out1 <= X"3";
when "1110011" => out1 <= X"0";
when "1110100" => out1 <= X"4";
when "1110101" => out1 <= X"5";
when "1110110" => out1 <= X"0";
when "1110111" => out1 <= X"7";
when "1111000" => out1 <= X"0";
when "1111001" => out1 <= X"0";
when "1111010" => out1 <= X"6";
when "1111011" => out1 <= X"0";
when "1111100" => out1 <= X"8";
when "1111101" => out1 <= X"9";
when "1111110" => out1 <= X"0";
when "1111111" => out1 <= X"a";
when others => out1 <= X"2";
end case;
end process;
end beh;
```

## Action

For certain families, the synthesis tool automatically maps ROMs into Block SelectRAMs. To disable this feature, use the `syn_romstyle` attribute to implement the ROM as logic or distributed ROM.

The following example implements a ROM structure as a Select ROM.

```
signal z : std_logic_vector(3 downto 0);
attribute syn_romstyle : string;
attribute syn_romstyle of z : signal is "select_rom";
```

The following example implements a ROM structure as logic.

```
signal z : std_logic_vector(8 downto 0);
attribute syn_romstyle : string;
attribute syn_romstyle of z : signal is "logic";
```

## FX214

### @N: Generating ROM Data\_1[11:0]

A ROM was implemented using the distributed RAM resources in the CLBs. This type of implementation can happen when a `syn_rom_style` attribute is set to `select_rom` or when the mapper maps a ROM in distributed ROM resources.

```
module test_rom_style(op, a,clk);
output [3:0] op;
input [6:0] a;
input clk;
reg [3:0] z /* synthesis syn_romstyle ="select_rom" */;
reg [3:0] op ;

always @(a) begin
 case (a)
 7'b0000000: z = 4'b1011;
 7'b0000001: z = 4'b0001;
 7'b0010000: z = 4'b0011;
 7'b0011000: z = 4'b0010;
 7'b0000111: z = 4'b1110;
 7'b0100001: z = 4'b0111;
 7'b0001010: z = 4'b0101;
 7'b0100101: z = 4'b0100;
 7'b1000000: z = 4'b1100;
 7'b1000001: z = 4'b1101;
 7'b1000010: z = 4'b1111;
 7'b1001001: z = 4'b1110;
 7'b1100011: z = 4'b1010;
 7'b1101011: z = 4'b1011;
 7'b1111011: z = 4'b1001;
 7'b1111111: z = 4'b1000;
 default: z = 4'b0000;
 endcase
end
```

```
always@(posedge clk)
 op=z;
endmodule
```

## Action

You can change the implementation style of the ROM by using the `syn_rom`-style attribute and selecting the implementation as `logic` or `block_rom`, as required. This style change is illustrated in the modified test case below where the ROM style is `block_rom`.

```
module test_rom_style(op, a,clk);
output [3:0] op;
input [6:0] a;
input clk;
reg [3:0] z /* synthesis syn_romstyle ="block_rom" */;
reg [3:0] op ;

always @ (a) begin
 case (a)
 7'b00000000: z = 4'b1011;
 7'b00000001: z = 4'b0001;
 7'b00100000: z = 4'b0011;
 7'b00110000: z = 4'b0010;
 7'b00001111: z = 4'b1110;
 7'b01000001: z = 4'b0111;
 7'b00001010: z = 4'b0101;
 7'b01001010: z = 4'b0100;
 7'b10000000: z = 4'b1100;
 7'b10000001: z = 4'b1101;
 7'b10000010: z = 4'b1111;
 7'b10010001: z = 4'b1110;
 7'b11000011: z = 4'b1010;
 7'b11010111: z = 4'b1011;
 7'b11110111: z = 4'b1001;
 7'b11111111: z = 4'b1000;
 default: z = 4'b0000;
 endcase
end

always@(posedge clk)
 op=z;
endmodule
```

Note that the synthesis tool supports the mapping of ROM into block RAM in some architectures, provided that the conditions listed below are met:

1. Place a dff register in front of the ROM, or place one of the following after the ROM:

<b>Asynchronous</b>	<b>Synchronous</b>
dff, dffe	
dffr, dffre	sdffr, sdffre
dffs, dffse	sdffs, sdffse
dffpatr, dffpatre	sdffpatr, sdffpatre

In the above table, dffe is an enabled flip-flop, dffre is an enabled flip-flop with asynchronous reset, dffse is an enabled flip-flop with asynchronous set, and dffpatre is an enabled, vectored flip-flop with asynchronous reset pattern.

2. Make sure that registers and ROMs are within the same hierarchy.
3. Make sure that the number of outputs of the candidate ROM is 64 or fewer.
4. Assign values to at least half of the addresses. For example, for a ROM with ten address bits (1024 unique addresses), at least 512 of those unique addresses must be assigned values.
5. Make sure that the ROMs are detected by the synthesis tool so that the optimal implementation is made for such structures.

## FX273

### **@N: Packed ROMs "<rom>" and "<rom>" into dual-port block RAM**

The mapper is packing the specified ROMs into a single, dual-port block RAM. This message is notification that the ROMs are now merged into block RAM in the output netlist.

## Action

Informative message; no action required.

# FX275

### **@N: Startup value for instance <I\_1> is 0.**

The mapper encountered a RAM structure initialized to 0 by a \$readmemh or \$readmemb system task. The mapper converts the initial value specified by the task into technology-specific initial values in the EDIF file (for example, property INIT (string <>)).

The test case below infers a 16x1 select RAM and uses a \$readmemb construct to initialize the RAM with values from a data.dat file. For the test case, the data.dat file contains 16 single-line entries with individual values of 0000000000000000.

```
module top(input clk,we, input in1, input [3:0] addr, output out);
reg mem [15:0] ;
initial
begin
 $readmemh("data.dat", mem);
end

always @(posedge clk)
begin
 if (we)
 mem[addr] <= in1;
end

assign out = mem[addr];
endmodule
```

## Action

The RAM has been initialized to 0 and this same value will be propagated to the P&R tools in the EDIF file. To initialize the RAM structure being inferred to a non-zero value, provide a valid set of values in the accompanying data file.

FX276

The mapper encountered a multi-port write RAM structure defined by a \$readmemh or \$readmemb system task. The mapper converts the initial value specified by the task into technology-specific initial values in the EDIF file (for example, property INIT\_<> (string <>)).

The test case below infers a multi-port RAM with two write ports (depth=8, width=2) and uses a \$readmemh construct to initialize the RAM with values from a data.dat file. For the test case, the data.dat file contains 8 single-line entries with values of 001, 000, 001, 000, 001, 000, 001, and 000.

```

module ram(input [1:0] data1, data2, input [2:0] waddr0, waddr1,
 input we1, we2, clk1,clk2, output [1:0] q1,q2);
reg [1:0] mem [7:0] /* synthesis syn_ramstyle="no_rw_check" */ ;
reg [2:0]rdad0,rdad1;

assign q1 = mem[rdad0];
assign q2 = mem[rdad1];

initial
begin
 $readmemh("data.dat" , mem);
end

always @(posedge clk1)
begin
rdad0 <= waddr0;
 if(we1) mem[waddr0] <= data1;
end

always @(posedge clk2)
begin
rdad1 <= waddr1;
 if(we2) mem[waddr1] <= data2;
end

endmodule

```

## Action

The initial RAM values will be propagated to the P&R tools in the EDIF file. You must provide the required set of initial legal values for the RAM structure being inferred.

# FX284

### **@W: Initial values other than 0 and 1 are mapped to 0**

The mapper encountered a RAM structure defined by a \$readmemh or \$readmemb system task with an initial value that is not made up exclusively of 0s and 1s. The mapper converts the invalid initial values to 0s and issues the above warning.

The test case below infers a 16x1 select RAM and uses a \$readmemb construct to initialize the RAM with values from a data.dat file that includes an illegal character. For the test case, the data.dat file contains 16 single-line entries with individual values of 0100101111100X01 (the X entry is illegal).

```
module top(input clk,we, input in1 , input [3:0] addr, output out);
reg mem [15:0] ;
initial
begin
 $readmemh("data.dat" , mem) ;
end

always @(posedge clk)
begin
 if (we)
 mem[addr] <= in1;
 end

assign out = mem[addr];
endmodule
```

## Action

Make sure that only 1s and 0s are used to specify initial RAM values. For the test case, if the desired initial value is 07D2, the warning can be ignored (i.e., if the intended value for X is 0). If the desired initial value is 27D2, the entry in the data.dat file must be changed from X to 1.

# FX286

### **@E: Port "<P>" of Instantiated pad "<u1>" is not connected to a chip I/O pin**

An input/output port of an IBUF/OBUF was not connected to an I/O pin on the FPGA. The IBUF/OBUF requires its input/output port to be the top-level pins as they exist on the FPGA boundary. These pins cannot be connected to internal wires. In the test case below, the input port of instance u1 is mapped to internal wire temp which causes the mapper to error out.

```
module IBUF (O, I) /* synthesis syn_black_box */;
parameter CAPACITANCE = "DONT_CARE";
parameter IBUF_DELAY_VALUE = "0";
parameter IFD_DELAY_VALUE = "AUTO";
parameter IOSTANDARD = "DEFAULT";
output O;
input I;
endmodule

module top (a, b,c,d, out);
input a,b,c,d;
output out;
wire temp,temp2;
assign temp = a & b & c;
IBUF u1 (temp2, temp);
assign out = temp2 | d;
endmodule
```

## Action

Make sure that the input/output ports are mapped to I/O ports for an IBUF/OBUF. If a buffer is needed internally, use the BUF primitive instead of IBUF/OBUF. In the corrected test case below, a BUF primitive is used instead of an IBUF primitive because both the inputs and outputs are intermediate signals.

```
module BUF (O, I) /* synthesis syn_black_box */;
 output O;
 input I;
endmodule

module top (a, b,c,d, out);
 input a,b,c,d;
 output out;
 wire temp,temp2;
 assign temp = a & b & c;
 BUF u1 (temp2, temp);
 assign out = temp2 | d;
endmodule
```

## FX339

### @N: Found addmux in <view:work.adder(verilog)> inst <out\_5[15:0]>

For some device families, the mapper notes all of the multiplexers followed by an adder and optimizes for area by moving the multiplexer before the adder if applicable. For an adder followed by a multiplexer, it can be implemented in a carry-chain followed by a look-up table for every bit of the adder. If the look-up table can be implemented in the input of the adder and the result is a smaller circuit, the mapper will map accordingly. For all adders followed by multiplexer structures, the above note appears in the log file as in the following test case.

```
module adder (out, a, b, c, cin, sel, reset, clk);
 parameter size = 16; /* declare a parameter. Default required */
 output [size-1:0] out;
 wire [size-1:0] sum, out1;
 reg [size-1:0] out;
```

```
// sum uses the size parameter
input cin, sel, reset, clk;
input [size-1:0] a, b, c; // 'a' and 'b' use the size parameter
assign sum = a + b + cin;
assign out1 = sel ? sum : c;

always @(posedge clk)
if (reset)
 out <= 16'h0000;
else
 out <= out1;
endmodule
```

## Action

An adder plus multiplexer structure was detected. The multiplexer may be positioned in front of the adder structure in the Technology view for improved area optimizations.

In the RTL view of the above test case, the adder is followed by a mux, but in the technology view, the XORCY chain is preceded by a LUT implementation that includes the mux functionality.

# FX344

## **@W: Unrecognized syn\_ramstyle “<block\_rom>” on instance <mem[15:0]>**

An incorrect RAM style was specified using the syn\_ramstyle attribute. In the following test case, attribute syn\_ramstyle = “block\_rom” directs the mapper to map the memory to block\_rom. However, because the values that syn\_ramstyle can take are registers, block\_ram, no\_rw\_check, and rw\_check the mapper issues the above warning.

```
module ram
(input clk, we, input [9:0] rdaddr, wraddr, input [15:0] din,
output [15:0] dout);
reg [9: 0] reg_rdaddr ;
reg [15:0] mem[0 : 1023] ;
// synthesis syn_ramstyle = "block_rom"
```

```

always @ (posedge clk)
begin
 if (we)
 mem[wraddr] <= din;
 reg_rdaddr <= rdaddr ;
 end

 assign dout = mem[reg_rdaddr] ;
endmodule

```

## Action

The valid RAM styles can be registers, block\_ram, no\_rw\_check, and rw\_check. Specifying the correct RAM style as shown in the corrected test case below eliminates the warning.

```

module ram
(input clk, we, input [9:0] rdaddr, wraddr, input [15:0] din,
output [15:0] dout);
reg [9: 0] reg_rdaddr ;
reg [15:0] mem[0 : 1023] ; // synthesis syn_ramstyle = "block_ram"

always @ (posedge clk)
begin
 if (we)
 mem[wraddr] <= din;
 reg_rdaddr <= rdaddr ;
 end

 assign dout = mem[reg_rdaddr] ;
endmodule

```

# FX369

### **@W: IOBUF <u1>'s IO pin connects to a non bi-directional port<d2>**

An I/O pin of an IOBUF cell was connected to a non-bidirectional port (connecting an IOBUF primitive to a non-bidirectional port prevents the functionality of the primitive from being determined). In the test case below, the I/O pin of primitive IOBUF is connected to input port d2 which results in the above message.

```
module test (d1,d2,t,clk,q);
 input d1,d2,t,clk;
 output reg q;
 wire temp;
 IOBUF u1 (temp,d2,d1,t);

 always@(posedge clk)
 q <= temp;
endmodule
```

## Action

Connect a bidirectional port to the I/O pin of primitive IOBUF as shown in the modified test case below.

```
module test (d1,d2,t,clk,q);
 input d1,clk,t;
 inout d2;
 output reg q;
 wire temp;
 IOBUF u1 (temp,d2,d1,t);

 always@(posedge clk)
 q <= temp;
endmodule
```

# FX379

**@E: Unable to insert correct pad for port <*out\_top*>. Port is driven by a tristate in an unreachable hierarchy (blackbox or compile point).**

A design with a compile-point module contained a tristate buffer that drives a top-level port (tristate buffers cannot be pulled into pads which prevents pad insertion from being performed correctly). In the test case below, lower level entity io is declared as a compile point which causes the mapper to error out when performing its initial bottom-up synthesis to map compile points.

## Constraint File Entry

```
define_compile_point {v:work.io} -type {locked} -cpfile {}
```

## Test Case

```
module io (in,en,out);
 input in,en;
 output out;
 assign out = en ? in : 1'bZ;
endmodule

module top (in_top,en_top,out_top);
 input in_top,en_top;
 output out_top;
 io u1(in_top,en_top,out_top);
endmodule
```

## Action

If pad insertion is required, avoid placing compile points on modules that contain tristate buffers driving top-level ports. As an alternative, manually instantiate the I/O pads.

# FX384

## **@W: Instance <mem[7:0]> in view <ram> cannot be mapped to LRAM. Ignoring the syn\_ramstyle=select\_ram attribute**

The mapper encountered a true dual-port RAM model with the `syn_ramstyle` attribute set to `select_ram`. Because the mapper cannot map this model into LRAM, the attribute is ignored as reported in the above message.

```
module ram(data0, data1, waddr0, waddr1, we0, we1, clk0, clk1,
 q0, q1);
 parameter d_width = 8;
 parameter addr_width = 8;
 parameter mem_depth = 256;
 input [d_width-1:0] data0, data1;
 input [addr_width-1:0] waddr0, waddr1;
 input we0, we1, clk0, clk1;
 output [d_width-1:0] q0, q1;
```

```

reg [addr_width-1:0] reg_addr0, reg_addr1;
reg [d_width-1:0] mem [mem_depth-1:0]
/* synthesis syn_ramstyle="select_ram" */;
assign q0 = mem[reg_addr0];
assign q1 = mem[reg_addr1];

always @(posedge clk0)

begin
reg_addr0 <= waddr0;
if (we0)
 mem[waddr0] <= data0;
end

always @(posedge clk1)
begin
reg_addr1 <= waddr1;
if (we1)
 mem[waddr1] <= data1;
end
endmodule

```

## Action

For a true dual-port RAM, set the `syn_ramstyle` attribute to either `block_ram` (block RAM with glue logic) or `no_rw_check` (no glue logic) to map the RAM model to block RAM.

```

module ram(data0, data1, waddr0, waddr1, we0, we1, clk0, clk1,
q0, q1);
parameter d_width = 8;
parameter addr_width = 8;
parameter mem_depth = 256;
input [d_width-1:0] data0, data1;
input [addr_width-1:0] waddr0, waddr1;
input we0, we1, clk0, clk1;
output [d_width-1:0] q0, q1;
reg [addr_width-1:0] reg_addr0, reg_addr1;

/* Block-RAM with no glue logic*/
reg [d_width-1:0] mem [mem_depth-1:0]
/* synthesis syn_ramstyle="no_rw_check" */;

/* Block-RAM with glue logic*/
// reg [d_width-1:0] mem [mem_depth-1:0]
// /* synthesis syn_ramstyle="block_ram" */;
```

```
assign q0 = mem[reg_addr0];
assign q1 = mem[reg_addr1];

always @(posedge clk0)
begin
 reg_addr0 <= waddr0;
 if (we0)
 mem[waddr0] <= data0;
end

always @(posedge clk1)
begin
 reg_addr1 <= waddr1;
 if (we1)
 mem[waddr1] <= data1;
end
endmodule
```

## FX385

### **@W: Instance <mem[7:0]> in view <ram> cannot be mapped to LRAM. Ignoring the syn\_ramstyle=logic\_ram attribute**

The mapper encountered a true dual-port RAM model with the `syn_ramstyle` attribute set to `logic_ram`. Because the mapper cannot map this model into LRAM, the attribute is ignored as reported in the above message.

```
module ram(data0, data1, waddr0, waddr1, we0, we1, clk0, clk1,
 q0, q1);
parameter d_width = 8;
parameter addr_width = 8;
parameter mem_depth = 256;
input [d_width-1:0] data0, data1;
input [addr_width-1:0] waddr0, waddr1;
input we0, we1, clk0, clk1;
output [d_width-1:0] q0, q1;
reg [addr_width-1:0] reg_addr0, reg_addr1;
reg [d_width-1:0] mem [mem_depth-1:0]
/* synthesis syn_ramstyle="logic_ram" */;
assign q0 = mem[reg_addr0];
assign q1 = mem[reg_addr1];
```

```
always @(posedge clk0)
begin
reg_addr0 <= waddr0;
if (we0)
 mem[waddr0] <= data0;
end

always @(posedge clk1)
begin
reg_addr1 <= waddr1;
if (we1)
 mem[waddr1] <= data1;
end
endmodule
```

## Action

For a true dual-port RAM, set the `syn_ramstyle` attribute to either `block_ram` (block RAM with glue logic) or `no_rw_check` (no glue logic) to map the RAM model to block RAM.

```
module ram(data0, data1, waddr0, waddr1, we0, we1, clk0, clk1,
q0, q1);
parameter d_width = 8;
parameter addr_width = 8;
parameter mem_depth = 256;
input [d_width-1:0] data0, data1;
input [addr_width-1:0] waddr0, waddr1;
input we0, we1, clk0, clk1;
output [d_width-1:0] q0, q1;
reg [addr_width-1:0] reg_addr0, reg_addr1;

/* Block-RAM with no glue logic*/
reg [d_width-1:0] mem [mem_depth-1:0]
/* synthesis syn_ramstyle="no_rw_check" */;

/* Block-RAM with glue logic*/
//reg [d_width-1:0] mem [mem_depth-1:0]
/* synthesis syn_ramstyle="block_ram" */;

assign q0 = mem[reg_addr0];
assign q1 = mem[reg_addr1];
```

```
always @(posedge clk0)
begin
reg_addr0 <= waddr0;
if (we0)
 mem[waddr0] <= data0;
end

always @(posedge clk1)
begin
reg_addr1 <= waddr1;
if (we1)
 mem[waddr1] <= data1;
end
endmodule
```

## FX389

### **@E: Primitives with both asynchronous set and asynchronous reset are not supported in this technology.**

The mapper encountered an instance with both an asynchronous set and an asynchronous reset when primitives with both asynchronous sets and resets are not supported. In the test case below, the module describes a flip-flop with both an asynchronous set and an asynchronous reset which causes the mapper to error out with the above message when the primitive is encountered.

```
module test (q, clk, set, d, rst);
input clk;
input set;
input d;
input rst;
output reg q;

always @ (posedge clk or posedge set or posedge rst)
 if (rst)
 q <= 1'b0;
 else if (set)
 q <= 1'b1;
 else
 q <= d;
endmodule
```

## Action

Make sure that the RTL code describes primitives with either an asynchronous set or an asynchronous reset, but not both. In the corrected test case below, the set condition is changed to synchronous to eliminate the error.

```
module test (q, clk, set, d, rst);
 input clk;
 input set;
 input d;
 input rst;
 output reg q;

 always @(posedge clk or posedge rst)
 if (rst)
 q <= 1'b0;
 else if(set)
 q <= 1'b1;
 else
 q <= d;
endmodule
```

## CHAPTER 26

# FX Messages 401 – 1192

---

## FX401

### **@W: Unable to map RAM instance <mem[1:0]> to LRAM for technology specified.**

A RAM model was coded in such a way that it cannot be mapped to LRAM and there is a constraint for the mapper to map the RAM model to LRAM (`syn_ramstyle=select_ram`). The following test case describes a RAM model that cannot be mapped to LRAM; the `select_ram` attribute value results in the above message, and the RAM model is mapped to logic.

```
module test (q,we,wclk,addr,d);
output [1:0]q;
input we,wclk;
input [1:0]d;
input [3:0]addr;
reg [3:0] reg_addr;
reg [1:0] mem [15:0] /* synthesis syn_ramstyle="select_ram" */;

always @(posedge wclk)
begin
reg_addr <= addr;
if (we)
 mem[reg_addr] <= d;
end
```

```
assign q = mem[addr];
endmodule
```

## Action

Code the RAM model according to the LRAM architecture as shown in the modified test case below.

```
module test (q,we,wclk,addr,d);
output [1:0]q;
input we,wclk;
input [1:0]d;
input [3:0]addr;
reg [3:0] reg_addr;
reg [1:0] mem [15:0] /* synthesis syn_ramstyle="select_ram" */;
always @(posedge wclk)
begin
reg_addr <= addr;
if (we)
 mem[reg_addr] <= d;
end
assign q = mem[reg_addr];
endmodule
```

# FX403

**@N: Property "block\_ram" or "no\_rw\_check" found for RAM  
<mem[1:0]> with specified coding style. Inferring Block-RAM.**

The mapper has encountered a RAM with a syn\_ramstyle attribute set to either block\_ram or no\_rw\_check and that it is mapping the RAM to block RAM (i.e., the coding style is correct) as noted in the message text. In the following test case, a RAM is inferred at the RTL level. The mapper encounters the syn\_ramstyle=no\_rw\_check attribute and, since the coding style is a valid for a block RAM, maps the RAM to block RAM.

```
module bram_KxnS (q,we,wclk,addr,d);
parameter ram_depth = 16;
parameter ram_add_width = 4;
parameter data_bus_size = 2;
output reg [data_bus_size-1:0]q;
input we,wclk;
input [data_bus_size -1:0]d;
input [ram_add_width-1:0]addr;
reg [data_bus_size-1:0] mem [ram_depth-1:0]
/*synthesis syn_ramstyle="no_rw_check" */;

always @(posedge wclk)
begin
 if (we)
 begin
 mem[addr] <= d;
 q <= mem[addr];
 end
 else
 q <= d;
end
endmodule
```

## Action

Informative message; no user action required.

# FX406

### **@N: Found startup values on NRAM instance <mem\_1[1:0]>**

The mapper has encountered initialization values for a multi-port write RAM structure defined by a \$readmemh or \$readmemb system task.

The test case below infers a multi-port RAM with two write ports (depth=8, width=2) and uses a \$readmemb construct to initialize the RAM with values from a data.dat file. For the test case, the data.dat file contains 8 single-line entries with values of 001, 000, 001, 000, 001, 000, 001, and 000.

```
module ram(input [1:0] data1, data2, input [2:0] waddr0, waddr1,
 input we1, we2, clk1,clk2, output [1:0] q1,q2);
 reg [1:0] mem [7:0] /* synthesis syn_ramstyle="no_rw_check" */ ;
 reg [2:0]rdad0,rdad1;

 assign q1 = mem[rdad0];
 assign q2 = mem[rdad1];
 initial
 begin
 $readmemh("./data.dat", mem);
 end

 always @(posedge clk1)
 begin
 rdad0 <= waddr0;
 if(we1) mem[waddr0] <= data1;
 end

 always @(posedge clk2)
 begin
 rdad1 <= waddr1;
 if(we2) mem[waddr1] <= data2;
 end
endmodule
```

## Action

Informative message; no action required. The note indicates that initialization values have been found for the identified RAM.

# FX410

### @N: Implementing byte-wide Write-Enable feature for group <mem>

The mapper has encountered a RAM model with a byte-wide/write-enable configuration and that it will use the byte-wide write-enable feature available with the block RAM. In the following test case, RAM mem includes two byte-wide write-enables (byteena[1] and byteena[0]) which allow the mapper to use the byte-wide/write-enable feature to map the RAM model into a single block RAM instead of two separate block RAMs.

```
module test (clock,wren,address,byteena,data,q);
 input clock;
 input wren;
 input [7:0] address;
 input [1:0] byteena;
 input [15:0] data;
 output reg [15:0] q;
 reg [15:0] mem [255:0];

 always @(posedge clock)
 begin
 if (wren)
 begin
 if (byteena[1])
 mem[address][7:0] <= data[7:0];
 else if (byteena[0])
 mem[address][15:8] <= data[15:8];
 end
 end

 always @(posedge clock)
 begin
 q <= mem[address];
 end
endmodule
```

## Action

Informative message; no user action required.

# FX414

### **@W: INIT value is missing on LUT6\_2 instance "<u1>"**

A LUT6\_2 primitive was instantiated without specifying an INIT value with a defparam statement (without an INIT value, the mapper cannot infer the relationship between the input and output ports). In the test case below, LUT primitive LUT6\_2 is instantiated but, without an INIT value, results in the above message.

```
module test (output q,q1, input a,b,c,d,e,f);
LUT6_2 u1 (q,q1,a,b,c,d,e,f);
endmodule
```

## Action

To avoid the above warning, always specify an INIT value when instantiating a LUT primitive (the INIT value specifies the functionality of the LUT). In the modified test case below, a defparam is used to pass an initial INIT value of 64'h000000000000000100 to the LUT.

```
module test (output q,q1, input a,b,c,d,e,f);
defparam u1.INIT=64'h000000000000000100;
LUT6_2 u1 (q,q1,a,b,c,d,e,f);
endmodule
```

# FX429

### **@W: Possible timing failure in place & route due to a negative setup value <-1.000> on net <q> clocked by <clk>.**

The mapper encountered a negative setup value on an output net (a negative setup value is written as a negative OFFSET out value). Negative setup values can cause subsequent place and route failures as indicated in the message text.

As an example, if the clock period is specified as 4ns with a default output delay of 5ns, a negative setup condition exists as reported by the above warning.

## Constraint File Contents

```
define_clock {clk} -period 4 -clockgroup default_clkgroup_0
define_output_delay -default 5.00 -route 0.00 -ref {clk:r}
```

## Test Case

```
module reg_to_reg (input d1,d2,clk,output reg q);
reg d1_reg,d2_reg;
```

```
always@(posedge clk)
begin
 d1_reg <= d1;
 d2_reg <= d2;
 q <= d1_reg & d2_reg;
end
endmodule
```

## Result

The resultant ncf file will contain the equivalent

```
Output Constraints
NET "q" TNM = "q";
TIMEGRP "q" OFFSET = OUT: -1.000 : AFTER clk;
```

which will cause the subsequent place and route operation to fail.

## Action

Make sure that there are no negative setup conditions by applying proper and reasonable I/O delay constraints.

# FX430

## **@N: Found <n> global buffers instantiated by user**

Some number of global buffers have been instantiated in the RTL code. In the test case below, five BUFGs are instantiated as reported in the above message.

```
module multi_clks (d,q,clk);
parameter k = 4;
input [k:0] d,clk;
output [k:0] q;
wire [k:0] clk_int;
BUFG u1[k:0] (clk_int,clk);
FD u2[k:0] (q,clk_int,d);
endmodule
```

## Action

Informative message; no action required.

## Action

Either make sure that the number of instantiated global buffers does not exceed the number of global buffers available for the selected part as shown in the corrected test case below or use a device that includes the required number of global buffers.

```
module multi_clks (d,q,clk);
parameter k = 14;
input [k:0] d,clk;
output [k:0] q;
wire [k:0] clk_int;
BUFG u1[k:0] (clk_int,clk);
FD u2[k:0] (q,clk_int,d);
endmodule
```

# FX433

**@W: Clock buffer insertion on net <clkz> (fanout <2>) in view <view:work.core(verilog)> was not possible because of restrictions of compile points. You may need to instantiate a clock buffer on this net.**

A compile point was defined on a clock net inside a compile point and I/O insertion was disabled. Because of the compile point, the mapper cannot insert the clock buffer as indicated in the above message. For the following test case, the following project and constraint file settings result in the above warning.

```
##project options##
set_option -disable_io_insertion 1

##Constraints##
define_attribute {n:u1.clk} {syn_insert_buffer} {BUFG}
define_compile_point -comment {Automatically Inserted} {v:core}
-type {locked} -cpfile {core_cp.sdc}
```

## Test Case

```
module top (input [1:0] d, input clk1, clk2, rst, output [1:0] q);
assign clk = clk1 & clk2;
core u1 (d, clk, rst, q);
endmodule

module core (input [1:0] d, input clk, rst, output reg [1:0] q);
always @(posedge clk or posedge rst)
if (rst)
q <= 2'd0;
else
q <= d;
endmodule
```

## Action

Manually insert the clock buffer instead of applying the syn\_insert\_buffer attribute as shown in the modified test case below.

```
##project options##
set_option -disable_io_insertion 1

##Constraints##
define_compile_point -comment {Automatically Inserted} {v:core}
-type {locked} -cpfile {core_cp.sdc}
```

## Test Case

```
module top (input [1:0] d, input clk1, clk2, rst, output [1:0] q);
assign clk = clk1 & clk2;
core u1 (d, clk, rst, q);
endmodule

module core (input [1:0] d, input clk, rst, output reg [1:0] q);
wire clk_buf;
BUFG u2 (clk_buf, clk);

always @(posedge clk_buf or posedge rst)
if (rst)
q <= 2'd0;
else
q <= d;
endmodule
```

## FX434

**@W: Because of resource limitations, clock buffer insertion could not be done on net <netName> in view <viewName> (fanout <integer>).**

The number of clock buffer resources exceeds the 32 clock-net limit. The net and the fanout for that net are reported in the message.

### Action

The clock-net limit is more than sufficient for almost any design. Examine the srs log file and check for extraneous clock nets that could cause the limit to be exceeded (for example, clock nets defined within IP blocks).

## FX447

**@W: Module "<moduleName>" contains uninitialized components that restrict the use of module "<moduleName>" instances to only timing purposes. Initialize the components by recompiling the design, including IP, with Synplify version 9.0.2 (or later).**

This message can occur when using secure IP cores. The synthesis tool converts the secure IP core and generates an edif netlist. However, the software is only aware of the ports and logic components within the specified module for the IP, but does not know how to implement the functionality for this logic. Therefore, only timing delay information is used for optimizations by the synthesis tool.

### Action

You must recompile the design to initialize components for the secure IP core using synthesis software version 9.0.2 or later.

## FX469

**@W: Net <netName> is undriven; adding disabled tristate to drive this net. To avoid the addition of a tristate, add a driver to the net.**

The mapper encountered an undriven net. A disabled tristate is automatically added to the net.

### Action

To eliminate this warning, add a driver to the named net.

## FX474

**@W: User-specified initial value defined for some sequential elements which can prevent optimum synthesis results from being achieved.**

During sequential optimization, the software encountered sequential elements with INIT values applied through the RTL or through a constraints file. In the test case below, output register dop has an INIT value of 1.

```
module ff (clk, clr, di, dop,dop2);
 input clk, clr;
 input di;
 output reg dop /* synthesis INIT="1" */;
 output reg dop2;

 always @ (posedge clr or posedge clk) begin
 if (clr)
 dop <= 0;
 else
 dop <= di;
 end

 always @ (posedge clk) begin
 dop2<=dop;
 end
endmodule
```

## Action

This warning is generated as a part of the sequential optimization process; you can ignore the warning if the INIT value applied on the register is required. Applying initial values can cause registers to be preserved and prevent retiming/pipelining from being performed. Conversely, removal of unnecessary INIT values from the RTL code can improve synthesis results.

## FX480

**@N: Block RAM cannot be inferred for RAM <mem[1:0]> with specified coding style. Please set synthesis syn\_ramstyle = "no\_rw\_check" to infer block RAM."**

The mapper has encountered a RAM coding style that cannot be mapped to block RAM. In the following test case, a RAM is inferred at the RTL level. The mapper attempts to pack the RAM into block RAM (syn\_ramstyle=block\_ram) but, with the given coding style, cannot infer the block RAM as noted in the message text. The RAM is implemented in logic.

```
module bram_KxnS (q,we,wclk,addr,d);
parameter ram_depth = 16;
parameter ram_add_width = 4;
parameter data_bus_size = 2;
output reg [data_bus_size-1:0]q;
input we,wclk;
input [data_bus_size -1:0]d;
input [ram_add_width-1:0]addr;
reg [data_bus_size-1:0] mem[ram_depth-1:0]
/* synthesis syn_ramstyle="block_ram" */;

always @(posedge wclk)
begin
 if (we)
 begin
 mem[addr] <= d;
 q <= mem[addr];
```

```
 end
 else
 q <= d;
end
endmodule
```

## Action

To pack the RAM into block RAM, set the `syn_ramstyle` attribute to `no_rw_check` as shown in the corrected test case below.

```
module bram_KxnS (q,we,wclk,addr,d);
parameter ram_depth = 16;
parameter ram_add_width = 4;
parameter data_bus_size = 2;
output reg [data_bus_size-1:0]q;
input we,wclk;
input [data_bus_size -1:0]d;
input [ram_add_width-1:0]addr;
reg [data_bus_size-1:0] mem[ram_depth-1:0]
 /* synthesis syn_ramstyle="no_rw_check" */;

always @(posedge wclk)
begin
 if (we)
 begin
 mem[addr] <= d;
 q <= mem[addr];
 end
 else
 q <= d;
end
endmodule
```

## FX527

### @W: Found false-path constraint on register <out\_reg[0]> which prevents the register from being packed into DSP

A false-path constraint is defined on an output register which prevents the register from being packed into a DSP. In the following test case, the multiplier, by default, is mapped into a DSP primitive. When there are registers around the multiplier, the mapper attempts to pack any registers driven by the multiplier into the same DSP primitive; the false-path constraint on the output register prevents it from being packed into the DSP as indicated by the warning message. For the following test case, the `set_false_path` constraint in the constraint file results in the above warning:

```

Delay Paths

set_false_path -from {{i:out_reg[15:0]}} -to {{i:out[15:0]}}
```

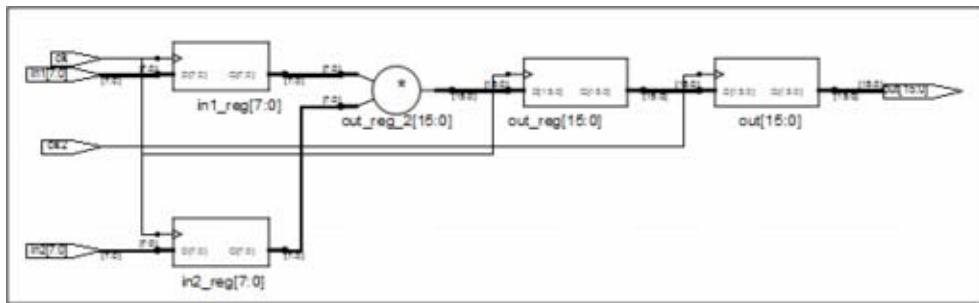
### Test Case

```
module test (
 input clk,
 input clk2,
 input [7:0] in1,
 input [7:0] in2,
 output reg [15:0] out);
reg [7:0] in1_reg;
reg [7:0] in2_reg;
reg [15:0] out_reg;

always @(posedge clk) begin
 in1_reg <= in1;
 in2_reg <= in2;
 out_reg <= in1_reg * in2_reg;
end

always @(posedge clk2) begin
 out <= out_reg;
end

endmodule
```



## Action

Remove/disable the false-path constraint specified on the register to allow the register to be packed into the DSP primitive. For example:

```
set_false_path -disable -from {{i:out_reg[15:0]}}
 -to {{i:out[15:0]}}
```

## FX553

### **@W: Could not implement Block ROM for <ramName>.**

Indicates that the mapper was unable to infer a block-RAM for the ROM. The target RAM (*ramName*) is reported in the message. The ROM must have a synchronous read.

## Action

To implement the ROM from a block-RAM, the ROM must include either output registers or address registers to be synchronous.

## FX580

### @N: Implementing RAM64M Packing for group <mem>

The mapper has inferred a RAM64M primitive for the named memory object. The below test case is a 2-bit wide by 64-bit deep RAM. The mapper infers a RAM64M primitive for memory element mem as indicated in the message text.

```
module top (DOA, ADDRA, ADDRDRD, DIA, WCLK, WE);
 input [5:0] ADDRA, ADDRDRD;
 input [1:0] DIA;
 input WCLK, WE;
 output [1:0]DOA;
 reg [1:0]mem[63:0];

 always @(posedge WCLK)
 if (WE)
 mem[ADDRDRD] <= DIA;
 assign DOA=mem[ADDRA];
endmodule
```

#### Action

Informative message; no user action required.

## FX614

### @W: Clock “<clock>” overrides a generated clock. Please set the syn\_clock\_priority attribute to 1 on object “<objectName>”.

A declared clock is overriding a derived clock and the declared clock has not been assigned a priority.

#### Action

Set the priority to 1 on the clock source (*objectName*) using the *syn\_clock\_priority* attribute.

## FX616

**@W: Declared clock <clock> is overriding a derived clock with same or higher priority. Please use the syn\_clock\_priority attribute to assign PRIORITY on object “<objectName>”. The PRIORITY should be higher than base clock PRIORITY.**

The priority of a declared clock on an object was the same or lower than the priority of the derived clock that it was attempting to override.

### Action

Set the priority of the declared clock to be higher than the priority of the derived clock's base clock using the syn\_clock\_priority attribute or by removing any priority on the base clock.

## FX638

**@W: INIT value not found on <u1>, calculating bdd**

A LUT primitive was instantiated without specifying an INIT value as a defparam. In such cases, the mapper calculates the bdd to evaluate the function of the LUT. In the test case below, LUT primitive LUT4 is instantiated, but because no INIT value is specified, the mapper displays the above message.

```
module my_and (a,b,c,d,o);
 input a,b,c,d;
 output o;
 LUT4 u1 (o,a,b,c,d);
endmodule
```

### Action

When instantiating a LUT primitive, be sure to explicitly specify the INIT value to define the functionality of the LUT. In the following test case, passing an INIT value (16'h8000) defines the functionality of the LUT as a 4-input AND gate.

```
module my_and (a,b,c,d,o);
 input a,b,c,d;
 output o;
 defparam u1.INIT =16'h8000;
 LUT4 u1 (o,a,b,c,d);
endmodule
```

## FX647

### **@E: Design has tristate buffer which could not be converted to Mux or Tristate Pad**

This error message is reported when:

- The tool is unable to infer MUX structures for internal tristates.
- The tool is unable to push an internal tristate to the top level thereby failing to infer a tristate pad.

In the test case below, e is an inout port in the module sub which is also a top-level inout port. It is not possible to infer tristate pads as cross-boundary optimization is stopped when the syn\_hier=fixed property is applied on the module sub.

```
module top(a, b, c, d, e, f);
 output [2:0] f;
 inout [3:0] e;
 input [2:0] a, b, c, d;
 assign f = e[2:0] & d;
 sub xx(a,b,c,d,e);
endmodule

module sub (a,b,c,d,e)/synthesis syn_hier="fixed"/;
 inout [3:0] e;
 input [2:0] a,b,c,d;
 assign e =
 {a[2], (a & b)}
 ? {c[2], (c & d)} : 4'bz;
endmodule
```

## Action

Removing the attribute `syn_hier="fixed"` enables cross-boundary optimization and tristate pads are inferred by the mapper.

```
module top(a, b, c, d, e, f);
output [2:0] f;
inout [3:0] e;
input [2:0] a, b, c, d;
assign f = e[2:0] & d;
sub xx(a,b,c,d,e);
endmodule

module sub (a,b,c,d,e);
inout [3:0] e;
input [2:0] a,b,c,d;
assign e = {a[2], (a & b)?{c[2], (c & d)}: 4'bz;
endmodule
```

## FX675

**@N: Design reset will not be synchronized, since no valid synchronization clock found on this chip.**

The above note indicates that no valid synchronization clock is available which prevents synchronization of the design through a common reset.

### Action

Informative message; no action required.

## FX702

**@N: Found startup values on RAM instance <mem>**

The mapper has encountered initialization values for a RAM structure defined by a \$readmemh or \$readmemb system task. The mapper converts the initial value specified by the task into technology-specific initial values in the EDIF file (for example, property INIT (string <>)). In the test case below, a 16x1 select RAM is inferred and initialized by a \$readmemb system task that takes its values from a data.dat file. The data.dat file contains 16 single-line entries with individual values of 1010011111010010.

```
module top(input clk,we, input in1 , input [3:0] addr, output out);
reg mem [15:0] ;
initial
begin
 $readmemh("./data.dat", mem);
end
```

```
always @(posedge clk)
begin
 if (we)
 mem[addr] <= in1;
end

assign out = mem[addr];

endmodule
```

## Action

Informative message; no action required. The note indicates that initialization values have been found for the identified RAM.

# FX707

**@W: Register <register> has both asynchronous set and reset. Your design may not work on the board. <Vendor> recommends you change your RTL. This warning is only issued once, although it may apply to other registers as well.**

One or more registers are defined with both an asynchronous set and an asynchronous reset which is not supported by the specified vendor technology. This type of coding can result in a simulation mismatch if the asynchronous set and reset occur simultaneously.

```
module test (
 input clk, set, rst,
 input d,
 output reg q);

 always@(posedge clk or posedge set or posedge rst)
 begin
 if(rst)
 q <= 0;
 else if(set)
 q <= 1;
 else
 q <= d;
 end
endmodule
```

## Action

Modify the RTL to remove the set asynchronous input as shown in the corrected test case below.

```
module test (
 input clk,set,rst,
 input d,
 output reg q);

 always@(posedge clk or posedge rst)
 begin
 if(rst)
 q <= 0;
 else if(set)
 q <= 1;
 else
 q <= d;
 end
endmodule
```

## FX712

**@E: Register <register> has both asynchronous set and reset.  
<Vendor> recommends you change your RTL.**

A register has been defined with both an asynchronous set and asynchronous reset which is not supported by the specified vendor technology.

```
module test (
 input clk,set,rst,
 input d,
 output reg q /*synthesis syn_map_dffrs = 0*/);

 always@(posedge clk or posedge set or posedge rst)
 begin
 if(rst)
 q <= 0;
 else if(set)
```

```
 q <= 1;
 else
 q <= d;
end
endmodule
```

## Action

Set the `syn_map_dffrs` attribute to 1 to cause the software to map either the set or reset input to logic or modify the RTL to remove one of the asynchronous inputs. Note that using the attribute results in warning [FX707](#) to indicate that the design may not work as intended. The default for the `syn_map_dffrs` attribute is 1; setting the attribute to 0 forces the error message. The following example illustrates using the `syn_map_dffrs` directive.

```
module test (
 input clk,set,rst,
 input d,
 output reg q /*synthesis syn_map_dffrs = 1*/;

always@(posedge clk or posedge rst)
begin
 if(rst)
 q <= 0;
 else if(set)
 q <= 1;
 else
 q <= d;
end
endmodule
```

# FX726

## **@W: Ignoring out of range global buffer count of <6> for chip <view:work.top(verilog)>**

The user-specified global buffer count was greater than the number of available global buffers in the device. In such cases, the mapper uses the available number of global buffers and then uses other buffers for the remaining clocks.

The test case below is implemented using a device that includes four global buffers. In the test case, the buffer count specified by the syn\_global\_buffers attribute is 6, which results in the above message.

```
module top (clk,d, q, reset)
 /* synthesis syn_global_buffers = 6 */;
 input [5:0]clk;
 input [5:0]d;
 output [5:0]q;
 input reset;

 generate
 begin :b1
 genvar i;
 for (i = 0; i < 6 ; i = i + 1)
 begin : u
 d_flop u1(.d(d[i]), .clk(clk[i]), .reset(reset),
 .q(q[i]));
 end
 end ;
 endgenerate
 endmodule

 module d_flop (d,clk,reset,q);
 input clk,d,reset;
 output reg q;

 always @(posedge clk or posedge reset)
 if (reset)
 q <= 1'b0;
 else
 q <= d;
 endmodule
```

## Action

Make sure that the value for syn\_global\_buffers is less than or equal to the total number of global buffers available in the device. In the corrected test case below, the buffer count specified by the attribute (3) is less than the number of buffers available (4).

```
module top (clk,d,q,reset) /* synthesis syn_global_buffers = 3 */;
 input [5:0]clk;
 input [5:0]d;
 output [5:0]q;
 input reset;
```

```
generate
 begin :b1
 genvar i;
 for (i = 0; i < 6 ; i = i + 1)
 begin : u
 d_flop u1(.d(d[i]), .clk(clk[i]), .reset(reset),
 .q(q[i]));
 end
 end ;
 endgenerate
endmodule

module d_flop (d,clk,reset,q);
 input clk,d,reset;
 output reg q;

 always @(posedge clk or posedge reset)
 if (reset)
 q <= 1'b0;
 else
 q <= d;
endmodule
```

## FX756

**@W: Cannot map <mem[1:0]> to <technology> block RAM -- no support for READ\_FIRST mode.**

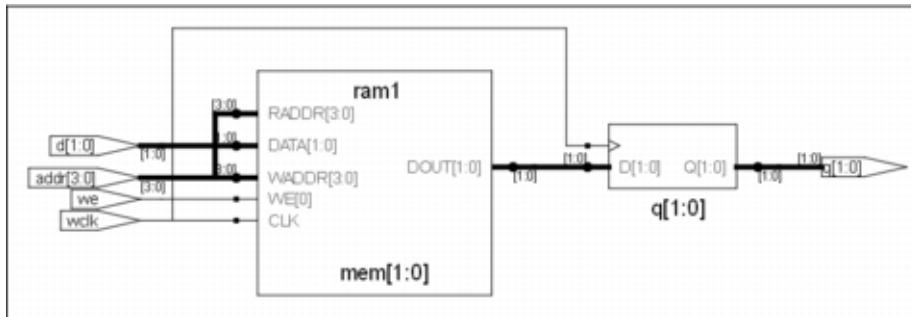
The mapper encountered a RAM model in read-first mode with a constraint to map to block RAM (block RAMs for the selected technology do not support a read-first mode). For the following test case, the mapper displays the above message and maps the RAM model to registers.

```
module bram_KxnS (q,we,wclk,addr,d);
 output reg [1:0]q;
 input we,wclk;
 input [1:0]d;
 input [3:0]addr;
 reg [1:0] mem [15:0] /*synthesis syn_ramstyle="block_ram" */;
```

```

always @(posedge wclk)
begin
 if (we)
 mem[addr] <= d;
 q <= mem[addr];
end
endmodule

```



## Action

To map a RAM model into block RAM, code the RAM in a style other than read-first mode. In the modified test case below, the RAM is coded in a write-first mode which allows the mapper to map the RAM model into block RAM.

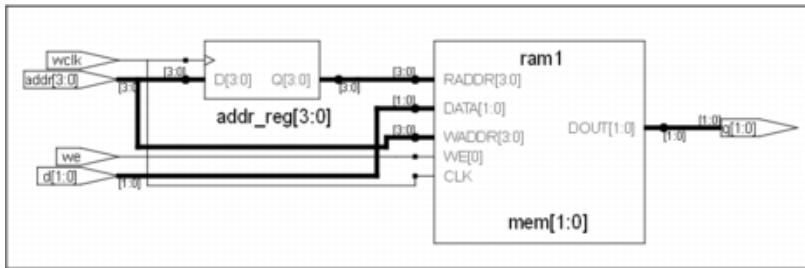
```

module bram_KxnS (q,we,wclk,addr,d);
output [1:0]q;
input we,wclk;
inout [1:0]d;
input [3:0]addr;
reg [1:0] mem [15:0] /*synthesis syn_ramstyle="block_ram" */;
reg [3:0] addr_reg;

always @(posedge wclk)
begin
 if (we)
 mem[addr] <= d;
 addr_reg <= addr;
end

assign q = mem[addr_reg];
endmodule

```



## FX867

**@E:** A regional clock IOB <padName> placed at <padLocation> is found that drives BUFG <instanceName>. You can use the CLOCK\_DEDICATED\_ROUTE constraint with a FALSE value in the SDC file to have your design continue. However, timing results might degrade.

The clock IOB driving the BUFG is not optimally placed and can generate an error during placement and routing.

### Action

Use the CLOCK\_DEDICATED\_ROUTE attribute in the constraint file to have the synthesis software ignore the specific clock placement rule and continue through placement and routing. If possible, you should fix all clock placement rule violations in the design. Otherwise, the timing results might degrade.

## FX873

**@E:** DSP is highly utilized and has many cascade chains. To generate a legal placement, every odd cascade chain needs to match a single DSP. However, there are <integer> unpacked odd cascade

**chains, but only <integer> single DSP remains after packing.  
Consider mapping some DSPs to logic gates.**

DSP that utilizes many cascade chains is near 100% utilization. However, the number of odd unpacked cascade chains remaining can exceed the number of non-cascading DSPs. For legal placement, every odd cascade chain remaining requires a non-cascading DSP.

#### Action

To lower DSP utilization, pack some DSPs to logic gates.

## FX878

**@E: The BUFG <instanceName> is being driven by <cellType> <instanceName> which is placed at location <placementLocation>. The <padName> placement is located at a site that prevents the <padName> from driving a BUFG. Please modify the <padName> placement to a location that is valid to drive a BUFG.**

The specified BUFG instance location constraint is not allowed.

#### Action

Use the syn\_loc attribute to specify a valid location placement constraint for the BUFG instance in the constraint file.

## FX879

**@E: Unable to place BUFG <instanceName> legally.**

The BUFG instance location constraint has not been specified.

### Action

Use the `syn_loc` attribute to specify a location placement constraint for the BUFG instance in the constraint file.

## FX880

**@E: Missing location constraint for BUFR instance <*instanceName*>. Please lock the placement for this instance.**

The BUFR instance location constraint has not been specified.

### Action

Use the `syn_loc` attribute to specify the location placement constraint for the BUFR instance in the constraint file.

## FX881

**@E: Missing location constraint for BUFH instance <*instanceName*>. Please lock the placement of this instance.**

The BUFH instance location constraint has not been specified.

### Action

Use the `syn_loc` attribute to specify the location placement constraint for the BUFH instance in the constraint file.

## FX907

### **@E: Illegal pad connection on instance <instancePort>**

The synthesis software either encountered:

- An internal tristate present in the design. The architecture requires that tristates can only be mapped to the top-level pads.
- A package pin that is not connected to a top-level port for an inferred I/O instance.

#### Action

You must change the RTL to correct this problem.

## FX932

### **@E: The constraint <constraintValue> is not applicable to component <instanceName>. Please modify the constraint to a valid setting.**

An inappropriate constraint was applied to a component instance. The following placement-location constraint illustrates this error:

```
define_attribute {i:f1p_inst} syn_loc {RAMB36_X0Y0}
```

In the above constraint, the specified instance cannot be set to a block RAM location.

#### Action

Make sure that the constraint specified is appropriate for the target component.

**@W:**

**@W:**

**@E: End of unplaced-components list.**

Indicates that one or more unplaced components have been encountered in the netlist.

**Action**

Unplaced components, which can include BUFH, IODELAY, and ODELAY, are reported earlier in the log file. Manually place the indicated component or components and repeat the operation until this error message is no longer generated.

## FX953

**@E: The XDC file was not specified to create the constraints file for Synplify Placement. Please add the XDC file to your project.**

The input XDC file has not been specified for your project, so Synplify Placement cannot create the output PDC constraint file.

**Action**

Make sure to add the XDC file to the project. Synplify Placement calls the xdc2pdc utility and requires that the XDC file be included in the project to run successfully. Ensure that the contents of the XDC file is correct.

## FX955

### **@E: The PDC file was not specified to create the placement constraints for Synplify Placement.**

The PDC file has not been specified for your project, so Synplify Placement cannot create the output placement constraints for this file.

#### Action

Make sure to add the PDC file to the project. Synplify Placement calls the xdc2pdc utility and requires that the PDC file be included in the project to run successfully. Check for correct contents of the PDC file or use the default PDC file after you import constraints.

## FX963

### **@A: Dual-adder packing for <instanceName> unsuccessful – number of pins is greater than four**

If you specify SIMD mode for 4-input adders, the synthesis software will not decompose them into 2-input adders. Therefore, these adders cannot be packed into the DSP using dual-adders with SIMD mode.

#### Action

This advisory message notifies you that SIMD mode cannot be used for these 4-input adders.

## FX964

### **@A: Dual-adder packing for <*instanceName*> unsuccessful – number of pins is greater than three**

If you specify SIMD mode for 3-input adders, the synthesis software will not decompose them into 2-input adders. Therefore, these adders cannot be packed into the DSP using dual-adders with SIMD mode.

#### Action

This advisory message notifies you that SIMD mode cannot be used for these 3-input adders.

## FX965

### **@A: Dual-adder packing unsuccessful – size for <*instanceName*> exceeds 24 bits**

The instance specified for SIMD mode exceeds the 24-bit size limitation for dual-adder packing to be successful. Therefore this adder will be packed into a single DSP.

#### Action

This advisory message notifies you that the specified instance will cause SIMD mode to be unsuccessful because it exceeds the 24-bit size limit. Make sure that the sizes for each adder are less than or equal to 24-bits so they can be packed into one DSP.

## FX966

### **@W: Accumulator not supported with dual-adder packing for <instanceName>**

Accumulators cannot be packed into the DSP using dual-adders with SIMD mode. Currently, SIMD mode only supports adders and subtractors.

#### Action

Be aware that even though you specify `syn_DSPstyle="simd"`, the software will pack only one adder into the DSP for an accumulator.

## FX968

### **@W: Dual-adder packing for <instanceName> unsuccessful – adder pair could not be found**

SIMD mode requires that a pair of registers be packed successfully into the DSP. For example, if a design contains three adders, then two adders are packed into a DSP using dual-adders with SIMD mode and the remaining adder without a pair is packed into its own DSP. This message notifies you about the instance specified with SIMD mode for which the software could not find a pair to pack into the DSP.

#### Action

Make sure that all adders in your design can be assigned to a pair for SIMD mode to run successfully. Otherwise, the adder without a pair is packed into a single DSP.

## FX979

### **@A: Adder <instanceName> includes a driving multiplier -- mult-add is preferred over dual-adder packing**

If one of the two adders you specified with SIMD mode contains a multiplier driving the adder, then dual-adder packing does not occur for this multiplier-adder. The adders are not mapped to a single DSP.

The advisory message above occurs for the following design, which includes a multiplier driving an adder specified with SIMD mode:

```
module mult (a,b,c,d,e,o1,o2);
 input [10:0] a;
 input [10:0] b;
 input [21:0] c;
 input [21:0] d,e;
 output [21:0] o1 /* synthesis syn_DSPstyle = "simd" */;
 output [21:0] o2 /* synthesis syn_DSPstyle = "simd" */;
 assign o1 = a*b+c;
 assign o2= d+e;
endmodule;
```

#### Action

To ensure that dual-adder packing occurs, do not specify SIMD mode for a multiplier that drives adders. Otherwise, each adder is mapped to a separate DSP.

## FX985

### **@E: ROM instance <instanceName> exceeds the register threshold and cannot be mapped to RAM in the specified technology.**

The register threshold for the ROM has been exceeded. For this technology, the number of registers that can be mapped to the RAM is controlled by the `syn_max_memsize_reg` attribute. The default value for this attribute is 128. This message occurs when the number of registers to be mapped to RAM exceeds the register threshold.

## Action

You can either:

- Use the `syn_max_memsize_reg` attribute to raise the threshold limit so that the ROM can be mapped to RAM.
- Specify the `syn_ramstyle=registers` attribute to force the specified ROM to be mapped to registers.

## FX986

### **@E: Differential I/O standard <IOStdType> found on non-differential pad <IOPadName>**

You specified a differential I/O standard type for a single-direction I/O port that does not support this standard. You must select a correct I/O standard type for the specified I/O ports when synthesizing your design.

The message above occurs if you synthesize the following design with the I/O standard constraint specified below in the constraint file:

```
define_io_standard {in_bit} syn_pad_type {LVDS_18} -delay_type {bidir}

module test (in_bit,in_bus,in_bus2,clk,out_bit,out_bus, out_bus2,
bidir, en, out_bit2); /* synthesis syn_useioff = 1 */
input in_bit,clk, en;
input [3:0] in_bus,in_bus2;
output reg out_bit;
output out_bit2;
output reg [3:0] out_bus,out_bus2;
 inout bidir;

always @(posedge clk)
begin
out_bus = in_bus;
out_bit = in_bit;
out_bus2 = in_bus2;
end
```

```
assign bidir = en ? in_bit : 1'bz;
assign out_bit2 = bidir;

endmodule
```

## Action

Make sure to specify a valid I/O standard type that supports the I/O ports. In the example above, specify the following I/O standard constraint instead and rerun synthesis:

```
define_io_standard {in_bit} syn_pad_type {LVCMS_18} -delay_type {bidir}
```

Otherwise, this I/O standard type DRC violation can cause the place and route to fail.

# FX991

## **@E: Invalid configuration encountered during mapping.**

During mapping, computations on the BDD (binary decision diagram) representing the intermediate netlist resulted in corrupt data or the entry into an illegal state.

# FX1001

## **@W: Duplicate clock constraint set on net <netName>. Constraint for clock <clockName> on this net not forward-annotated in UCF. Remove the duplicate constraint from your design to ensure the correct clock constraint is forward-annotated.**

Duplicate clocks are defined for the named net, but only one clock constraint can be forward annotated to the constraint file. The clock identified in the warning will not be used.

## Action

To eliminate this warning, check the `create_clock` and `create_generated_clock` entries and make sure that only one clock is defined for the named net.

# FX1009

### **@E: Differential I/O pin pairs <*pinName1*> and <*pinName2*> reference different I/O standards.**

You specified a different I/O standard for each pin of the differential I/O pair that generates this error. Therefore, the synthesis software cannot insert the differential I/O pad.

## Action

Make sure that the differential I/O pin pair uses the same I/O standard. For example, suppose differential I/O pin 1 was specified with LVDS\_18 and pin 2 was specified with LVDS\_25. You must change the I/O standard so that the type you use is the same for both pins of the differential pair.

To ensure that differential I/O pads are inferred, the two ports of a differential I/O pair must:

- Specify location information for the pins with the `syn_loc` attribute.
- Use the same differential I/O standard type.

# FX1010

### **@E: Both pins of differential pin pair <*pinName1*> – <*pinName2*> are in use.**

If the synthesis software determines that either of the following conditions occur:

- Both input ports have loads of which one is not a differential I/O pin pair.
- Both output ports have drivers of which one is not a differential I/O pin pair.

Then, the synthesis software errors out. The software assumes both pins are being used, but one of the input/output pins is not a differential pin pair. Therefore, the software cannot insert the differential I/O pad.

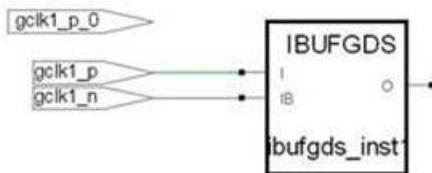
### Action

Use the `syn_loc` attribute to change the placement location constraint to a pin pair location or remove the connection to the input/output pin that is not the differential pin pair.

## FX1012

**@E: Differential buffer <IPname> <moduleName> is connected to pins <pinName1> and <pinName2> which are not a differential pair.**

After analyzing an existing differential buffer in the design, the synthesis software has verified that the two input/output ports are not a differential pair and errors out. In the following example, the differential I/O pad `ibufgds_inst` `IBUFGDS` is connected to pins `gclk2_p_n` and `gclk1_p`. However, the software determines that these pins are not a differential pair.



## Action

Use the syn\_loc attribute to change the placement location constraint for the pin of the differential I/O pad that is not a differential pin pair or correct the connection to the differential pin pair.

# FX1013

### **@E: <instanceName> is not supported for this device**

The instance is not supported for the specified device.

## Action

Remove or replace the instance by one that supports the device.

# FX1014

### **@E: ISERDES instance “<instanceName>” needs to be fixed. Please check the input constraints to correct this.**

ISERDES instances require that you set a location in the constraint file. The specified instance does not have a location constraint.

## Action

You must add a location constraint for the ISERDES instance specified in the constraints file.

## FX1019

### **@N: Adding ASYNC\_REG property on synchronizing instance <instanceName>.**

An ASYNC\_REG property is being added to the instances in a synchronization register chain (register chains are used to synchronize a signal between two asynchronous clock domains). In the message, *instanceName* identifies the first register in the chain. The number of instances in the chain is specified by a syn\_async\_reg attribute in the FDC constraint file (the default is two). Each instance in the chain includes an attached ASYNC\_REG property which is forwarded annotated to the XDC file. Tcl find commands can be used to identify the synchronization registers in a design; for information on locating synchronization registers, see the syn\_async\_reg attribute description in the *Attribute Reference Manual* or search for the string syn\_async\_reg in online help.

#### Action

Informative message; no user action required.

## FX1021

### **@E: Explicit initial value on port <portName> of instance <instanceName> is not supported in mixed language mode.**

This error occurs when a VHDL entity containing an input port with an initial value is instantiated in the Verilog top-level module.

For the following testcase, the VHDL entity specifies an initial value for the input port data that is instantiated in the Verilog top-level module. Since this port is undriven, the mapper cannot forward-annotate the initial value and this error is generated.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
```

```
entity up_counter_load is
 port (
 cout :out std_logic_vector (7 downto 0);
 data :in std_logic_vector (7 downto 0):= "11110000";
 load :in std_logic;
 enable :in std_logic;
 clk :in std_logic;
 reset :in std_logic
);
end entity;

--Rest of code
```

## Action

When instantiating VHDL components in Verilog modules, make sure that the undriven input ports do not specify any initial values in the VHDL entity.

# FX1028

**@E: Parameter <paramName> of instance <instanceName> set to improper value <value>.**

The value of the parameter is incorrect. The period of the clock cannot be computed.

## Action

Refer to the vendor place-and-route documentation for default values, allowed values, and data types for each parameter.

## FX1029

**@E: Parameter <paramName> of instance <instanceName> set to improper value <value>.**

The value of the parameter is incorrect. The period of the clock cannot be computed.

### Action

Refer to the vendor place-and-route documentation for default values, allowed values, and data types for each parameter.

## FX1034

**@E: Asymmetric TDP RAM with width more than 36 is not yet supported.**

An asymmetric TDP (true dual port) RAM has a width greater than 36 bits.

### Action

Split RAMs with widths greater than 36 into smaller RAMs.

## FX1035

**@N: Inserting BUFG on non-clock net <netName> (fanout: <integer>). To prevent insertion of a BUFG on this net, copy and paste the string 'define\_attribute {n:<netName>} {syn\_auto\_insert\_bufg} {0}' into your FDC file.**

A BUFG is being inserted on a high-fanout net. Both the net and the fanout count are reported in the message.

## Action

By default, BUFGs are automatically inserted on high-fanout control nets such as set, reset, clear, or enable when the fanout reaches 1000. To disable buffer insertion on the named net, cut and paste the `define_attribute` string from the message into your FDC file and remap.

To disable buffer insertion globally for a design, include the following attribute in your FDC constraint file:

```
define_global_attribute syn_auto_insert_bufg {0}
```

When buffer insertion is globally disabled, attach a `syn_insert_buffer` attribute with a value of 1 to the net to allow buffer insertion on that net.

# FX1040

## **@E: Unable to assign RTL Pin <pin> to HAPS I/O <port>, pin is already assigned to <port>**

This message occurs when a previously assigned pin is assigned to a second port.

## Action

A pin cannot be assigned to more than one port. Check the constraint file for multiple `define_haps_io` commands with the same pin assignment. Assign only one port per pin.

# FX1041

## **@E: Found <number> HAPS I/O errors -- aborting**

One or more errors occurred during pre-mapping. Pre-mapping continued until all `define_haps_io` commands are processed, then aborted.

**Action**

Review errors in the premap log file. Correct all errors and rerun project.

## FX1052

**@E: Instance <instance> is an internal tristate. Selected technology does not include support for mapping internal tristate.**

Selected technology does not have a primitive that supports mapping for internal tristates.

**Action**

Use an equivalent primitive that supports mapping for internal tristates.

## FX1065

**@E: ECC is not supported for targeted technology.**

You specified syn\_ramstyle=ecc in the constraint file or RTL source code for a technology that does not support ECC mode.

**Action**

Make sure the technology specified supports ECC mode for RAM.

## FX1072

### **@E: Selected technology part does not support syn\_ramstyle = distributed\_ram on ram instances**

This error can occur when you specify syn\_ramstyle=distributed\_ram in the constraint file or RTL source code for a technology that does not support logic RAM or distributed RAM for the part.

#### Action

Do not specify syn\_ramstyle=distributed\_ram or select a valid technology part.

## FX1073

### **@E: Selected technology part does not support syn\_romstyle = distributed on rom instances**

This error can occur when you specify syn\_romstyle=distributed in the constraint file or RTL source code for a technology that does not support logic ROM or distributed ROM for the part.

#### Action

Do not specify syn\_romstyle=distributed or select a valid technology part.

## FX1076

**@E: Pin "COUT" of GTP\_LUT5CARRY instance <instanceName> can only connect to a "CIN" pin of a GTP\_LUT5CARRY when pin "Z" is in use.**

This error can occur when the software encounters a design rule check (DRC) violation for the instantiated GTP\_LUT5CARRY primitive. The DRC violation can happen if both outputs for the COUT and Z pins are used (i.e., they are required to drive other logic), then the COUT pin must drive or be connected to CIN of another GTP\_LUT5CARRY primitive when the Z pin is used (i.e., Z pin is driving logic).

### Action

Fix the DRC violation and rerun synthesis.

## FX1079

**@E: Target technology supports ECC only on two-port RAM**

The technology you selected only supports `syn_ramstyle=ecc` mode for simple dual-port RAM. This error can occur when you specify ECC mode on a single-port or true dual-port RAM.

### Action

Do not specify ECC mode for the RAM.

## FX1080

### **@E: Target technology does not support ECC on URAM**

This error can occur when you specify `syn_ramstyle=ecc` on a logic RAM with asynchronous read. The technology you specified does support ECC mode on URAM.

#### Action

Do not specify ECC mode for the RAM.

## FX1084

### **@W: RAM <instanceName> has asynchronous read/write clocks and hence read/write conflict check is NOT added. Possible simulation mismatch. To insert rw conflict check for this RAM, set `syn_ramstyle = rw_check`. For more information, search for “read/write conflict check” in Online Help.**

The identified RAM includes asynchronous read/write clocks which, by definition, prevent a read/write conflict check from being applied. As a result, a simulation mismatch is possible.

#### Action

To insert bypass logic around the RAM, set `syn_ramstyle = rw_check` on the instance. Inserting this logic, however, can cause unwanted clock-domain crossings as the RAM has read and write clocks that belong to asynchronous clock groups. For more information, search for “`syn_ramstyle`” in Online Help.

## FX1086

### **@E: Illegal value of CLKHF\_DIV parameter for Oscillator primitive <primitiveName>**

The SB\_HFOSC oscillator primitive was instantiated with an illegal value specified for the CLKHF\_DIV parameter.

#### Action

Make sure the CLKHF\_DIV parameter is specified correctly. CLKHF\_DIV supports the values: 0b00, 0b01, 0b10, and 0b11.

## FX1087

### **@E: Illegal value of I2C\_CLK\_DIVIDER parameter for SB\_I2C primitive <primitiveName>**

The SB\_I2C primitive was instantiated with an illegal value specified for the I2C\_CLK\_DIVIDER parameter.

#### Action

Make sure the I2C\_CLK\_DIVIDER parameter is specified correctly. I2C\_CLK\_DIVIDER supports values from 1 to 1023.

## FX1088

### **@E: Illegal value of I2C\_CLK\_DIVIDER parameter for SB\_I2C\_FIFO primitive <primitiveName>**

The SB\_I2C\_FIFO primitive was instantiated with an illegal value specified for the I2C\_CLK\_DIVIDER parameter.

## Action

Make sure the I2C\_CLK\_DIVIDER parameter is specified correctly.  
I2C\_CLK\_DIVIDER supports values from 1 to 1023.

# FX1089

### **@E: Illegal value of SPI\_CLK\_DIVIDER parameter for SB\_SPI primitive <primitiveName>**

The SB\_SPI primitive was instantiated with an illegal value specified for the SPI\_CLK\_DIVIDER parameter.

## Action

Make sure the SPI\_CLK\_DIVIDER parameter is specified correctly.  
SPI\_CLK\_DIVIDER supports values from 1 to 1023.

# FX1095

### **@E: Selected technology part does not support syn\_ramstyle=distributed\_ram on RAM instance.**

This error can occur when you specify syn\_ramstyle=distributed\_ram in the constraint file or RTL source code for a technology that does not support distributed RAM for the part.

## Action

Do not specify syn\_ramstyle=distributed\_ram or select a valid technology part.

## FX1096

### **@E: Selected technology part does not support syn\_romstyle=distributed\_rom on ROM instance.**

This error can occur when you specify syn\_romstyle=distributed\_rom in the constraint file or RTL source code for a technology that does not support distributed ROM for the part.

#### Action

Do not specify syn\_romstyle=distributed\_rom or select a valid technology part.

## FX1109

**@E: syn\_allowed\_resources sets "blockrams" to <memValue>, which is more than twice the chip resources of <module>.**

The value of the syn\_allowed\_resources attribute is set to a memory value that is more than twice the capacity of the specified top-level module or compile point.

### Action

Check the value of the syn\_allowed\_resources attribute in the FDC file or attached to the module and make sure that the value specified is less than twice the available resource limit.

## FX1110

**@E: syn\_allowed\_resources sets dsps to <blockValue>, which is more than twice the chip resources <module>.**

The value of the syn\_allowed\_resources attribute is set to a block value that is more than twice the capacity of the specified top-level module or compile point.

### Action

Check the value of the syn\_allowed\_resources attribute in the FDC file or attached to the module and make sure that the value specified is less than twice the available resource limit.

## FX1111

**@E: \*LUT usage is <integer>, which is more than twice the chip resources of <module>.**

The number of LUTs (Look Up Tables) required is more than double the available LUT resources.

### Action

Try balancing the utilization by mapping RAMs to BRAMs, SRLs to registers, and arithmetic operators to DSP blocks to free-up additional LUT resources. As an alternative, try re-partitioning the design.

If this error occurs in ProtoCompiler, consider setting an upper (-max\_ratio) bound limit using the dissolve\_control attribute for the named module.

## FX1122

**@E: Cannot map asymmetric ram <instanceName>, set attribute syn\_ramstyle=no\_asymmetric\_ram on instance to disable asymmetric ram mapping."**

This error can occur when there is an asymmetric ram in the design that cannot map to block rams.

### Action

Set attribute syn\_ramstyle to no\_asymmetric\_ram on instance.

## FX1183

**@W: User-specified initial value set for instance <instance> cannot be supported due to limitations in architecture. Please remove the initial value set on the instance to avoid the warning.**

This warning occurs when RTL code with an initial value is written to a shift register.

### Action

User should modify the RTL code to not set an initial value on the sequential shift register. If the user does not modify it, the software will drop the initial value set on the instance.

## CHAPTER 27

# MF Messages 100 – 986

---

## MF100

**@N: FSM instance <*machine* [0:2]> has user-specified encoding style; no exploration performed.**

While running the FSM Explorer, a state-machine extracted by the FSM Compiler was encountered that has a fixed encoding style set by a syn\_encoding attribute (without the attribute, the FSM Explorer runs through each encoding style for each state machine and selects the best encoding style).

In the following test case, the FSM Explorer is enabled (set\_option -use\_fsm\_explorer 1 in project file) and a syn\_encoding attribute is applied to the state machine with a value of gray from a constraint file (define\_attribute {machine[0:2]} syn\_encoding {gray}).

```
library ieee;
use ieee.std_logic_1164.all;

entity shift_enum is
 port (clk, rst : bit;
 O : out std_logic_vector(2 downto 0));
end shift_enum;
```

```
architecture behave of shift_enum is
type state_type is (S0, S1, S2);
attribute syn_enum_encoding: string;
attribute syn_enum_encoding of state_type : type is "001 010 101";
signal machine : state_type;
begin
 process (clk, rst)
 begin
 if rst = '1' then
 machine <= S0;
 elsif clk = '1' and clk'event then
 case machine is
 when S0 => machine <= S1;
 when S1 => machine <= S2;
 when S2 => machine <= S0;
 end case;
 end if;
 end process;

 with machine select
 O <= "001" when S0,
 "010" when S1,
 "101" when S2;

end behave;
```

## Action

To enable the FSM Explorer to select the best encoding style for a state machine, do not use the `syn_encoding` attribute. If you want a state machine to be implemented with a specific encoding style, use the `syn_encoding` attribute with an appropriate value.

When the FSM Explorer is run in the absence of a `syn_encoding` attribute, it finds the best encoding style for the state machine and adds an entry for this style to the `filename_fsm.sdc` file (in the revision directory). If you rerun the FSM Explorer, no further exploration is necessary because the selection information, present in the `filename_fsm.sdc` file, is used during subsequent mapper runs. To update the FSM Explorer results in the `filename_fsm.sdc` file, select Run->FSM Explorer.

# MF104

## @N: Found compile point of type <*locked*> on View <*view:work.inc(verilog)*>

During multipoint synthesis, indicates that one or more modules are declared as compile points. The note indicates the compile-point name and source-code type. In the test case below, module inc is declared as a compile point in the constraint file which results in the above note.

```
Constraint file contains the following Compile Points:
define_compile_point -comment {Automatically Inserted}
{v:work.inc} -type {locked}

module inc(a_in, a_out) ;
 input [3:0] a_in;
 output [3:0] a_out;
 assign a_out = a_in + 1;
endmodule

module reg4(clk, rst, d, q);
 input [3:0] d;
 input clk, rst;
 output [3:0] q;
 reg [3:0] q;

 always @(posedge clk or posedge rst)
 if(rst)
 q <= 0;
 else
 q <= d;
 endmodule

module test_syn_hier(clk, rst, q);
 input clk, rst;
 output [3:0] q;
 wire [3:0] a_in;
 inc i1(q, a_in);
 reg4 r1(clk, rst, a_in, q);
endmodule
```

## Action

Indicates all of the compile points in the design. Note that these compile points will not be resynthesized until:

- You change the HDL source code that defines the compile point in such a way that the design logic is changed.
- You change the constraints applied to the compile point.
- You change any of the options on the Device panel of the Options for implementation dialog box (except Update Compile Point Timing Data) in which case the entire design is resynthesized, including all compile points.
- You intentionally force the resynthesis of your entire design, including all compile points, by selecting Run->Resynthesize All on the Run menu.
- The Update Compile Point Timing Data device mapping option is enabled and at least one child of the compile point (at any level) has been remapped. The option requires that the parent compile point be resynthesized using the updated timing model of the child.

## MF105

**@N: Performing bottom up mapping of Top level  
<view:work.eight\_bit\_uc>(structural)**

When there are compile points specified for a design, bottom up mapping takes place for mapping the top-level design. Modules with compile points are initially synthesized before they are mapped at the top level.

### Action

Informative message; no action required.

## MF106

**@N: Mapping <Top level><view:work.test\_syn\_hier(verilog)>  
because**

The mapper remapped the top-level module because of:

- a change in the HDL code at the top level
- any change in the constraints
- a change in a compile point's constraint file
- a change in the HDL code that results in the remapping of the compile point as well as the top level

For the following test case, include the following in the top-level FDC file:

```
define_attribute {v:work.reg4} syn_hier {flatten}
define_compile_point -comment {Automatically Inserted}
{v:work.reg4} -type {locked}
```

### Test Case:

```
module inc(a_in, a_out) ;
 input [3:0] a_in;
 output [3:0] a_out;
 assign a_out = a_in + 1;
endmodule

module reg4(clk, rst, d, q);
 input [3:0] d;
 input clk, rst;
 output [3:0] q;
 reg [3:0] q;

 always @ (posedge clk or posedge rst)
 if(rst)
 q <= 0;
 else
 q <= d;
endmodule

module test_syn_hier(clk, rst, q);
 input clk, rst;
 output [3:0] q;
 wire [3:0] a_in;
 inc i1(q, a_in);
 reg4 r1(clk, rst, a_in, q);
endmodule
```

## Action

Either the top-level module or both the top-level and compile-point modules are being remapped. This note is informative; no action is required.

# MF107

### **@N: Old database up-to-date, remapping <*Top level*> <*view:work*> unnecessary**

During multi-point synthesis, a project was run a second time and indicated that no compile-point data changed from the initial run. The following test case, when run for a second time with the following constraints (which declared module sub\_4s as a compile point), resulted in the above note.

```
Compile Points
define_compile_point -comment {Automatically Inserted} {v:sub_4s}
-type {soft}

module top (q,a,b,c,d);
parameter top_param = 4;
output [top_param:0] q;
input [top_param:0] a,b,c,d;
sub u2 (q,a,b,c,d);
defparam u2.param=4;
endmodule

module sub (q,a,b,c,d);
parameter param = 4;
output [param:0] q;
input [param:0] a,b,c,d;
assign q = a & b & c & d;
endmodule
```

## Action

To force the entire design to be resynthesized, select Run->Resynthesize All on the Run menu.

## MF108

### **@N: Updating model for compile point <*view:work.ALU(first)*>**

The status of the compile-point module has changed (module recompiled or modified constraint file) and it was remapped.

#### Action

Informative message; no action required.

## MF111

### **@W: Compile points not supported for this device; removing compile point from <*view : work.Module\_1(rtl)*>**

MultiPoint flow was used with an unsupported technology. The compile point directive was ignored, and the mapper removed the compile point before mapping the entire design.

#### Action

Make sure that the MultiPoint flow is used only with supported technologies.

## MF112

### **@W: Provide a constraint file for compile point <*compile\_point\_name*>**

MultiPoint flow was used and no constraint file was present for the compile-point module. The synthesis tool does not propagate timing constraints through a hierarchical nested compile point; a child compile point does not inherit the constraint file of the parent, nor does the parent inherit the constraint file of the child.

## Action

Add an accurate timing model (time budgeting) for each constraint file associated with each compile-point module. Add reasonably accurate timing constraints for each compile point in your design.

# MF121

### **@W: No initial value for dff <q> in MULTACCUM.**

The mapper inferred a MAC module or DSP block for registered multiplier logic and the register inside the MAC module was not initialized. The following test case generates the above warning for flip-flop q.

```
module warn (a, b, q, clk);
 input [7:0] a, b;
 input clk;
 output [16:0] q;
 reg [16:0] q;

 always @(posedge clk)
 begin
 q <= q + a*b;
 end

endmodule
```

## Action

To implement the registered multiplier logic inside a MAC, you must asynchronously initialize the register. Edit the code as shown in the corrected test case below.

```
module warn (a, b, q, clk);
 input [7:0] a, b;
 input clk;
 output [16:0] q;
 reg [16:0] q;
```

```

always @(posedge clk or negedge reset)
begin
 if (!reset) then
 q <= 1'b0;
 else
 q <= q + a*b;
end
endmodule

```

Alternatively, you can force the mapper to implement the multiplier in lpm\_mult or as logic using the syn\_multstyle attribute on the multiplier as shown below. These implementations prevent the synthesis tool from using the MAC module.

```
define_attribute {unl_a[15:0]} syn_multstyle {lpm_mult}
```

## MF122

### **@N: Added probe <tmp1\_probe\_1[2]> on <tmp1[2]> in <testprobe>**

A syn\_probe attribute was applied to the desired point and the name of the probe are displayed. The probe name depends on the value specification of the attribute. For details of how probe port names are assigned, see help on the syn\_probe attribute. This note is accompanied by [MF123](#) and [MF124](#).

In the following example, a syn\_probe attribute is applied to net tmp1[2:0] with a value of 1 which causes the probe to be named according to the net name.

```

module testprobe (a, b, c, d, x, y, z, clk, rst, f1, f2, q);
 input a, b, c, d, x, y, z;
 input clk, rst;
 output f1, f2, q;
 reg q;
 wire [2:0]tmp1 /* synthesis syn_probe= 1 */;
 wire tmp2;
 assign tmp1 = (a & b) | (c & d);
 assign tmp2 = (x | y) & z;
 assign f1= tmp1 * tmp2;
 assign f2= tmp2 + tmp1;

```

```

always@(posedge clk or posedge rst)
begin
 if (rst)
 q <= 0;
 else
 q <= (f1 & f2);
end
endmodule

```

## Action

Because probes are used as debugging aids, add probes only to points that are critical. Remember that probes count in the overall pin count and that they also require pads.

# MF123

### **@N: Also padding probe <tmp1\_probe\_3[0]>**

All of the probes in the design were padded with buffers. This note is accompanied by [MF122](#) and [MF124](#).

In the test case below, the syn\_probe attribute is applied to net tmp1[2:0] and the corresponding probes are implemented with buffers.

```

module probe (a, b, c, d, x, y, z, clk, rst, f1, f2, q);
 input a, b, c, d, x, y, z;
 input clk, rst;
 output f1, f2, q;
 reg q;
 wire [2:0]tmp1 /* synthesis syn_probe= 1 */;
 wire tmp2;
 assign tmp1 = (a & b) | (c & d);
 assign tmp2 = (x | y) & z;
 assign f1= tmp1 * tmp2;
 assign f2= tmp2 + tmp1;

```

```
always@(posedge clk or posedge rst)
begin
 if (rst)
 q <= 0;
 else
 q <= (f1 & f2);
end
endmodule
```

## Action

Because probes are used as debugging aids, add probes only to points that are critical. Remember that probes count in the overall pin count and that they also require pads.

# MF124

### **@N: Total <3> probes and <3> pads inserted.**

Identifies the total number of probes and pads inserted in the design as a result of the `syn_probe` attribute. This note is accompanied by [MF122](#) and [MF123](#).

In the test case below, the `syn_probe` attribute is applied to vectored net `tmp1[2:0]` which results in the assignment of three probes, each with one pad.

```
module probe (a, b, c, d, x, y, z, clk, rst, f1, f2, q);
 input a, b, c, d, x, y, z;
 input clk, rst;
 output f1, f2, q;
 reg q;
 wire [2:0]tmp1 /* synthesis syn_probe= 1 */;
 wire tmp2;
 assign tmp1 = (a & b) | (c & d);
 assign tmp2 = (x | y) & z;
 assign f1= tmp1 * tmp2;
 assign f2= tmp2 + tmp1;
```

```

always@(posedge clk or posedge rst)
begin
 if (rst)
 q <= 0;
 else
 q <= (f1 & f2);
end
endmodule

```

## Action

Because probes are used as debugging aids, add probes only to points that are critical. Remember that probes count in the overall pin count and that they also require pads.

# MF125

### **@W: Probe name '<a>' already in use**

When the `syn_probe` attribute was applied to the entire bus of a signal, the mapper generated the above warning message.

The test case below shows signal `tmp1` consisting of three bits assigned the `syn_probe` attribute. The mapper applies the `syn_probe` attribute on each bit of this signal separately with the specified name usually under scoring the bit number of the probe. In this case, the mapper generates a probe for each of the bits as follows: `a_0`, `a_1`, and `a_2`.

```

module probe (a, b, c, d, x, y, z, clk, rst, f1, f2, q);
 input a, b, c, d, x, y, z;
 input clk, rst;
 output f1, f2, q;
 reg q;
 wire [2:0]tmp1 /* synthesis syn_probe="a" */;
 wire tmp2;
 assign tmp1 = (a & b) | (c & d);
 assign tmp2 = (x | y) & z;
 assign f1= tmp1 * tmp2;
 assign f2= tmp2 + tmp1;

```

```
always@(posedge clk or posedge rst)
begin
if (rst)
 q <= 0;
else
 q <= (f1 & f2);
end

endmodule
```

## Action

Apply the `syn_probe` attribute on the signal as shown in the corrected test case below.

```
module probe (a, b, c, d, x, y, z, clk, rst, f1, f2, q);
input a, b, c, d, x, y, z;
input clk, rst;
output f1, f2, q;
reg q;
wire [2:0]tmp1 /* synthesis syn_probe= 1 */;
wire tmp2;
assign tmp1 = (a & b) | (c & d);
assign tmp2 = (x | y) & z;
assign f1= tmp1 * tmp2;
assign f2= tmp2 + tmp1;

always@(posedge clk or posedge rst)
begin
if (rst)
 q <= 0;
else
 q <= (f1 & f2);
end

endmodule
```

# MF126

## **@W: Probe name '</alu2[]>' must begin with an alphabet.**

The quoted string entered as the syn\_probe attribute value did not begin with an alphabetic character (an unquoted numeric value of 1 or 0 is also acceptable). In the test case below, the string value assigned to the syn\_probe attribute begins with a ‘/’ (slash).

```
module alu(out1, opcode, clk, a, b, sel);
 output [7:0] out1;
 input [2:0] opcode;
 input [7:0] a, b;
 input clk, sel;
 reg [7:0] alu_tmp /* synthesis syn_probe="/alu2[]" */;
 reg [7:0] out1;

 always @(opcode or a or b or sel)
 begin
 case (opcode)
 3'b000:alu_tmp <= a+b;
 3'b010:alu_tmp <= a-b;
 3'b011:alu_tmp <= a^b;
 3'b100:alu_tmp <= sel ? a:b;
 default:alu_tmp <= a|b;
 endcase
 end

 always @(posedge clk)
 out1 <= alu_tmp;

endmodule
```

## Action

Make sure that the string value entered for the syn\_probe attribute begins with an alphabetic character as shown in the corrected test case below.

```
module alu(out1, opcode, clk, a, b, sel);
 output [7:0] out1;
 input [2:0] opcode;
 input [7:0] a, b;
 input clk, sel;
 reg [7:0] alu_tmp /* synthesis syn_probe="alu2[]" */;
 reg [7:0] out1;
```

```

always @(opcode or a or b or sel)
begin
 case (opcode)
 3'b000:alu_tmp <= a+b;
 3'b010:alu_tmp <= a-b;
 3'b011:alu_tmp <= a^b;
 3'b100:alu_tmp <= sel ? a:b;
 default:alu_tmp <= a|b;
 endcase
end

always @(posedge clk)
 out1 <= alu_tmp;

endmodule

```

## MF127

### **@W: Probe name '<test\_probe[ ]>' must be alpha\_numeric.**

The string value for a syn\_probe attribute included a non-alphanumeric character other than an underscore or the square bracket pair ([] ) that identifies a vector value. In the test case below, the value string includes a period (.) which results in the above warning.

```

module alu(out1, opcode, clk, a, b, sel);
output [7:0] out1;
input [2:0] opcode;
input [7:0] a, b;
input clk, sel;
reg [7:0] alu_tmp /* synthesis syn_probe="test_probe[]." */;
reg [7:0] out1;

always @(opcode or a or b or sel)
begin
 case (opcode)
 3'b000:alu_tmp <= a+b;
 3'b010:alu_tmp <= a-b;
 3'b011:alu_tmp <= a^b;
 3'b100:alu_tmp <= sel ? a:b;
 default:alu_tmp <= a|b;
 endcase
end

```

```

 always @(posedge clk)
 out1 <= alu_tmp;

 endmodule

```

## Action

Make sure that the string value entered for the `syn_probe` attribute includes only alphanumeric characters (or the underscore or square bracket pair) as shown in the corrected test case below.

```

module alu(out1, opcode, clk, a, b, sel);
 output [7:0] out1;
 input [2:0] opcode;
 input [7:0] a, b;
 input clk, sel;
 reg [7:0] alu_tmp /* synthesis syn_probe="test_probe[]" */;
 reg [7:0] out1;

 always @(opcode or a or b or sel)
 begin
 case (opcode)
 3'b000:alu_tmp <= a+b;
 3'b010:alu_tmp <= a-b;
 3'b011:alu_tmp <= a^b;
 3'b100:alu_tmp <= sel ? a:b;
 default:alu_tmp <= a|b;
 endcase
 end

 always @(posedge clk)
 out1 <= alu_tmp;

endmodule

```

## MF130

### **@W: You can only probe nets (keepbuf), <un1\_count[2]> not a net**

When the `syn_probe` attribute was applied to any object other than a net, the mapper generated the above warning.

In the following Verilog test case, the `syn_probe` attribute is applied to register `qa` rather than to a net. In this case, the mapper creates a probe on the net at the output of register `qa` and gives the above warning message.

```
module reg_reg (a, clk, data, qb);
 input clk;
 input a, data;
 output qb;
 wire tmp;
 reg qa /* synthesis syn_probe=1 */;
 reg qb;
 assign tmp = data & a;

 always@ (posedge clk)
 qb <= qa;
 always@ (posedge clk)
 qa <= tmp;

endmodule
```

## Action

Apply the `syn_probe` attribute only to a net. To eliminate this warning in the above test case, apply the `syn_probe` attribute on the wire signal `tmp` instead of register `qa` as shown in the corrected test case below.

```
module reg_reg (a, clk, data, qb);
 input clk;
 input a, data;
 output qb;
 wire tmp /* synthesis syn_probe=1 */;
 reg qa;
 reg qb;
 assign tmp = data & a;

 always@ (posedge clk)
 qb <= qa;
 always@ (posedge clk)
 qa <= tmp;

endmodule
```

# MF132

**@W: Numeric probe name '<5>' required name adjustment to '<q1\_keep\_probe\_1>'.**

The probe name in a syn\_probe attribute did not begin with an alpha character (the probe was automatically renamed by the mapper as indicated in the message). For the test case below, the probe name specified by the syn\_probe attribute in the constraint file is 5 which results in the above message.

## Constraint File Content

```
define_attribute {n:q1} syn_probe {5}
```

## Test Case

```
module test (d1,d2,t,clk,q);
 input d1,clk,t;
 inout d2;
 output reg q;
 wire temp;
 IOBUF u1 (temp,d2,d1,t);

 always@(posedge clk)
 q <= temp;
endmodule

module test1 (clk, rst, d1,d2,q);
 input d1,d2;
 input clk, rst;
 output reg q;
 reg q1,q2;

 always @(posedge clk or posedge rst)
begin
 if (rst) begin
 q <= 1'b0;
 q1 <= 1'b1;
 q2 <= 1'b0;
 end
 else begin
 q1 <= d1;
```

```
 q2 <= d2;
 q <= q1 ^ q2;
 end
end
endmodule
```

## Action

To avoid random probe name assignment and the above message, make sure that all probe names begin with an alphabetic character. For additional information on probe naming conventions, see the topic `syn_probe` in online help.

# MF133

### **@W: You used `syn_probe` both as boolean and as string**

The `syn_probe` attribute was applied as both a boolean and as a string value in the same module. In the test case below, the `syn_probe` attribute is applied to signal `tmp1` as a string value and to signal `tmp2` as a boolean value. In this case, the mapper honors the `syn_probe` attribute on both signals and generates the above warning.

```
module probe (a, b, c, d, x, y, z, clk, rst, f1, f2, q);
 input a, b, c, d, x, y, z;
 input clk, rst;
 output f1, f2, q;
 reg q;
 wire tmp1 /* synthesis syn_probe="xyz" */;
 wire tmp2 /* synthesis syn_probe=1 */;
 assign tmp1 = (a & b) | (c & d);
 assign tmp2 = (x | y) & z;
 assign f1= tmp1 * tmp2;
 assign f2= tmp2 + tmp1;

 always@(posedge clk or posedge rst)
 begin
 if (rst)
 q <= 0;
 else
 q<= (f1 & f2);
 end
```

```
endmodule
```

## Action

Use either a string or a boolean value for the `syn_probe` attribute.

```
module probe (a, b, c, d, x, y, z, clk, rst, f1, f2, q);
 input a, b, c, d, x, y, z;
 input clk, rst;
 output f1, f2, q;
 reg q;
 wire tmp1
 wire tmp2 /* synthesis syn_probe=1 */;
 assign tmp1 = (a & b) | (c & d);
 assign tmp2 = (x | y) & z;
 assign f1= tmp1 * tmp2;
 assign f2= tmp2 + tmp1;

 always@(posedge clk or posedge rst)
 begin
 if (rst)
 q <= 0;
 else
 q<= (f1 & f2);
 end
endmodule
```

# MF139

## @W: Rom <z\_1[3:0]> won't pack to blockrom, using logic

For certain devices, ROMs were implemented using the device's RAM resources. To implement ROM in block ram, the reading process must be synchronous. The ROM is forced to be mapped to RAM by the `syn_romstyle` attribute but, because reading is not synchronous to a clock, the above warning is issued.

```
module rom(z,a);
 output [3:0] z;
 input [4:0] a;
 reg [3:0] z /* synthesis syn_romstyle = "block_rom" */;
```

```

always @(a) begin
 case (a)
 5'b00000: z = 4'b1011;
 5'b00001: z = 4'b0001;
 5'b00100: z = 4'b0011;
 5'b00110: z = 4'b0010;
 5'b00111: z = 4'b1110;
 5'b01001: z = 4'b0111;
 5'b01010: z = 4'b0101;
 5'b01101: z = 4'b0100;
 5'b10000: z = 4'b1100;
 5'b10001: z = 4'b1101;
 5'b10010: z = 4'b1111;
 5'b10011: z = 4'b1110;
 5'b11000: z = 4'b1010;
 5'b11010: z = 4'b1011;
 5'b11110: z = 4'b1001;
 5'b11111: z = 4'b1000;
 default: z = 4'b0000;
 endcase
end
endmodule

```

## Action

Make sure that the reading process is synchronous to a clock when a block\_rom attribute value is applied. In the corrected test case below, reading is made synchronous to the clock which correctly maps the ROM into block RAM and eliminates the warning.

```

module rom(z,a,clk);
output [3:0] z;
input [4:0] a;
input clk;
reg [3:0] z /* synthesis syn_romstyle = "block_rom" */;

always @(a) begin
 case (a)
 5'b00000: z = 4'b1011;
 5'b00001: z = 4'b0001;
 5'b00100: z = 4'b0011;
 5'b00110: z = 4'b0010;
 5'b00111: z = 4'b1110;
 5'b01001: z = 4'b0111;
 5'b01010: z = 4'b0101;
 5'b01101: z = 4'b0100;
 endcase
end
endmodule

```

```

5'b10000: z = 4'b1100;
5'b10001: z = 4'b1101;
5'b10010: z = 4'b1111;
5'b10011: z = 4'b1110;
5'b11000: z = 4'b1010;
5'b11010: z = 4'b1011;
5'b11110: z = 4'b1001;
5'b11111: z = 4'b1000;
default: z = 4'b0000;
endcase
end
endmodule

```

## MF149

### **@N: <out1> is level <1> of the pipelined module <out1\_3>**

Indicates the level of pipelining for a register. This note is mostly seen in devices that do not have MULT blocks. In the example below, output register out1 is pipelined directly after the multiplier (i.e., the first level), and the mapper issues the above note indicating the first level. If there had been another register storing multiplication and feeding output register out1, then out1 would be reported as the second pipelining level.

```

module test(a, b,c,d,out,out1,clk,rst);
input [9:0]a,b,c,d;
input clk,rst;
output [19:0]out,out1;
reg [9:0]a_reg,b_reg,c_reg,d_reg;
reg [19:0]out /*synthesis syn_pipeline =0*/;
reg [19:0]out1 /* synthesis syn_pipeline=1 */;

always @(negedge clk)
begin
 a_reg<=a;
 b_reg<=b;
 c_reg<=c;
 d_reg<=d;
end

```

```
always @(negedge clk)
begin
 if(!rst)
 out<=20'b0;
 else
 out <= a_reg*b_reg;
end

always @(negedge clk)
begin
 if(!rst)
 out1<=20'b0;
 else
 out1 <= c_reg*d_reg;
end

endmodule
```

## Action

Informative message; no action required.

# MF150

### @N: Pipelining module <out1\_3>

Indicates that the mapper encountered a pipelined register. This note is mostly seen in devices that do not have MULT blocks. In the following example, register out1 is pipelined as indicated by the above message. In the message, the corresponding multiplier is referred to as a module. Pipelining is controlled by the syn\_pipeline attribute as shown in the test case below.

```
module test(a, b,c,d,out,out1,clk,rst);
 input [9:0]a,b,c,d;
 input clk,rst;
 output [19:0]out,out1;
 reg [9:0]a_reg,b_reg,c_reg,d_reg;
 reg [19:0]out /*synthesis syn_pipeline=0*/;
 reg [19:0]out1 /* synthesis syn_pipeline=1 */;
```

```
always @(negedge clk)
begin
 a_reg<=a;
 b_reg<=b;
 c_reg<=c;
 d_reg<=d;
end

always @(negedge clk)
begin
 if(!rst)
 out<=20'b0;
 else
 out <= a_reg*b_reg;
end

always @(negedge clk)
begin
 if(!rst)
 out1<=20'b0;
 else
 out1 <= c_reg*d_reg;
end

endmodule
```

## Action

Informative message; no action required.

# MF151

### **@N: The module <*moduleName*> is too small to be pipelined.**

The size of a multiplier was too small to be pipelined (i.e., only logic exists between the two perspective registers with no scope for pipelining). The test case below displays the above message when the multiplier is 2x2 so that the logic between input registers a<sub>\_reg</sub> and b<sub>\_reg</sub> and the output registers fits within 1 LUT for each pair as shown in the Technology view.

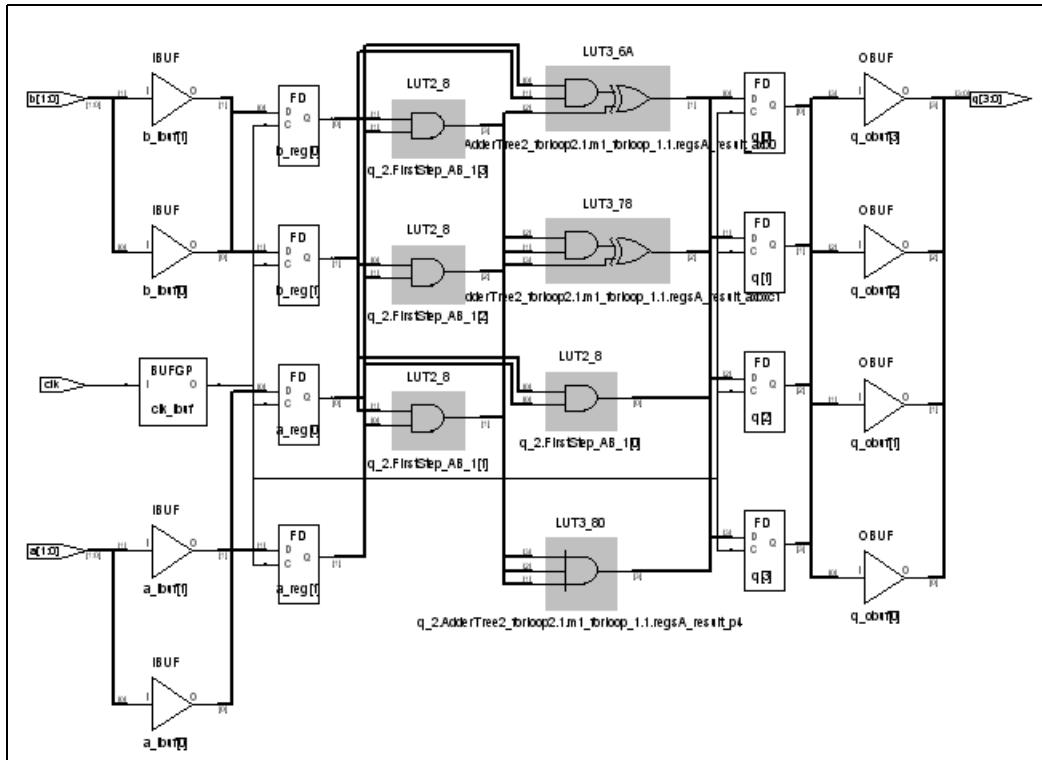
```

module test(a,b,clk,q);
parameter k =1;
input [k:0] a,b;
input clk;
output reg [2 * k + 1:0] q ;
reg [k:0] a_reg, b_reg;

always @(posedge clk)
begin
 a_reg<=a;
 b_reg<=b;
 q<= a_reg * b_reg;
end
endmodule

```

## Technology View



## Action

If you want to perform pipelining, the multiplier inputs must be at least three bits wide as shown in the test case below.

```
module test(a,b,clk,q);
parameter k =2;
input [k:0] a,b;
input clk;
output reg [2 * k + 1:0] q ;
reg [k:0] a_reg, b_reg;

always @(posedge clk)
begin
 a_reg<=a;
 b_reg<=b;
 q<= a_reg * b_reg;
end
endmodule
```

# MF152

### **@N: Need the pro-license to use the pipelining option.**

Indicates that a `syn_pipeline` attribute was encountered either in the HDL code or in the constraint file and that the pipelining feature was not available in the synthesis tool. Running the test case below in a Synplify tool, that does not support pipelining results in the above message.

```
module test(a,b,clk,q);
input [1:0] a,b;
input clk;
output reg [3:0] q /* synthesis syn_pipeline = 1 */;
reg [1:0] temp1,temp2;

always @(posedge clk)
begin
 temp1<=a;
 temp2<=b;
 q<= temp1 * temp2;
end
endmodule
```

## Action

Informative message; no action required. To use the pipelining feature, you must have a Synplify Pro license.

# MF155

### **@W: Retiming overrides Syn\_pipeline = 0; Pipelining Register <out[0]>**

This warning results if the constraint file has attributes syn\_pipeline = 0 and the retiming option is enabled in the project view. This is not a legal combination. The only possible legal combinations of syn\_pipeline and Retiming are:

```
Retiming = 0 and syn_pipeline = 0
Retiming = 0 and syn_pipeline = 1
Retiming = 1 and syn_pipeline = 1
```

When retiming is enabled in the project view user interface, syn\_pipeline is automatically enabled. The distinction between retiming and pipelining is that pipelining moves registers backwards into certain operators (for example, multipliers), while retiming moves registers in either direction across any combinational logic. The following test case and attribute result in this warning:

```
module mult (out, clk, ina, inb);
parameter width = 4;
input [width -1:0] ina, inb;
input clk;
output [width*2 -1:0] out;
reg [width*2 -1:0] out;

always @(posedge clk)
 out = ina * inb;
endmodule
```

#### **Constraint File Disables Pipelining**

```
define_attribute {i:out[7:0]} syn_pipeline {0}
```

#### **Retiming Set in Project View (.prj)**

```
set_option -retiming 1
```

## Action

Use a legal combination of retiming and pipelining. To eliminate the warning in the above test case, change the constraint file to enable pipelining.

```
module mult (out, clk, ina, inb);
parameter width = 4;
input [width -1:0] ina, inb;
input clk;
output [width*2 -1:0] out;
reg [width*2 -1:0] out;

always @(posedge clk)
 out = ina * inb;
endmodule
```

## Constraint File Enables Pipelining

```
define_attribute {i:out[7:0]} syn_pipeline {1}
```

## Retiming Set in Project View

```
.prj file : set_option -retiming 1
```

# MF169

## @N: Pushed in register <registerName>.

Identifies the registers that were pushed in for pipelining. In the test case below, output register q is pipelined which results in the above message.

In the test case, global pipelining is disabled (Pipelining option unchecked on the Device panel).

```
module mult (a,b, q, clk);
input [1:0] a,b;
input clk;
output [3:0] q;
reg [3:0]q /* synthesis syn_pipeline = 1 */;
reg [1:0]a_reg,b_reg;
```

```

always @(posedge clk)
begin
 a_reg <= a;
 b_reg <= b;
 q <= a_reg*b_reg;
end
endmodule

```

## Action

Pipelining is controlled by the `syn_pipeline` attribute. To disable pipelining on an individual register basis, set the value of `syn_pipeline` to 0 as shown in the test case below.

```

module mult (a,b, q, clk);
 input [1:0] a,b;
 input clk;
 output [3:0] q;
 reg [3:0]q /* synthesis syn_pipeline = 0 */;
 reg [1:0]a_reg,b_reg;

 always @(posedge clk)
 begin
 a_reg <= a;
 b_reg <= b;
 q <= a_reg*b_reg;
 end
endmodule

```

# MF170

## **@W: Register <name> not retimed because of false-path constraints.**

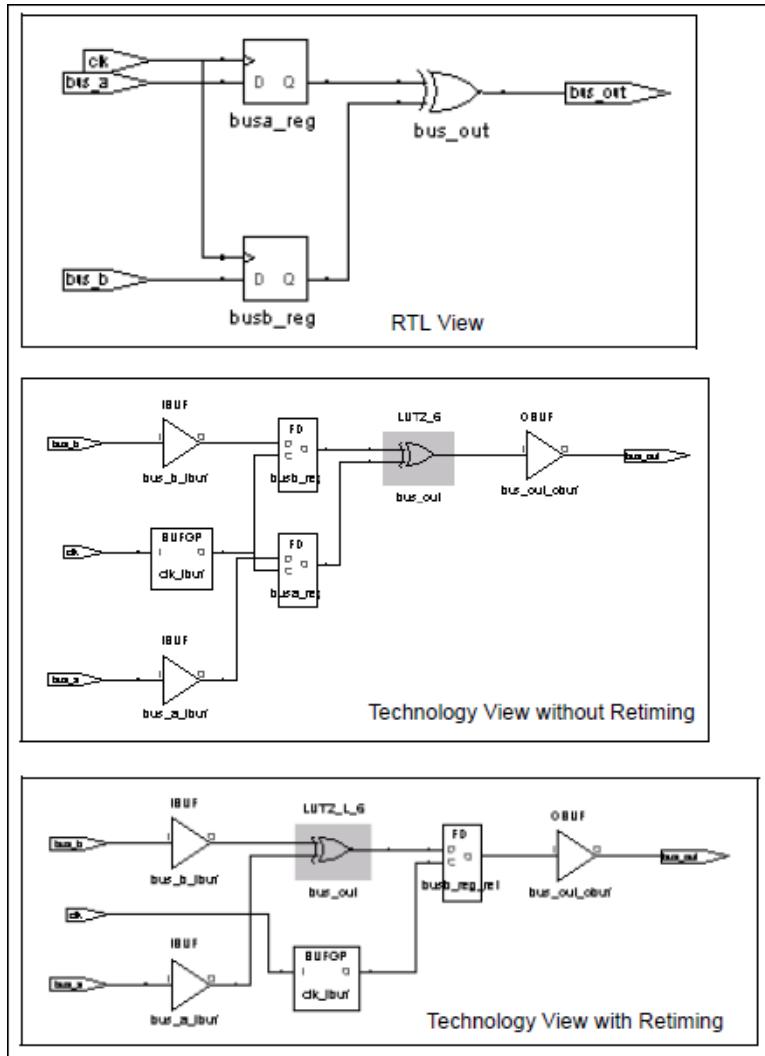
Retiming was on and timing exceptions (false paths) were applied.

Retiming moves registers to improve the worst-case slack in the design. During retiming, the synthesis tool may need to merge two or more registers into one. For example, when retiming registers across an AND gate with registered inputs, the registers may be merged at the output of the AND gate. However, if one register has a false path defined on it, the registers cannot be merged and hence retiming cannot move the registers from the input of the

AND gate to its output. This warning is only issued when retiming attempts to move some registers, but cannot do so because of a false path constraint applied to one of the registers.

In the design below, a false path is set on register `busa_reg`. The tool tries to retime both registers (`busa_reg` and `busb_reg`) to one register on the output of the xor (`bus_out`). However, because the false path constraint is applied to one register, the registers are not merged. The following schematics show the technology views with and without retiming.

## Schematics



The following test case causes this warning.

```
module test(bus_a, bus_b, clk, bus_out);
 input bus_a, bus_b;
 output bus_out;
 input clk;
 reg busa_reg, busb_reg;
 wire bus_out_wire;

 always @(posedge clk)
 begin
 busa_reg <= bus_a;
 busb_reg <= bus_b;
 end

 assign bus_out = busa_reg ^ busb_reg;
endmodule
```

## Constraint File Contents

```
create_clock -name {clk} {p:clk} -period 10
set_output_delay -clock {clk} 5.00 -ref {bus_out}
set_false_path -to {i:busa_reg}
```

Retiming is checked in the project window.

## Action

The warning is to let the user know why retiming could not occur. Possible solutions includes ignoring the warning, turning off retiming on the instance, or removing the false path constraint on the register.

# MF175

## **@N: Need <13> ESB/EABs for instance <mem[100:0]> but only <12> available.**

Indicates that the ESB/EAB resources available were insufficient to meet the requirements of the design. Both the number of ESB/EAB resources required and the number of ESB/EAB resources available are reported. For the test case example, the resources required for RAM mem are 13 ESBs; with only 12 ESBs available in the device, the mapper reports the above message.

## Project Options

```
#device options
set_option -technology APEX20KE
set_option -part EP20K30E
```

## Test Case

```
module test (wadd,din,wclk,rclk,wren,radd,dout);
 input [7:0]wadd;
 input [7:0] radd;
 input wclk,rclk,wren;
 input [100:0]din;
 output [100:0]dout;
 reg [100:0]mem[255:0];
 reg [7:0]reg_radd;

 always @ (posedge rclk)
 reg_radd <= radd;

 always@(posedge wclk)
 begin
 if(wren)
 mem[wadd] <= din;
 end

 assign dout= mem[reg_radd];
endmodule
```

## Action

Make sure that the device and part selected have the required number of EAB/ESB resources.

## MF176

### @N: Default generator successful

Indicates that the mapper generated the technology equivalent of an operator such as a multiplier, adder, or comparator (the corresponding operator symbol appears in the RTL view). The test case below displays the above message when run with the specific technologies.

```
module test (a, b, q, clk);
 input [3:0] a, b;
 input clk;
 output [7:0] q;
 reg [7:0] q;

 always @(posedge clk)
 begin
 q <= a * b;
 end
endmodule
```

#### Action

Informative message; no action required.

## MF180

### @N: Generating type <mult> multiplier

The mapper encountered a multiplier. This note is only reported for devices that do not include MULT blocks and indicates that multiplier described in the RTL is being implemented in logic.

In the test case below, inputs a and b are multiplied at every edge of clk. For the test case, global pipelining is enabled (Pipelining option checked on the Options panel).

```
module mult (a,b, q, clk);
 input [1:0] a,b;
 input clk;
 output [3:0] q;
 reg [3:0]q;

 always @(posedge clk)
 begin
 q = a*b;
 end
endmodule
```

## Action

Informative message; no action required.

# MF186

### **@W: Too many registers and EAB/ESBs required by RAMs for part; using ESB/EABs. See above**

The number of registers and EAB/ESBs required by the RAM implementation was greater than the number of registers available in the selected device (the actual number required was reported in the log (srr) file immediately preceding the warning). This warning also indicates that the mapper will use only EAB/ESBs and not use registers.

For the following test case, the SRR report indicates the following:

When implementing instance mem[100:0] the resources required are:  
EAB/ESBs = 13  
Registers 25856,  
but the resources available are:  
EAB/ESBs = 12  
Registers = 1200.

## Project Options

```
#device options
set_option -technology APEX20KE
set_option -part EP20K30E
```

## Test Case

```
module test (wadd,din,wclk,rclk,wren,radd,dout);
 input [7:0]wadd;
 input [7:0] radd;
 input wclk,rclk,wren;
 input [100:0]din;
 output [100:0]dout;
 reg [100:0]mem[255:0];
 reg [7:0]reg_radd;

 always @ (posedge rclk)
 reg_radd <= radd;

 always@(posedge wclk)
 begin
 if(wren)
 mem[wadd] <= din;
 end

 assign dout= mem[reg_radd];
endmodule
```

## Action

The actual resource utilization exceeds the device resources; you must either use a larger part or reduce the design's RAM requirements. For the above test case, changing the part from an EP20K30E to an EP20K60E provides the required resources.

## MF194

### @W: Ignoring invalid value <altshift\_tap> for syn\_srlstyle on instance <regBank[3:0]> in <view:work.test\_srl(verilog)>

The mapper detected an invalid value assigned to the `syn_srlstyle` attribute for the specified technology. The test case below is implemented using a technology that accepts values of `registers`, `select_srl`, and `noextractff_srl`. Because the `syn_srlstyle` value is `altshift_tap`, the mapper issues the above warning and uses the default style for the selected technology.

```
module test_srl(clk, enable, dataIn, result, addr);
 input clk, enable;
 input [3:0] dataIn;
 input [3:0] addr;
 output [3:0] result;
 reg [3:0] regBank[15:0] /* synthesis
 syn_srlstyle="altshift_tap" */;
 integer i;

 always @(posedge clk)
 begin
 if (enable == 1)
 begin
 for (i=15; i>0; i=i-1)
 begin
 regBank[i] <= regBank[i-1];
 end
 regBank[0] <= dataIn;
 end
 end
 assign result = regBank[addr];
endmodule
```

### Action

When specifying attribute values, make sure that the values are valid for the selected technology. In the corrected test case below, the value `noextractff_srl` is specified for `syn_srlstyle` attribute.

```

module test_srl(clk, enable, dataIn, result, addr);
 input clk, enable;
 input [3:0] dataIn;
 input [3:0] addr;
 output [3:0] result;
 reg [3:0] regBank[15:0]
 /* synthesis syn_srlstyle="select_srl" */;
 integer i;

 always @(posedge clk)
 begin
 if (enable == 1)
 begin
 for (i=15; i>0; i=i-1)
 begin
 regBank[i] <= regBank[i-1];
 end
 regBank[0] <= dataIn;
 end
 end
 assign result = regBank[addr];
endmodule

```

## MF203

### @N: Set autoconstraint\_io

When I/O ports are not explicitly mentioned, constraints are based on the clock period of the attached register. The use of this type of constraint is an implementation option that is enabled by checking Use clock period for unconstrained I/O on the Constraints tab of the implementation options or by adding `set_option -auto_constraint_io 1` to the project file. In the test case below, auto constrain I/O is enabled and, because no explicit I/O constraints are specified, the I/O constraints are automatically set.

```

module divider
 (input clk, input signed [7:0] a, b, output reg signed [7:0] y);

 always @ (posedge clk)
 y <= (a + b) ;

endmodule

```

## Action

Automatic constraints are not intended to replace regular timing constraints in the normal synthesis flow, but are intended to be used as a quick first pass to evaluate the kind of timing constraints you need to set in the design.

# MF236

### **@N: Generating a type <sdiv> divider**

A division operator (/) has been encountered. In the test case below, signed division is performed on the operands a and b which causes the Verilog compiler to issue the above note.

```
module divider
 (input clk, input signed [7:0] a, b, output reg signed [7:0] y);
 always @ (posedge clk)
 y <= (a / b) ;
endmodule
```

If the division is performed on signed data types, an sdiv divider type is generated; if the division is performed on unsigned data types, a div divider type is generated.

## Action

Informative message; no action required.

# MF237

### **@N: Generating a type <srem> remainder**

A division operator (%) has been encountered. In the test case below, signed division is computed on the operands a and b which causes the Verilog compiler to issue the above note.

```
module remainder
 (input clk, input signed [7:0] a, b, output reg signed [7:0] y);
 always @ (posedge clk)
 y <= (a % b) ;
endmodule
```

If the remainder is computed on signed data types, a `srem` remainder type is generated; if the remainder is computed on unsigned data types, a `rem` remainder type is generated.

## Action

Informative message; no action required.

# MF238

### **@N: Found <8>-bit incrementor, '<out\_1[7:0]>'**

The mapper has encountered an incrementor with an input size greater than or equal to four bits. In the test case below, running with specific technologies displays the above message (with parameter k being greater than or equal to 3).

```
module incrementor(in,clk,out);
parameter k =7;
input [k:0] in;
input clk;
output reg [k:0] out;

always@(posedge clk)
 out = in + 1;
endmodule
```

## Action

Informative message; no action required.

## MF239

### **@N: Found <8>-bit decrementor, '<out\_1[7:0]>'**

The mapper has encountered a decrementor with an input size greater than or equal to four bits. In the test case below, running with specific technologies displays the above message (with parameter k being greater than or equal to 3).

```
module decrementor(in,clk,out);
parameter k =7;
input [k:0] in;
input clk;
output reg [k:0] out;

always@(posedge clk)
 out = in - 1;
endmodule
```

#### Action

Informative message; no action required.

## MF244

### **@W: Glue logic created around RAM "<mem[3:0]>" to avoid read-write conflict.**

When you read and write to the same RAM address, the value of the output is indeterminate (an indeterminate output can create a simulation mismatch between RTL and post-synthesis simulation). To avoid the read-write address conflict and prevent the mismatch, the mapper creates bypass or “glue” logic around the RAM as indicated by the above warning.

In the test case below glue logic is created to resolve the read-write address conflict.

```
module DP_Block_Ram (WADDR, RADDR, data_in, CLK, WEA, data_out);
output [3:0] data_out;
input [3:0] WADDR,RADDR;
input [3:0] data_in;
input CLK, WEA;
reg [3:0] mem [15:0] /* synthesis syn_ramstyle="M4K" */;
reg [3:0] RADDR_reg;

always@(posedge CLK)
if(WEA)
mem[WADDR] = data_in;

always @(posedge CLK)
RADDR_reg <= RADDR;

assign data_out = mem[RADDR_reg];
endmodule
```

## Action

If you know that your design does not read and write to the same address simultaneously, use the `syn_ramstyle` attribute with a value of `no_rw_check` as shown in the test case below to eliminate the creation of bypass logic.

```
module DP_Block_Ram (WADDR, RADDR, data_in, CLK, WEA, data_out);
output [3:0] data_out;
input [3:0] WADDR,RADDR;
input [3:0] data_in;
input CLK, WEA;
reg [3:0] mem [15:0] /*synthesis syn_ramstyle="M4K,no_rw_check"*/ ;
reg [3:0] RADDR_reg;

always@(posedge CLK)
if(WEA)
mem[WADDR] = data_in;

always @(posedge CLK)
RADDR_reg <= RADDR;

assign data_out = mem[RADDR_reg];
endmodule
```

# MF245

## @W: To avoid glue logic, use syn\_ramstyle = "no\_rw\_check" attribute.

The above warning occurs in conjunction with [MF244](#) and indicates that the creation of bypass or “glue” logic can be avoided by using the `syn_ramstyle` attribute with a value of `no_rw_check`. In the test case below, glue logic is created around the RAM to avoid simultaneously reading and writing the same RAM location.

```
module DP_Block_Ram (WADDR, RADDR, data_in, CLK, WEA, data_out);
output [3:0] data_out;
input [3:0] WADDR,RADDR;
input [3:0] data_in;
input CLK, WEA;
reg [3:0] mem [15:0] /*synthesis syn_ramstyle="M4K"*/;
reg [3:0] RADDR_reg;

always@(posedge CLK)
if(WEA)
 mem[WADDR] = data_in;

always @ (posedge CLK)
 RADDR_reg <= RADDR;

assign data_out = mem[RADDR_reg];
endmodule
```

## Action

If you know that your design does not read and write to the same address simultaneously, use the `syn_ramstyle` attribute with a value of `no_rw_check` as shown in the test case below to eliminate the creation of bypass logic.

```
module DP_Block_Ram (WADDR, RADDR, data_in, CLK, WEA, data_out);
output [3:0] data_out;
input [3:0] WADDR,RADDR;
input [3:0] data_in;
input CLK, WEA;
reg [3:0] mem [15:0] /*synthesis syn_ramstyle="M4K,no_rw_check"*/;
reg [3:0] RADDR_reg;
```

```
always@(posedge CLK)
 if(WEA)
 mem[WADDR] = data_in;

 always @(posedge CLK)
 RADDR_reg <= RADDR;

 assign data_out = mem[RADDR_reg];

endmodule
```

## MF249

### **@N: Running in 32-bit mode.**

Indicates that synthesis was performed in 32-bit mode, even when the host supported 64-bit operation.

#### Action

If you want to run in 64-bit mode (on a 64-bit machine) and you receive the above message, make sure that the Enable 64-bit mapping box is checked on the Project Properties dialog box.

To display the Project Properties dialog box, right click on the project name in the Project view and select Project Options.

## MF252

### **@W: Running in 32-bit mode. 64-bit mode was requested but is unavailable.**

Indicates that synthesis was performed in 32-bit mode when 64-bit mode was requested and that the host does not support 64-bit operation.

### Action

To eliminate the warning, make sure that the Enable 64-bit mapping box is not checked on the Project Properties dialog box.

To display the Project Properties dialog box, right click on the project name in the Project view and select Project Options.

## MF257

### **@N: Gated clock conversion enabled**

The Fix gated clocks option was enabled on the Device tab of the Options for implementation dialog box (a value of 1, 2, or 3 can be specified) and that a gated-clock report will be included in the log (srr) file according to the value specified. The equivalent command in the project file is

```
set_option -fixgatedclocks 1 (or 2 or 3)
```

### Action

Informative message; no action required. Specifying a value of 0 disables gated-clock conversion.

## MF258

### **@N: Gated clock conversion disabled**

The Fix gated clocks option was disabled on the Device tab of the Options for implementation dialog box (a value of 0 is specified). The equivalent command in the project file is

```
set_option -fixgatedclocks 0
```

## Action

Informative message; no action required. Specifying a value of 1, 2, or 3 enables gated-clock conversion.

# MF273

### **@N: Providing a constraint file for compile point <sub\_4s> can improve results.**

There was no associated constraint file for a type soft or hard compile point (soft and hard compile points are not supported and, if present, result in a warning). For the test case below, submodule sub\_4s is defined as a compile point in the top-level constraint file but, because there is no constraint file entry for the compile point, the mapper displays the above message.

```
##Top-level Constraints##
define_compile_point -comment {Automatically Inserted}
{v:sub_4s} -type {soft}
```

### Test Case

```
module top (q,a,b,c,d);
parameter top_param = 4;
output [top_param:0] q;
input [top_param:0] a,b,c,d;
sub u2 (q,a,b,c,d);
defparam u2.param=4;
endmodule

module sub (q,a,b,c,d);
parameter param = 4;
output [param:0] q;
input [param:0] a,b,c,d;
assign q = a & b & c & d;
endmodule
```

## Action

When working with soft or hard compile points, specify a compile constraint file for the compile point as shown in the second entry below.

```
##Top-level Constraints##
define_compile_point -comment {Automatically Inserted}
 {v:sub_4s} -type {soft} -cpfile {sub_4s_cp.sdc}

##sub_4s_cp.sdc##
define_current_design {sub_4s}
```

# MF283

## @N: Configuring synthesis for optimum stability of results

Indicates that a syn\_stable\_mode attribute with a value of 1 was encountered in the constraint file (minimize changes to output netlist following an incremental design change).

### Constraint File Contents

```

Attributes
#
define_global_attribute {syn_stable_mode} {1}
```

### Test Case

```
module top (q,a,b,c,d);
parameter top_param = 4;
output [top_param:0] q;
input [top_param:0] a,b,c,d;
sub u2 (q,a,b,c,d);
defparam u2.param = 4;
endmodule
```

```
module sub (q,a,b,c,d);
parameter param = 4;
output [param:0] q;
input [param:0] a,b,c,d;
assign q= a & b & c & d;
endmodule
```

### Action

Informative message; no user action required.

## MF320

### **@E: Verilog generator failed. See log file <*logFilename*>**

The above error occurs when there is a failure in the call to the Verilog compiler from the mapper.

### Action

Check the log file for additional information. The call from the mapper can fail for any of the reasons listed below:

- No write permission for the implementation directory. Make sure you have write access to this directory.
- Insufficient disk space available. Make sure there is adequate disk space for the implementation directory (large databases can require in excess of 10 GBytes); remove unnecessary content to free up additional memory.
- Failure to checkout a license as a result of network delays or problems external to the tool. Repeating the operation can eliminate this error.

# MF529

## @E: Tristates on constant net VCC/GND triggering multiple-drivers failure

The inout port is assigned multiple times and one assignment is a constant 0 or 1. The value of the inout port register *temp* is a constant with a signal assignment.

### Action

- Avoid multiple assignments for the inout port.
- Do not assign constants for the inout register *temp*.

### Example

Initial configuration that displays the error message:

```
module bidirec (oe, clk, inp, outp, bidir,temp);
 // Port Declaration

 input oe;
 input clk;
 input [7:0] inp;
 output [7:0] outp;
 inout [7:0] bidir;
 inout temp;

 reg [7:0] a;
 reg [7:0] b;

 assign bidir = oe ? a : 8'bZ ;
 assign temp = oe ? b : 8'bz;
 assign temp = 1;
 assign outp = b;

 // Always Construct

 always @ (posedge clk)
 begin
 b <= bidir;
 a <= inp;
```

```
end
endmodule
```

The code modified to correct the errors:

```
module bidirec (oe, clk, inp, outp, bidir,temp);

// Port Declaration

input oe;
input clk;
input [7:0] inp;
output [7:0] outp;
inout [7:0] bidir;
inout temp;

reg [7:0] a;
reg [7:0] b;

assign bidir = oe ? a : 8'bZ ;
assign temp = oe ? b : 8'bz;
assign outp = b;

// Always Construct

always @ (posedge clk)
begin
 b <= bidir;
 a <= inp;

end
endmodule
```

# MF531

## **@E: Conflicting driver <signal\_name>**

The inout signal is being driven by multiple drivers. This error is connected to the error MF529.

### Action

Open the RTL source file and modify the content as explained in MF529.

## MF559

**@W: Active phase of latch <latchName> prevents its use as a stability latch for clock gate <clockGateName>.**

The above warning occurs in an IGC-type circuit when an incorrect phase of the clock is connected to the latch which prevents the latch from being converted and used as a stability latch. The warning indicates that the loads on the specified latch either cannot be determined or that the loads change on both a rising and falling edge.

### Action

Modify the circuit as follows:

- If an active-high latch is used with an AND gate in a circuit targeted for a rising edge flip-flop, make sure that the latch is clocked by the inverted version of the clock used for gating.
- If an active-high latch is used with an OR gate in a system targeted for a falling edge flip-flop, make sure that the latch is clocked by the non-inverted version of the clock used for gating.

For more information, see *Integrated Clock Gating Cells* in the *User Guide*.

## MF561

**@W: Active phase of latch <latchName> prevents its use as a stability latch for clock gate <clockGateName>; bypassing latch**

The above warning occurs in an IGC-type circuit when an incorrect phase of the clock is connected to the latch which prevents the latch from being considered as a stability latch. The named latch is bypassed (removed from consideration as a stability latch). This warning usually indicates a clock-gating problem that can occur when the output of the latch is ANDed with its initial clock signal. This type of gating generates a late edge or a pulse when the clock goes high which allows the output of the AND gate to change; when the latch goes transparent, the output can change again. This type of gating usually indicates a design error.

## Action

Modify the circuit to eliminate the clock-gating problem. For more information, see *Integrated Clock Gating Cells* in the *User Guide*.

# MF562

### **@W: Active phase of latch <latchName> prevents its use as a stability latch for clock gate <clockGateName>; replacing latch with DFF**

The named latch is not suitable as a stability latch and is being replaced with a D-flip-flop.

## Action

Either correct the conditions preventing the latch from being used as a stability latch or allow the latch replacement with the D-flip-flop. For more information, see *Integrated Clock Gating Cells* in the *User Guide*.

# MF576

### **@W: Transparent phase of latch has been extended by clock gate. Performing clock optimization on instance <instanceName>.**

The above warning occurs in an IGC-type circuit during an OR-gated latch conversion in which the transparent phase of the latch is extended by the clock gate. The target instance of the clock optimization is reported in the message.

## Action

Either correct the conditions that extend the transparent phase of the latch to eliminate the warning or allow the transparent phase to become extended. For more information, see *Integrated Clock Gating Cells* in the *User Guide*.

## MF579

### **@W: Active phase of latch <latchName> prevents its use as a stability latch.**

The above warning occurs in an IGC-type circuit when an incorrect phase of the clock is connected to the latch which prevents the latch from being used as a stability latch. The warning indicates that the loads on the specified latch either cannot be determined or that the loads change on both a rising and falling edge.

#### Action

Modify the circuit as follows:

- If an active-high latch is used with an AND gate in a circuit targeted for a rising edge flip-flop, make sure that the latch is clocked by the inverted version of the clock used for gating.
- If an active-high latch is used with an OR gate in a system targeted for a falling edge flip-flop, make sure that the latch is clocked by the non-inverted version of the clock used for gating.

For more information, see *Integrated Clock Gating Cells* in the *User Guide*.

## MF580

### **@W: Active phase of latch <latchName> prevents its use as a stability latch; bypassing latch**

The above warning occurs in an IGC-type circuit when an incorrect phase of the clock is connected to the latch which prevents the latch from being considered as a stability latch. This warning usually indicates a clock-gating problem that can occur when the output of the latch is ANDed with its initial clock signal. This type of gating generates a late edge or a pulse when the clock goes high which allows the output of the AND gate to change; when the latch goes transparent, the output can change again. This type of gating usually indicates a design error.

## Action

Modify the circuit to eliminate the clock-gating problem. For more information, see *Integrated Clock Gating Cells* in the *User Guide*.

# MF581

### **@W: Active phase of latch <latchName> prevents its use as a stability latch for clock gate <clockGateName>; replacing latch with DFF**

The named latch is not suitable as a stability latch and is being replaced with a D-flip-flop.

## Action

Either correct the conditions preventing the latch from being used as a stability latch or allow the latch replacement with the D-flip-flop. For more information, see *Integrated Clock Gating Cells* in the *User Guide*.

# MF585

### **@W: High reliability inference not supported for RAM <moduleName> with mapping style <type> – ignoring syn\_ramstyle attribute**

The high-reliability feature (for example, syn\_radhardlevel set to distributed\_tmr or duplicate\_with\_compare) does *not* support specifying MLAB or select RAM with the syn\_ramstyle attribute. When the synthesis software encounters these RAM, they are mapped to either registers or block RAM.

## Action

The message informs you how the RAM will be implemented when it is different from what was specified for the syn\_ramstyle attribute. No action is required.

You have specified different technologies on a multi-FPGA system. System-level timing with back-annotation is not supported for mixed technologies. If you also receive the related MF805 error message, check that message for recommendations.

## MF627

### **@E: PortWidth property for instance <instanceName> not found or invalid**

The PortWidth property is a required parameter. The above error occurs if the specified PortWidth parameter either is not set or has an invalid value. In the following excerpt, the parameter is not set when the module is instantiated which results in the error.

```
SceMiMessageInPort i_DataInPort
 .ReceiveReady (iPort_re_ready),
 .Message (iPort_message),
 .TransmitReady (iPort_transmitReady) ;
```

#### Action

Make sure that a valid PortWidth parameter is specified in the module as shown in the following example:

```
SceMiMessageInPort #(.PortWidth (MessageBitWidth)) i_DataInPort
 .ReceiveReady (iPort_re_ready),
 .Message (iPort_message),
 .TransmitReady (iPort_transmitReady) ;
```

## MF663

### **@E: Synthesis netlist database <name> not found, or not complete**

The tool was unable to find the required multi-file (.srs or .srd) database netlist files in the implementation directory. The process cannot proceed without these multi-file database files.

## Action

Run synthesis to generate the multi-file database files required.

# MF680

**@E: RAM instance <instanceName> exceeds the register threshold and cannot be mapped to RAM in the specified technology.  
Please use "syn\_max\_memsize\_reg" attribute to change the threshold limit of the registers used to map a RAM instance.**

The register threshold for a RAM in the selected technology has been exceeded. The number of registers that can be mapped to a RAM is controlled by the syn\_max\_memsize\_reg attribute. The default value for this attribute is 128.

For the following test case, the above message occurs because the number of registers to be mapped to the RAM is greater than the threshold limit.

```
module test (data0,data1,waddr0,waddr1,we0,we1,clk,q0, q1);
parameter d_width = 2;
parameter addr_width = 7;
parameter mem_depth = 128;
input [d_width-1:0] data0, data1;
input [addr_width-1:0] waddr0, waddr1;
input we0, we1, clk;

reg [d_width-1:0] mem [mem_depth-1:0];
reg [addr_width-1:0] reg_waddr1 = 7'h7B;
reg [addr_width-1:0] reg_waddr0 = 7'h5C;
output reg [d_width-1:0] q0;
output reg [d_width-1:0] q1;

always @(posedge clk) begin
 q0 <= mem[reg_waddr0];
 q1 <= mem[reg_waddr1];
end

always @(posedge clk)
begin
 if (we0)
 mem[waddr0] <= data0;
```

```
 reg_waddr0 <= waddr0;
end

always @(posedge clk)
begin
 if (wel)
 mem[waddr1] <= data1;
 reg_waddr1 <= waddr1;
end
endmodule
```

## Action

You can either:

- Raise the register limit above the current value using the `syn_max_mem-size_reg` attribute to allow the registers to be mapped to RAM.
- Specify the `syn_ramstyle=registers` attribute to force the corresponding RAM to be mapped to discrete registers.

## MF682

**@E: Initial values are not supported for RAM <instanceName> which includes a global reset.**

See description for error message [MF683](#).

## MF683

**@E: Initial values found on one or more RAMs with global reset.  
Please check above errors.**

The above error and error [MF682](#) occur when an attempt is made to initialize a RAM that includes a global reset (RAMs with global resets cannot be initialized). The following is a test case that reproduces these errors.

```
module RAM (clk, we, rst, din, addr, dout);
parameter data_width = 4 ;
parameter depth = 16 ;
parameter add_width = 4;

input clk;
input rst;
input we;
input [data_width-1:0] din;
input [add_width-1:0] addr;
output [data_width-1:0] dout;
reg [data_width-1:0] dout;
integer i;
reg [data_width-1:0] mem [depth-1:0]
/* synthesis syn_ramstyle="block_ram" */;
initial
begin
 $readmemb("data.dat",mem);
end

always @(posedge clk or posedge rst)
begin
 if (rst)
 begin
 for (i = 0; i < depth; i = i+1)
 begin
 mem[i] <= 0;
 end
 dout <= 0;
 end
 else
 begin
 if (we == 1'b1)
 begin
 mem[addr] <= din;
 dout <= din;
 end
 else
 dout <= mem[addr];
 end
end
endmodule
```

## Initialization File

The following binary initialization file (data.dat) is used with test case.

```
0000
0001
0010
0011
0100
0101
0110
0111
```

## Action

Remove the initialization sequence or use a RAM without a global reset.

# MF689

## **@E: Multiple error monitoring values specified for instance *<instanceName>***

You specified error monitoring for an instance more than once using the `syn_create_err_net` or `syn_connect` Tcl command of the high-reliability feature.

## Action

You must remove the redundant error monitoring `syn_create_err_net` or `syn_connect` Tcl command for the specified instance from the constraint file (PDC).

## MF690

### **@E: Encountered hard hierarchy while threading net from <netName> to <netName>**

When the error monitoring instance you specified for the high-reliability feature encounters hard hierarchy, this message occurs while trying to thread the nets for the following:

- The control circuitry (for example, using the -err\_clk or -err\_set options of the syn\_create\_err\_net Tcl command)
- The target error net (for example, using the -to option of the syn\_connect Tcl command)

#### Action

To eliminate this error:

- Move the error monitoring signals to the same hierarchy as the error monitoring instance you specified or make sure that the error monitoring signals do not cross hard hierarchy.
- Remove the syn\_hier attribute from the hierarchy to allow the nets to be threaded.

## MF700

### **@E: Only one of the following RAMs <moduleName>, <moduleName> require error-monitoring Tcl commands.**

Two multi-port RAMs (NRAM) have been specified for error monitoring with the syn\_create\_err\_net and syn\_connect Tcl commands. The synthesis software generates this error because it cannot handle this condition properly.

#### Action

You must remove one of the RAM error monitoring Tcl commands from the constraint file (FDC).

## MF706

### **@E: Filename length for compile point <CPfileName> exceeds allowable limit.**

When you run a design using the compile point flow, you might encounter this error if the complete compile point file path name exceeds the threshold limit of 239 characters on a Windows platform. The specific compile point file that the synthesis software generates is specified as:

*implementationPath/CPname/report/CPname\_timing\_report.xml*

#### Action

The compiler unifies the compile points and can pass along long file names to the mapper, where the file length limit may be exceeded on Windows. To avoid this error, run the design on Linux instead.

## MF707

### **@N: Insert external logic with either syn\_ramstyle=rw\_check attribute or enable 'Read Write Check on RAM' option to resolve read/write conflict for <ramDevice>.**

The above note appears when the RTL code written for RAM inferencing attempts to both read from and write to the same address location at the same time (read/write address collision check). If the indicated block RAM does not support this behavior, the read data output would be unknown which could lead to a potential post-synthesis simulation mismatch.

To avoid the mismatch, the Synplify tools insert glue logic to resolve any possible conflict provided that either:

- the define `syn_ramstyle` attribute is set on the RAM with a value of `rw_check` (either in the RTL or defined in an FDC file)
- the enable 'Read Write Check on RAM' option on the Device tab of the Implementation Options dialog box is set.

By default, the tool does not check for read/write address collisions and therefore does not automatically insert any glue logic.

## MF720

### **@E: Found constraint file with both define\_clock and create\_clock commands**

You cannot specify both a `create_clock` and a legacy `define_clock` command in the same constraint file.

#### Action

Use only the `create_clock` command in an FDC constraints file.

## MF744

### **@E: Could not find instance '<instanceName>' for instrumentation of signal '<signalName>'. Verify SRS instrumented signals in IDC file**

Information written in the IDC constraint file does not match information in the current design. This occurs when the design changes or when a compiler-generated signal name is changed.

#### Action

Update the IDC constraint file to match the design.

## MF745

**@E: Could not find instrumented net '<netName>' within instance '<instanceName>'. Verify SRS instrumented signals in IDC file**

Information written in the IDC constraint file does not match information in the current design. This occurs when the design changes or when a compiler-generated signal name is changed.

Action

Update the IDC constraint file to match the design.

## MF752

**@E: edf2srs failed. See log file <logFileName>**

The edf2srs utility was unable to create an srs file from the input edif file.

Action

Contact Synopsys support by logging in to the Synopsys website.

## MF753

**@E: edf2srs failed to create an output file. See log file <logFileName>**

The edf2srs utility was unable to write the output srs file.

Action

Contact Synopsys support by logging in to the Synopsys website.

## MF754

### **@E: Reading of <outputFilename> file failed**

The edf2srs utility was unable to read the generated srs output file.

#### Action

Contact Synopsys support by logging in to the Synopsys website.

## MF756

### **@E: Could not map asymmetric RAM <name>. Use prescribed template only.**

The instantiation of the named syn\_asym\_tdp\_ram macro does not match the prescribed template. The asymmetric RAM could not be inferred.

#### Action

To infer true dual-port asymmetric RAM, use the syn\_asym\_tdp\_ram\_model template to correctly define the port connections and instantiate the RAM macro.

For details and examples of the Verilog/VHDL templates used to create an instantiation model for the asymmetric RAM, see *Inferring Asymmetric RAM as a Black Box Model* in the *FPGA Synthesis User Guide*.

## MF766

### **@E: Could not open file <filename> for reading**

A required file is in use by another program.

### Action

Unlock the file or remove it from the project, then rerun the project.

## MF775

### **@E: Targeting module <moduleName> to an FPGA inside an FPGA is not allowed.**

This error occurs when a submodule within a module targeting an FPGA also targets an FPGA.

### Action

For modules targeting an FPGA, remove any attributes which target an FPGA from the submodules.

## MF778

### **@E: Multiple objects <objects> specified for -to option of syn\_connect command.**

This error occurs when the -to argument of the syn\_connect command has more than one object specified. Example:

```
syn_connect { -from {n:netName} -to {value1 value2}};
```

In this example, the software attempts to connect *netName* to both *value1* and *value2*.

### Action

Make sure that the -to argument of syn\_connect has only one specified object.

## MF779

**@E: Bus net <net> not supported for error monitoring. Please see the user manual for correct usage.**

The value of the -to argument of syn\_connect has a width greater than 1, which leads to an ambiguous situation where more than one net is specified for error monitoring.

### Action

See *Command Reference Manual* for correct command usage.

## MF780

**@E: syn\_connect: Only -n (net), -p (port), or -t (pin) supported for -to argument; object specified is <type>. Please see the user manual for correct usage.**

An unsupported object type is specified for -to. Only a net, port, or pin may be specified as the value of the -to argument of syn\_connect.

### Action

See *Command Reference Manual* for correct command usage.

## MF781

**@E: syn\_connect: An existing bit port, port, or net is not specified for -to argument; object specified is <object>. Please See the user manual for correct usage.**

Object specified for -to argument does not exist. Value of argument must be an existing port, bit port, or net.

### Action

See *Command Reference Manual* for correct command usage.

## MF794

### **@N: RAM <name> required <integer> registers during mapping.**

Reports the number of registers required to map the indicated RAM when it cannot be mapped to block RAM. This type of information is used to monitor RAM-related resource usage within a design.

### Action

Informative message; no action normally required. In the presence of related error or warning messages, can indicate insufficient resources.

## MF804

### **@E: Cannot map asymmetric RAM instance <instanceName> into appropriate logic.**

The asymmetric RAM cannot be mapped to appropriate logic when the read and write address widths for the instance are not the same, as shown in the code example below:

```
input [3:0]raddr;
input [3:0] data;
input [3:0]waddr;
input we;
input clk;
output [4:0] dout;
reg [4:0] dout;
reg [4:0] mem [31:0];
reg [4:0] raddr_reg;
```

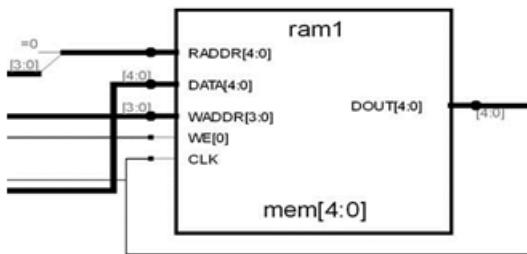
```

// write logic
always @(posedge clk)
begin
 if (we)
 mem[waddr] <= data;
end

//read logic
always @(posedge clk)
begin
 raddr_reg <= raddr;
 dout <= mem[raddr_reg];
end
endmodule

```

Here is the schematic for the asymmetric RAM instance above.



## Action

Make sure the read address width is the same as the write address width for the asymmetric RAM instance, so that mapping to logic can be implemented properly.

# MF818

**@W: No paths exist from user-instantiated pad instance <instanceName> to any top-level port.**

The message above occurs when the specified pad instance is not directly connected to a top-level port in the netlist.

### Action

You can remove the pad instance. If no action is taken, the place-and-route tool will resolve this warning; you have no control on its outcome.

## MF819

### **@W: User instantiated pad instance <*instanceName*> is connected to net <*netName*> with multiple fanins or fanouts**

The message above occurs when the net connecting the pad instance and a top-level port has additional inputs and/or outputs.

### Action

Make sure to place pad instances in locations where the top-level ports do not contain nets that are connected to multiple inputs or outputs. If no action is taken, the place-and-route tool will resolve this warning; you have no control on its outcome.

## MF820

### **@W: User instantiated pad instance <*instanceName*> is on the same path as user-instantiated pad instance <*instanceName*>.**

The message above occurs when you specify a pad instance on a path to a top-level port that already has a pad instance specified.

### Action

Remove one of the pad instances specified to the top-level port. If no action is taken, the place-and-route tool will resolve this warning; you have no control on its outcome.

## MF821

### **@E: Cannot find signal <*signalName*> for hyper connect**

The software tried to probe a signal across a hierarchy that does not exist.

#### Action

If you use cross-module referencing in a design, make sure to check the netlist and verify that the signal being instrumented is valid.

## MF822

### **@E: Un-instrumentation is not supported in distributed synthesis. Please remove instrumentation for signal <*signalName*> from the idc file.**

You tried to instrument a signal that does not exist or cannot be found in a design. This error can occur during distributed synthesis if the instrumented net does not exist or was specified incorrectly. For example, the net can have a typographical error.

#### Action

Specify the net or path name correctly. Otherwise, remove the net defined in the IDC file from instrumentation.

## MF824

### **@E: Failed to open internal log file for writing. Check disk space.**

The above error can occur when there is a failure in the VHDL compiler to start a sub-process for mapping and then could not successfully write any output to a log file.

### Action

The sub-process can fail to start mapping for the following reasons:

- Insufficient disk space available. Make sure you have adequate disk space for the sub-process to map and write to the log file.
- Possibly, a limit has been set for the number of processes. Make sure the number of processes to be run does not exceed this limit.

## MF829

### **@E: Debug SRS instrumentation failed. See 'instrumentation.log' for more information.**

The Identify instrumentor job failed to complete successfully during synthesis.

### Action

Check the instrumentor log file displayed for the obfuscated database to determine the cause of the error, which provides details that are usually self-explanatory.

## MF830

### **@E: Identify core generator failed. See '<logFileName>' for more information.**

The Identify instrumentor job failed to complete successfully during synthesis. This can commonly occur when there is a problem generating IP.

### Action

Check the named log file to determine the cause of the error, which provides details that are usually self-explanatory.

## MF831

**@E: Distributed debug instrumentation failed. See 'instrumentation.log' for more information.**

The Identify instrumentor job failed to complete successfully during distributed synthesis.

### Action

Check the instrumentor log file displayed for the obfuscated database to determine the cause of the error, which provides details that are usually self-explanatory.

## MF834

**@E: The number of registers used to synthesize RAM <instanceName> <technologyPart> is larger than the total number of registers available on the chip <numberOfRegisters> <technologyPart>.**

The software tried to map the RAM to registers, but the number of registers required to map this RAM consumed more resources than are available on the chip.

### Action

You can either select a larger part for the technology or use the syn\_ramstyle attribute to implement the RAM as block RAM.

## MF842

**@E: Legacy-style timing constraints are not supported in the selected device technology. Use FDC style constraints.**

Legacy-style FPGA timing constraints have been encountered while reading in a constraint file for an existing design.

Action

Check the corresponding FDC file and replace any define\_\* timing constraints including define\_clock, define\_input\_delay, define\_multicycle\_path, define\_false\_path, define\_max\_delay, and define\_output delay with equivalent create\_clock or set\_\* constraints (for example, set\_input\_delay or set\_multicycle\_path).

## MF845

**@E: Asymmetric RAM instance cannot be decomposed due to invalid configuration.**

The following message is technology specific and results when an attempt is made to decompose an asymmetric RAM instance and the RAM cannot be mapped using any valid read width configuration for the RAM.

Action

To avoid the error, enable read-write check on the RAM to prevent the RAM from being mapped as an asymmetric RAM and to allow the RAM to be mapped as a synchronous RAM.

## MF846

### **@E: Cannot decompose Asymmetric RAM instance due to invalid configuration.**

The following message is technology specific and results when an attempt is made to decompose an asymmetric RAM instance and the RAM cannot be mapped using any valid write width configuration for the RAM.

#### Action

To avoid the error, enable read-write check on the RAM to prevent the RAM from being mapped as an asymmetric RAM and to allow the RAM to be mapped as a synchronous RAM.

## MF985

### **@E: Multiple instantiation of trigger instance %I for IICE '%s' found in the design. Only one instance is allowed.**

This message is specific to multi-FPGA debug flow, and occurs during the pre-partition stage in ProtoCompiler.

In debug flow, only one trigger instance per IICE is allowed in the design. The flow cannot continue if you instantiate multiple instances directly or indirectly through instantiation of other views in the design.

#### Action

Configure only one trigger instance for each IICE in the design.

## MF986

**@E: Multiple trigger instances %I and %I for IICE '%s' found in the design. Only one instance is allowed.**

This message is specific to multi-FPGA debug flow. This error occurs if there are multiple trigger instances for an IICE. The tool allows only one trigger instance for each IICE.

### Action

Configure only one trigger instance for each IICE in the design.



## CHAPTER 28

# MO Messages 106 – 224

---

## MO106

### **@N: Found ROM <z\_1[3:0]> with <19> words by <4> bits.**

The mapper has encountered a ROM structure. The note includes the size of the ROM in terms of words and bits. The test case below infers a ROM size of 19 words by 4 bits.

```
module rom(z,a);
 output [3:0] z;
 input [4:0] a;
 reg [3:0] z;

 always @(a) begin
 case (a)
 5'b00000 : z = 4'b0001;
 5'b00001 : z = 4'b0010;
 5'b00010 : z = 4'b0110;
 5'b00011 : z = 4'b1010;
 5'b00100 : z = 4'b1000;
 5'b00101 : z = 4'b1001;
 5'b00110 : z = 4'b0000;
 5'b00111 : z = 4'b1110;
 5'b01000 : z = 4'b1111;
 5'b01001 : z = 4'b1110;
 5'b01010 : z = 4'b0001;
```

```

5'b01011 : z = 4'b1000;
5'b01100 : z = 4'b1110;
5'b01101 : z = 4'b0011;
5'b01110 : z = 4'b1111;
5'b01111 : z = 4'b1100;
5'b10000 : z = 4'b1000;
5'b10001 : z = 4'b0000;
5'b10010 : z = 4'b0011;
default : z = 4'b0111;
endcase
end
endmodule

```

## Action

Informative message; no action required.

# MO108

### **@W: Unknown SM encoding option <gray1>**

The mapper encountered an incorrect State Machine Encoding option specified by the `syn_encoding` attribute. In the test case below, an invalid attribute value (`gray1`) is specified which causes the mapper to issue the above warning.

```

module FSM1 (clk, rst, in1, out1);
 input clk, rst, in1;
 output [2:0] out1;
 `define s0 3'b000
 `define s1 3'b001
 `define s2 3'b010
 `define s3 3'bxxx
 reg [2:0] out1;
 reg [2:0] state /* synthesis syn_encoding="gray1" */;
 reg [2:0] next_state;

 always @(posedge clk or posedge rst)
 if (rst) state <= `s0;
 else state <= next_state;

```

```
// Combined Next State and Output Logic
always @(state or in1)
 case (state)
 `s0 : begin
 out1 <= 3'b000;
 if (in1) next_state <= `s1;
 else next_state <= `s0;
 end
 `s1 : begin
 out1 <= 3'b001;
 if (in1) next_state <= `s2;
 else next_state <= `s1;
 end
 `s2 : begin
 out1 <= 3'b010;
 if (in1) next_state <= `s3;
 else next_state <= `s2;
 end
 default : begin
 out1 <= 3'bxxx;
 next_state <= `s0;
 end
 endcase
endmodule
```

## Action

Make sure that a legal value of onehot, gray, sequential, safe (in conjunction with any of the previous three), or original is entered as the attribute value. In the corrected test case below, a value of gray is given to the `syn_encoding` attribute which causes the mapper to use gray style encoding for the state machine extracted by the FSM compiler.

```
module FSM1 (clk, in1, rst, out1);
 input clk, rst, in1;
 output [2:0] out1;
 `define s0 3'b000
 `define s1 3'b001
 `define s2 3'b010
 `define s3 3'bxxx
 reg [2:0] out1;
 reg [2:0] state /* synthesis syn_encoding="gray" */;
 reg [2:0] next_state;
```

```
always @(posedge clk or posedge rst)
 if (rst) state <= `s0;
 else state <= next_state;

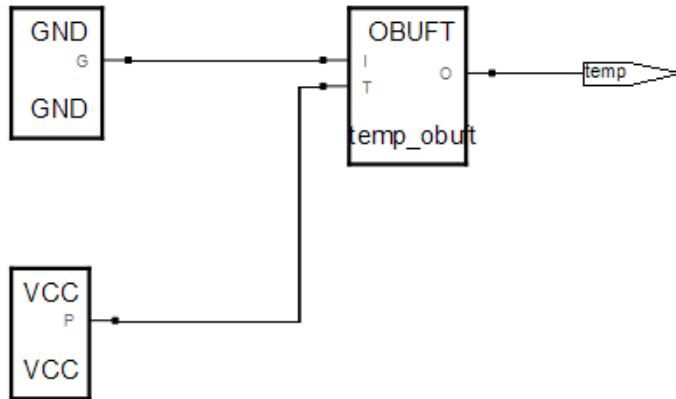
// Combined Next State and Output Logic
always @(state or in1)
 case (state)
 `s0 : begin
 out1 <= 3'b000;
 if (in1) next_state <= `s1;
 else next_state <= `s0;
 end
 `s1 : begin
 out1 <= 3'b001;
 if (in1) next_state <= `s2;
 else next_state <= `s1;
 end
 `s2 : begin
 out1 <= 3'b010;
 if (in1) next_state <= `s3;
 else next_state <= `s2;
 end
 default : begin
 out1 <= 3'bxxxx;
 next_state <= `s0;
 end
 endcase
endmodule
```

## MO111

### **@W: Tristate driver <temp> on net <temp> has its enable tied to GND (module <test>)**

An undriven primary output has been tristated by inferring an output buffer with a permanently inactive enable. In the following test case, output port temp is unassigned in the module which results in the insertion of an output buffer on the net as shown in the Technology view and indicated in the message text.

```
module test (tms,w_tms,swdoen,swdo,combo1,t1,t2,temp,tempin);
output temp;
input tempin;
inout tms;
output w_tms;
input swdoen, swdo;
output combo1;
input t1,t2;
assign tms = (swdoen) ? swdo : 1'bz;
assign w_tms = tms;
assign combo1 = t1 | t2 | tms;
endmodule
```



## Action

If the omitted signal assignment is an oversight, complete the assignment.

## MO112

### **@W: Deleting tristate instance <instanceName> on net <netName> because its enable is connected to 0.**

The enable of a tristate driver is always false which prevented it from driving its associated output. In the test case below, the enable input (en) is assigned a value of 0 which permanently disables the driver so that input in1 is never driven. The tristate driver appears in the RTL view, but is removed from the Technology view with the above message.

```
library ieee;
use ieee.std_logic_1164.all;

entity tristate is
 port (in1,in2,in3,clk: in std_logic;
 regout:out std_logic);
end tristate;

architecture behave of tristate is
signal en,tmp1,regin: std_logic;
begin
 en<='0';
 process (en, in1)
 begin
 if (en = '0') then
 tmp1 <= 'Z';
 else
 tmp1 <= in1;
 end if;
 end process;

 tmp1<=in3;
 regin <= tmp1 and in2;
 process (clk)
 begin
 if (clk = '1' and clk'event) then
 regout <= regin;
 end if;
 end process;
end behave;
```

## Action

Make sure that a tristate driver is never permanently disabled and also make sure that there is proper resolving among tristate drivers. In the test case below, none of the tristate drivers are permanently disabled and all are properly resolved.

```
library ieee;
use ieee.std_logic_1164.all;

entity tristate is
 port (en,en1,in1,in2,in3,clk: in std_logic;
 regout:out std_logic);
end tristate;

architecture behave of tristate is
signal tmp1,regin: std_logic;
begin
 process (en, in1)
 begin
 if (en = '0') then
 tmp1 <= 'Z';
 else
 tmp1 <= in1;
 end if;
 end process;

 process (en1,in3)
 begin
 if (en1 = '0') then
 tmp1 <= 'Z';
 else
 tmp1 <= in3;
 end if;
 end process;

 regin <= tmp1 and in2;
 process (clk)
 begin
 if (clk = '1' and clk'event) then
 regout <= regin;
 end if;
 end process;
end behave;
```

# MO113

## @W: Unrecognized value in syn\_hier attribute: <firm1>.

The mapper encountered an unrecognized value for the syn\_hier attribute. In the test case below, an illegal value (firm1) is specified for the syn\_hier attribute which results in the above warning.

```
module inc(a_in, a_out) /* synthesis syn_hier = "firm1" */;
 input [3:0] a_in;
 output [3:0] a_out;
 assign a_out = a_in + 1;
endmodule

module reg4(clk, rst, d, q);
 input [3:0] d;
 input clk, rst;
 output [3:0] q;
 reg [3:0] q;

 always @(posedge clk or posedge rst)
 if(rst)
 q <= 0;
 else
 q <= d;
endmodule

module test_syn_hier(clk, rst, q);
 input clk, rst;
 output [3:0] q;
 wire [3:0] a_in;
 inc i1(q, a_in);
 reg4 r1(clk, rst, a_in, q);
endmodule
```

### Action

The mappers perform default optimization across the hierarchy. To control optimization, a valid value (firm, soft, remove, hard, or flatten) must be specified for the syn\_hier attribute. In the modified test case below, the legal value (firm) is specified for the syn\_hier attribute which eliminates the above warning.

```
module inc(a_in, a_out)/* synthesis syn_hier = "firm" */;
 input [3:0] a_in;
 output [3:0] a_out;
 assign a_out = a_in + 1;
endmodule

module reg4(clk, rst, d, q);
 input [3:0] d;
 input clk, rst;
 output [3:0] q;
 reg [3:0] q;

 always @(posedge clk or posedge rst)
 if(rst)
 q <= 0;
 else
 q <= d;
endmodule

module test_syn_hier(clk, rst, q);
 input clk, rst;
 output [3:0] q;
 wire [3:0] a_in;
 inc i1(q, a_in);
 reg4 r1(clk, rst, a_in, q);
endmodule
```

## MO114

### **@W: Found both macro and flatten in syn\_hier attribute on instance.**

The mapper encountered conflicting values of both macro and flatten for the `syn_hier` attribute. In the test case below, macro and flatten are both specified by the `syn_hier` attribute which results in the above warning.

#### Test Case

```
module inc(a_in, a_out)
 /* synthesis syn_hier = "macro,flatten" */;
 input [3:0] a_in;
 output [3:0] a_out;
endmodule
```

```

module reg4(clk, rst, d, q);
 input [3:0] d;
 input clk, rst;
 output [3:0] q;
 reg [3:0] q;

 always @(posedge clk or posedge rst)
 if(rst)
 q <= 0;
 else
 q <= d;
endmodule

module top(clk, rst, q);
 input clk, rst;
 output [3:0] q;
 wire [3:0] a_in;
 inc i1(q, a_in);
 reg4 r1(clk, rst, a_in, q);
endmodule

```

## Action

Provide a valid value for the `syn_hier` attribute which can be soft (default), firm, hard, remove, macro, or flatten individually or combined with soft, firm, or remove.

# MO115

### **@W: Found both macro and remove in syn\_hier attribute on instance.**

The mapper encountered conflicting values of both macro and remove for the `syn_hier` attribute. In the test case below, macro and remove are both specified for the `syn_hier` attribute which results in the above warning.

## Test Case

```

module inc(a_in, a_out)
 /* synthesis syn_hier = "macro,remove" */;
 input [3:0] a_in;
 output [3:0] a_out;
endmodule

```

```
module reg4(clk, rst, d, q);
 input [3:0] d;
 input clk, rst;
 output [3:0] q;
 reg [3:0] q;

 always @(posedge clk or posedge rst)
 if(rst)
 q <= 0;
 else
 q <= d;
 endmodule

module top(clk, rst, q);
 input clk, rst;
 output [3:0] q;
 wire [3:0] a_in;
 inc i1(q, a_in);
 reg4 r1(clk, rst, a_in, q);
endmodule
```

## Action

Provide a valid value for the `syn_hier` attribute which can be soft (default), firm, hard, remove, macro, or flatten individually or combined with soft, firm, or remove. In the corrected test case below, a valid value of macro is used for the attribute.

```
module inc(a_in, a_out) /* synthesis syn_hier = "macro" */;
 input [3:0] a_in;
 output [3:0] a_out;
endmodule

module reg4(clk, rst, d, q);
 input [3:0] d;
 input clk, rst;
 output [3:0] q;
 reg [3:0] q;

 always @(posedge clk or posedge rst)
 if(rst)
 q <= 0;
 else
 q <= d;
 endmodule
```

```
module top(clk, rst, q);
 input clk, rst;
 output [3:0] q;
 wire [3:0] a_in;
 inc i1(q, a_in);
 reg4 r1(clk, rst, a_in, q);
endmodule
```

## MO116

### **@W: Found both macro and firm in syn\_hier attribute on instance.**

The mapper encountered conflicting values of both macro and firm for the syn\_hier attribute. In the test case below, macro and firm are both specified for the syn\_hier attribute which results in the above warning.

#### Test Case

```
module inc(a_in, a_out) /* synthesis syn_hier = "macro,firm" */;
 input [3:0] a_in;
 output [3:0] a_out;
endmodule

module reg4(clk, rst, d, q);
 input [3:0] d;
 input clk, rst;
 output [3:0] q;
 reg [3:0] q;

 always @(posedge clk or posedge rst)
 if(rst)
 q <= 0;
 else
 q <= d;
endmodule
```

```
module top(clk, rst, q);
 input clk, rst;
 output [3:0] q;
 wire [3:0] a_in;
 inc i1(q, a_in);
 reg4 r1(clk, rst, a_in, q);
endmodule
```

## Action

Provide a valid value for the syn\_hier attribute which can be soft (default), firm, hard, remove, macro, or flatten individually or combined with soft, firm, or remove. In the corrected test case below, a valid value of macro is used for the attribute.

```
module inc(a_in, a_out) /* synthesis syn_hier = "macro" */;
 input [3:0] a_in;
 output [3:0] a_out;
endmodule

module reg4(clk, rst, d, q);
 input [3:0] d;
 input clk, rst;
 output [3:0] q;
 reg [3:0] q;

 always @ (posedge clk or posedge rst)
 if(rst)
 q <= 0;
 else
 q <= d;
endmodule

module top(clk, rst, q);
 input clk, rst;
 output [3:0] q;
 wire [3:0] a_in;
 inc i1(q, a_in);
 reg4 r1(clk, rst, a_in, q);
endmodule
```

# MO117

## @W: Found both macro and locked in syn\_hier attribute on instance.

The mapper encountered conflicting values of both macro and locked for the syn\_hier attribute. In the test case below, macro and locked are both specified for the syn\_hier attribute which results in the above warning.

### Test Case

```
module inc(a_in, a_out)
 /* synthesis syn_hier = "macro,locked" */;
 input [3:0] a_in;
 output [3:0] a_out;
endmodule

module reg4(clk, rst, d, q);
 input [3:0] d;
 input clk, rst;
 output [3:0] q;
 reg [3:0] q;

 always @(posedge clk or posedge rst)
 if(rst)
 q <= 0;
 else
 q <= d;
endmodule

module top(clk, rst, q);
 input clk, rst;
 output [3:0] q;
 wire [3:0] a_in;
 inc i1(q, a_in);
 reg4 r1(clk, rst, a_in, q);
endmodule
```

### Action

Provide a valid value for the syn\_hier attribute which can be soft (default), firm, hard, remove, macro, or flatten individually or combined with soft, firm, or remove. In the corrected test case below, a valid value of macro is used for the attribute.

```
module inc(a_in, a_out) /* synthesis syn_hier = "macro" */;
 input [3:0] a_in;
 output [3:0] a_out;
endmodule

module reg4(clk, rst, d, q);
 input [3:0] d;
 input clk, rst;
 output [3:0] q;
 reg [3:0] q;

 always @(posedge clk or posedge rst)
 if(rst)
 q <= 0;
 else
 q <= d;
endmodule

module top(clk, rst, q);
 input clk, rst;
 output [3:0] q;
 wire [3:0] a_in;
 inc i1(q, a_in);
 reg4 r1(clk, rst, a_in, q);
endmodule
```

The value locked for a `syn_hier` attribute is inferred when creating a locked type compile point.

## MO118

### **@W: Found both macro and soft in `syn_hier` attribute on instance.**

The mapper encountered conflicting values of both macro and soft for the `syn_hier` attribute. In the test case below, macro and soft are both specified for the `syn_hier` attribute which results in the above warning.

## Test Case

```

module inc(a_in, a_out) /* synthesis syn_hier = "macro,soft" */;
 input [3:0] a_in;
 output [3:0] a_out;
endmodule

module reg4(clk, rst, d, q);
 input [3:0] d;
 input clk, rst;
 output [3:0] q;
 reg [3:0] q;

 always @(posedge clk or posedge rst)
 if(rst)
 q <= 0;
 else
 q <= d;
endmodule

module top(clk, rst, q);
 input clk, rst;
 output [3:0] q;
 wire [3:0] a_in;
 inc i1(q, a_in);
 reg4 r1(clk, rst, a_in, q);
endmodule

```

## Action

Provide a valid value for the `syn_hier` attribute which can be soft (default), firm, hard, remove, macro, or flatten individually or combined with soft, firm, or remove. In the corrected test case below, a valid value of macro is used for the attribute.

```

module inc(a_in, a_out) /* synthesis syn_hier = "macro" */;
 input [3:0] a_in;
 output [3:0] a_out;
endmodule

module reg4(clk, rst, d, q);
 input [3:0] d;
 input clk, rst;
 output [3:0] q;
 reg [3:0] q;

```

```
always @(posedge clk or posedge rst)
 if(rst)
 q <= 0;
 else
 q <= d;
endmodule

module top(clk, rst, q);
 input clk, rst;
 output [3:0] q;
 wire [3:0] a_in;
 inc i1(q, a_in);
 reg4 r1(clk, rst, a_in, q);
endmodule
```

## MO119

### **@W: Found both hard and firm in syn\_hier attribute on instance.**

The mapper encountered conflicting values of both hard and firm for the syn\_hier attribute. In the test case below, hard and firm are both specified for the syn\_hier attribute which results in the above warning.

```
module inc(a_in, a_out) /* synthesis syn_hier = "hard,firm" */;
 input [3:0] a_in;
 output [3:0] a_out;
 assign a_out = a_in + 1;
endmodule

module reg4(clk, rst, d, q);
 input [3:0] d;
 input clk, rst;
 output [3:0] q;
 reg [3:0] q;

 always @(posedge clk or posedge rst)
 if(rst)
 q <= 0;
 else
 q <= d;
endmodule
```

```
module test_syn_hier(clk, rst, q);
 input clk, rst;
 output [3:0] q;
 wire [3:0] a_in;
 inc i1(q, a_in);
 reg4 r1(clk, rst, a_in, q);
endmodule
```

## Action

Provide a valid value for the `syn_hier` attribute which can be soft (default), firm, hard, remove, macro, or flatten individually or combined with soft, firm, or remove. In the corrected test case below, a valid value of hard is used for the attribute.

```
module inc(a_in, a_out) /* synthesis syn_hier = "hard" */;
 input [3:0] a_in;
 output [3:0] a_out;
 assign a_out = a_in + 1;
endmodule

module reg4(clk, rst, d, q);
 input [3:0] d;
 input clk, rst;
 output [3:0] q;
 reg [3:0] q;

 always @(posedge clk or posedge rst)
 if(rst)
 q <= 0;
 else
 q <= d;
endmodule

module test_syn_hier(clk, rst, q);
 input clk, rst;
 output [3:0] q;
 wire [3:0] a_in;
 inc i1(q, a_in);
 reg4 r1(clk, rst, a_in, q);
endmodule
```

# MO120

## @W: Found both hard and remove in syn\_hier attribute on instance.

The mapper encountered conflicting values of both hard and remove for the syn\_hier attribute. In the test case below, hard and remove are both specified for the syn\_hier attribute which results in the above warning.

```
module inc(a_in, a_out)
 /* synthesis syn_hier = "hard,remove" */;
 input [3:0] a_in;
 output [3:0] a_out;
 assign a_out = a_in + 1;
endmodule

module reg4(clk, rst, d, q);
 input [3:0] d;
 input clk, rst;
 output [3:0] q;
 reg [3:0] q;

 always @(posedge clk or posedge rst)
 if(rst)
 q <= 0;
 else
 q <= d;
endmodule

module test_syn_hier(clk, rst, q);
 input clk, rst;
 output [3:0] q;
 wire [3:0] a_in;
 inc i1(q, a_in);
 reg4 r1(clk, rst, a_in, q);
endmodule
```

### Action

Provide a valid value for the syn\_hier attribute which can be soft (default), firm, hard, remove, macro, or flatten individually or combined with soft, firm, or remove. In the corrected test case below, a valid value of hard is used for the attribute.

```
module inc(a_in, a_out) /* synthesis syn_hier = "hard" */;
 input [3:0] a_in;
 output [3:0] a_out;
 assign a_out = a_in + 1;
endmodule

module reg4(clk, rst, d, q);
 input [3:0] d;
 input clk, rst;
 output [3:0] q;
 reg [3:0] q;

 always @(posedge clk or posedge rst)
 if(rst)
 q <= 0;
 else
 q <= d;
endmodule

module test_syn_hier(clk, rst, q);
 input clk, rst;
 output [3:0] q;
 wire [3:0] a_in;
 inc i1(q, a_in);
 reg4 r1(clk, rst, a_in, q);
endmodule
```

## MO122

### **@W: Found both locked and hard in syn\_hier attribute on instance.**

The mapper encountered conflicting values of both locked and hard for the syn\_hier attribute. In the test case below, locked and hard are both specified for the syn\_hier attribute which results in the above warning.

```
module inc(a_in, a_out)
 /* synthesis syn_hier = "locked,hard" */;
 input [3:0] a_in;
 output [3:0] a_out;
endmodule
```

```
module reg4(clk, rst, d, q);
 input [3:0] d;
 input clk, rst;
 output [3:0] q;
 reg [3:0] q;

 always @(posedge clk or posedge rst)
 if(rst)
 q <= 0;
 else
 q <= d;
 endmodule

module top(clk, rst, q);
 input clk, rst;
 output [3:0] q;
 wire [3:0] a_in;
 inc i1(q, a_in);
 reg4 r1(clk, rst, a_in, q);
endmodule
```

## Action

Provide a valid value for the `syn_hier` attribute which can be soft (default), firm, hard, remove, macro, or flatten individually or combined with soft, firm, or remove. In the corrected test case below, a valid value of fixed is used for the attribute.

```
module inc(a_in, a_out) /* synthesis syn_hier = "fixed" */;
 input [3:0] a_in;
 output [3:0] a_out;
endmodule

module reg4(clk, rst, d, q);
 input [3:0] d;
 input clk, rst;
 output [3:0] q;
 reg [3:0] q;

 always @(posedge clk or posedge rst)
 if(rst)
 q <= 0;
 else
 q <= d;
 endmodule
```

```
module top(clk, rst, q);
 input clk, rst;
 output [3:0] q;
 wire [3:0] a_in;
 inc i1(q, a_in);
 reg4 r1(clk, rst, a_in, q);
endmodule
```

The value **locked** for a **syn\_hier** attribute is inferred when creating a locked type compile point.

## MO123

### **@W: Found both hard and soft in syn\_hier attribute on instance.**

The mapper encountered conflicting values of both hard and soft for the **syn\_hier** attribute. In the test case below, hard and soft are both specified for the **syn\_hier** attribute which results in the above warning.

```
module inc(a_in, a_out) /* synthesis syn_hier = "hard,soft" */;
 input [3:0] a_in;
 output [3:0] a_out;
 assign a_out = a_in + 1;
endmodule

module reg4(clk, rst, d, q);
 input [3:0] d;
 input clk, rst;
 output [3:0] q;
 reg [3:0] q;

 always @(posedge clk or posedge rst)
 if(rst)
 q <= 0;
 else
 q <= d;
endmodule
```

```
module test_syn_hier(clk, rst, q);
 input clk, rst;
 output [3:0] q;
 wire [3:0] a_in;
 inc i1(q, a_in);
 reg4 r1(clk, rst, a_in, q);
endmodule
```

## Action

Provide a valid value for the syn\_hier attribute which can be soft (default), firm, hard, remove, macro, or flatten individually or combined with soft, firm, or remove. In the corrected test case below, a valid value of flatten,soft is used for the attribute.

```
module inc(a_in, a_out) /* synthesis syn_hier = "flatten,soft" */;
 input [3:0] a_in;
 output [3:0] a_out;
 assign a_out = a_in + 1;
endmodule

module reg4(clk, rst, d, q);
 input [3:0] d;
 input clk, rst;
 output [3:0] q;
 reg [3:0] q;

 always @(posedge clk or posedge rst)
 if(rst)
 q <= 0;
 else
 q <= d;
endmodule

module test_syn_hier(clk, rst, q);
 input clk, rst;
 output [3:0] q;
 wire [3:0] a_in;
 inc i1(q, a_in);
 reg4 r1(clk, rst, a_in, q);
endmodule
```

# MO125

## @W: \*Duplicate enable for tristate driver <un2\_y[7]> on <net y\_1[7]>!

The mapper detected that the multiple tristates driving a net have the same enable with the same value (i.e., duplicate enable). In the following test case, y is driven twice and is given the duplicate enable en which results in the above warning.

```
module tristate(a,b,en, y);
 input [7:0] a, b;
 input en;
 output [7:0] y;
 assign y= en ? a :8'hzz ;
 assign y= en ? b :8'hzz ;
endmodule
```

### Action

Make sure that all tristate drivers on the same net do not have duplicate enables. In the corrected test case below, the functionality of the enable is not same.

```
module tristate(a,b,en, y);
 input [7:0] a, b;
 input en;
 output [7:0] y;
 assign y= en ? a :8'hzz ;
 assign y= en ? 8'hzz : b ;
endmodule
```

## MO129

### @W: Sequential instance <state[2]> is reduced to a combinational gate by constant propagation.

A sequential instance present in the RTL view (srs) always outputs a constant value (i.e., its input is tied to a constant). The instance is optimized during mapping and replaced with combinational logic as required. In the test case below, state register state[2] always propagates a value 0 to the output which results in the above warning.

```
module FSM1 (clk, in1, rst, out1);
 input clk, rst, in1;
 output [2:0] out1;
 `define s0 3'b000
 `define s1 3'b001
 `define s2 3'b010
 `define s3 3'bxxx
 reg [2:0] out1;
 reg [2:0] state /* synthesis syn_state_machine = 0 */;
 reg [2:0] next_state;

 always @ (posedge clk or posedge rst)
 if (rst) state <= `s0;
 else state <= next_state;

 // Combined Next State and Output Logic
 always @ (state or in1)
 case (state)
 `s0 : begin
 out1 <= 3'b000;
 if (in1) next_state <= `s1;
 else next_state <= `s0;
 end
 `s1 : begin
 out1 <= 3'b001;
 if (in1) next_state <= `s2;
 else next_state <= `s1;
 end
 `s2 : begin
 out1 <= 3'b010;
 if (in1) next_state <= `s3;
 else next_state <= `s2;
 end
 endcase
 end
```

```
default : begin
 out1 <= 3'bxxx;
 next_state <= `s0;
end
endcase
endmodule
```

## Action

The above optimization is done as part of sequential optimization to improve the resource utilization (decreased register count). To preserve a sequential instance from being optimized away to conserve the LUT or combinational resource utilization, turn off sequential optimization by checking Disable Sequential Optimization on the Device tab of the Options for implementation dialog box.

Disable Sequential Optimization is a global option that disables all sequential optimizations for a design and is NOT ADVISED.

The above test case, when run with the sequential optimization enabled, increases the number of registers used by one, but does not give the above warning.

# MO144

## **@W: Found both locked and fixed in syn\_hier attribute on instance.**

The mapper encountered conflicting values of both locked and fixed for the syn\_hier attribute. In the test case below, locked and fixed are both specified for the syn\_hier attribute which results in the above warning.

```
module inc(a_in, a_out)
 /* synthesis syn_hier = "locked,fixed" */;
 input [3:0] a_in;
 output [3:0] a_out;
endmodule
```

```
module reg4(clk, rst, d, q);
 input [3:0] d;
 input clk, rst;
 output [3:0] q;
 reg [3:0] q;

 always @(posedge clk or posedge rst)
 if(rst)
 q <= 0;
 else
 q <= d;
 endmodule

module test(clk, rst, q);
 input clk, rst;
 output [3:0] q;
 wire [3:0] a_in;
 inc i1(q, a_in);
 reg4 r1(clk, rst, a_in, q);
endmodule
```

## Action

Provide a valid value for the `syn_hier` attribute which can be soft (default), firm, hard, remove, macro, or flatten individually or combined with soft, firm, or remove. In the corrected test case below, a valid value of fixed is used for the attribute.

```
module inc(a_in, a_out) /* synthesis syn_hier = "fixed" */;
 input [3:0] a_in;
 output [3:0] a_out;
endmodule

module reg4(clk, rst, d, q);
 input [3:0] d;
 input clk, rst;
 output [3:0] q;
 reg [3:0] q;

 always @(posedge clk or posedge rst)
 if(rst)
 q <= 0;
 else
 q <= d;
 endmodule
```

```
module test(clk, rst, q);
 input clk, rst;
 output [3:0] q;
 wire [3:0] a_in;
 inc i1(q, a_in);
 reg4 r1(clk, rst, a_in, q);
endmodule
```

The value locked for a syn\_hier attribute is inferred when creating a locked type compile point.

## MO160

**@W: Register bit <bit> (in view <view>) is always 0. To keep the instance, apply syn\_preserve=1 on the instance.**

A register with a constant fixed value of 0 was detected and is being removed.

### Action

The warning identifies the register being removed. An unchanging value of 0 results when the conditions of the clock, data, and control signals and the initial value cannot result in an output value of 1.

## MO161

**@W: Register bit <bit> (in view <view>) is always 1. To keep the instance, apply syn\_preserve=1 on the instance.**

A register with a constant fixed value of 1 was detected and is being removed.

### Action

The warning identifies the register being removed. An unchanging value of 1 results when the conditions of the clock, data, and control signals and the initial value cannot result in an output value of 0.

## MO171

**@W: Sequential instance <*instanceName*> reduced to a combinational gate by constant propagation.**

A sequential element with its output tracking its input was detected and is being replaced with combinational gating.

### Action

The warning identifies an unnecessary sequential instance which typically results from a transparent latch with a permanently “open” gate. The latch is replaced with a simple buffer.

-



## CHAPTER 29

# MT Messages 116 – 623

---

## MT116

**@W: Paths from clock (<c/k1:r>) to clock (<c/k2:r>) are overconstrained because the required time of <0.53> ns is too small.**

A very short delta period exists between two clock paths belonging to two clocks within the same clock group with both clocks active on the rising edge. In the following test case, 100 MHz clock clk1 and 95 MHz clk2 are constrained to be in the same clock group, but have only a 5 MHz (0.526 ns) delta as defined in the clocks section of the constraint file.

```
create_clock -name {clk1} {p:clk1} -period 10
create_clock -name {clk2} {p:clk2} -period 10.526
```

### Test Case

```
module test (clk1, clk2, a, b, c, d);
 input clk1, clk2, a, b;
 output c, d;
 reg c, d;
 wire temp;
 reg regtemp1, regtemp2;
 assign temp = clk1 & clk2;
```

```
always @ (posedge temp)
begin
 c <= a & b;
end

always @ (posedge clk1)
begin
 regtemp1 <= a;
 regtemp2 <= b;
end

always @ (posedge clk2)
begin
 d <= regtemp1 | regtemp2;
end

endmodule
```

## Action

Analyze the path between these two clocks and define the clock constraints so they are exact multiples of each other. In the test case, changing the period for clk2 to 12.5 ns (80 MHz) or 8.0 ns (125 MHz) eliminates the warning in the example:

```
create_clock -name {clk1} {p:clk1} -period 10
create_clock -name {clk2} {p:clk2} -period 12.5

create_clock -name {clk1} {p:clk1} -period 10
create_clock -name {clk2} {p:clk2} -period 8
```

Make sure that the paths are actual paths and not false paths. If the clocks do not originate from the same clock source (non-synchronized clocks), place the clocks in separate clock groups so that the tool ignores the cross-clocked paths.

## MT117

**@W: Paths from clock (<clk>:r) to clock (<clk1>:f) are overconstrained because the required time of <0.53> ns is too small.**

A very short delta period exists between two clock paths belonging to two clocks within the same clock group. In the following test case, 100 MHz rising clock clk and 95 MHz falling clk1 are constrained to be in the same clock group, but have only a 5 MHz (0.526 ns) delta as defined in the clocks section of the constraint file.

```
create_clock -name {clk} {p:clk} -period {10}
create_clock -name {clk1} {p:clk1} -period {10.526}
```

### Test Case

```
module reg_reg (a,clk,clk1,qa,qb);
 input clk,clk1;
 input a;
 output qa,qb;
 reg qa;
 reg qb;

 always@ (posedge clk)
 qa <= a;

 always@ (negedge clk1)
 qb <= qa;

endmodule
```

### Action

Analyze the path between the two clocks and define the clock constraints so they are exact multiples of each other. In the test case, changing the period for clk2 to 13.33 ns (75 MHz) or 8.0 ns (125 MHz) eliminates the warning in the example.

```
create_clock -name {clk} {p:clk} -period 10
create_clock -name {clk1} {p:clk1} -period 13.33

create_clock -name {clk} {p:clk} -period 10
create_clock -name {clk1} {p:clk1} -period 8
```

Make sure that the paths are actual paths and not false paths. If the clocks do not originate from the same clock source (non-synchronized clocks), place the clocks in separate clock groups so that the tool ignores the cross-clocked paths.

## MT118

**@W: Paths from clock (<clk>:f) to clock (<clk1>:r) are overconstrained because the required time of <0.26> ns is too small.**

A very short delta period exists between two clock paths belonging to two clocks within the same clock group. In the following test case, 100 MHz falling clock clk and 95 MHz rising clk1 are constrained to be in the same clock group, but have only a 5 MHz delta as defined in the clocks section of the constraint file.

```
create_clock -name {clk} {p:clk} -period {10}
create_clock -name {clk1} {p:clk1} -period {10.526}
```

### Test Case

```
module reg_reg (a,clk,clk1,qa,qb);
 input clk,clk1;
 input a;
 output qa,qb;
 reg qa;
 reg qb;

 always@ (negedge clk)
 qa <= a;

 always@ (posedge clk1)
 qb <= qa;

endmodule
```

## Action

Analyze the path between the two clocks and define the clock constraints so that they are exact multiples of each other. In the test case, changing the period for clk2 to 12.5 ns (80 MHz) or 7.5 ns (133.33 MHz) eliminates the warning in the example.

```
create_clock -name {clk} {p:clk} -period 10
create_clock -name {clk1} {p:clk1} -period 5
```

Make sure that these are real paths and not false paths. If the clocks do not originate from the same clock source (non-synchronized clocks), place the clocks in separate clock groups (when clocks are placed in different clock groups, the tool ignores the cross-clocked paths).

# MT119

**@W: Paths from clock (<clk>:f) to clock (<clk1>:f) are overconstrained because the required time of <0.26> ns is too small.**

A very short delta period exists between two clock paths belonging to two clocks within the same clock group with both clocks active on the falling edge. In the following test case, 100 MHz clock clk and 95 MHz clk1 are constrained to be in the same clock group, but have only a 5 MHz (0.526 ns) delta as defined in the clocks section of the constraint file.

```
create_clock -name {clk} {p:clk} -period {10}
create_clock -name {clk1} {p:clk1} -period {10.526}
```

## Test Case

```
module reg_reg (a,clk,clk1,qa,qb);
 input clk,clk1;
 input a;
 output qa,qb ;
 reg qa;
 reg qb;

 always@ (negedge clk)
 qa <= a;
```

```

always@ (negedge clk1)
 qb <= qa;
endmodule

```

## Action

Analyze the path between the two clocks and define the clock constraints so that they are exact multiples of each other. In the test case, changing the period for clk2 to 15 ns (66.67 MHz) or 6.67 ns (150 MHz) eliminates the warning in the example.

```

create_clock -name {clk} {p:clk} -period 10
create_clock -name {clk1} {p:clk1} -period 5

```

Also make sure that these are real paths and not false paths. If the clocks do not originate from the same clock source (non-synchronized clocks), place the clocks in separate clock groups (when clocks are placed in different clock groups, the tool ignores the cross-clocked paths).

## MT120

### **@W: Clocks are too close. All paths between clocks are considered false**

Clocks belonging to the same clock group have time periods differing by a small delta that prevents paths between them from being analyzed. In the following test case, the clock frequencies of clk1 and clk2 are too close in value.

```

module test (clk1, clk2, a, b, c, d);
 input clk1, clk2, a, b;
 output c, d;
 reg c, d;
 wire temp;
 reg regtemp1, regtemp2;
 assign temp = clk1 & clk2;

 always @ (posedge temp)
 begin
 c <= a & b;
 end

```

```

always @ (posedge clk1)
begin
 regtemp1 <= a;
 regtemp2 <= b;
end

always @ (posedge clk2)
begin
 d <= regtemp1 | regtemp2;
end

endmodule

```

### Constraints Applied

```

create_clock -name {clk1} p:clk1 -period 100
create_clock -name {clk2} p:clk2 -period 100.5

```

### Action

In the above test case, cross clocked paths between clk1 and clk2 are not analyzed due to the clocks being too close in frequency (delta period = 0.5 ns). This usually occurs when the two clocks specified are not exact multiples of each other.

Define clock constraints so that the specified time periods allow for cross clock path analysis. To eliminate the warning in the above test case (and to avoid warning [MT116](#)), change the clk2 period to 125 ns.

### Constraints Applied

```

create_clock -name {clk1} p:clk1 -period 100
create_clock -name {clk2} p:clk2 -period 125

```

## MT122

### **@E: Non-positive distance between clocks <clk[1]:<r> and <clk[2]:<r> detected**

A 0 or negative delay was detected between two clock paths belonging to two clocks within the same clock group.

For the test case below, the constraint file contains the following entries:

```
Clocks
create_clock -name {clk1} {p:clk1} -period {10}
create_clock -name {clk2} {p:clk2} -period {5}

Delay Paths
set_clock_route_delay [get_clocks {clk1} {clk2}] {5.5}
```

The value of the `clock_route_delay` (5.5 ns) is greater than the `clk2` period which causes the mapper to error out with the above message.

## Test Case

```
module test (clk1, clk2, a, b, c, d);
 input clk1, clk2, a, b;
 output c, d;
 reg c, d;
 wire temp;
 reg regtemp1, regtemp2;
 assign temp = clk1 & clk2;

 always @ (posedge temp)
 begin
 c <= a & b;
 end

 always @ (posedge clk1)
 begin
 regtemp1 <= a;
 regtemp2 <= b;
 end

 always @ (posedge clk2)
 //always @ (negedge clk2)
 begin
 d <= regtemp1 | regtemp2;
 end

endmodule
```

## Action

A positive delay for the path between two clocks of the same clock group must be specified as shown in the constraint file entries below.

```
Clocks
create_clock -name {clk1} {p:clk1} -period {10}
create_clock -name {clk2} {p:clk2} -period {5}

#Delay Paths
set_clock_route_delay [get_clocks {clk1} {clk2}] {2.4}
```

## MT204

### **@W: Auto Constrain mode is disabled because clocks are defined in the FDC file.**

Auto constraining, when enabled, causes the mapper to perform multiple iterations with different clock constraints on the critical paths to achieve the best possible result. This warning occurs if the auto constraint feature is enabled for the design, but the constraint file, included in the project, has defined clock constraints.

In the following test case, auto constraining is enabled, but the constraint file includes a create\_clock constraint.

```
module inc(a_in, a_out);
 input [3:0] a_in;
 output [3:0] a_out;
 assign a_out = a_in + 1;
endmodule

module reg4(clk, rst, d, q);
 input [3:0] d; input clk, rst;
 output [3:0] q; reg [3:0] q;

 always @(posedge clk or posedge rst)
 if(rst)
 q <= 0;
 else
 q <= d;
endmodule

module test (clk, rst, q);
 input clk, rst; output [3:0] q;
 wire [3:0] a_in; inc i1(q, a_in);
 reg4 r1(clk, rst, a_in, q);
endmodule
```

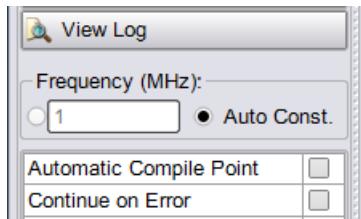
## Constraint File Contents

```
Clocks
#create_clock -name {clk} {p:clk} -period 10
```

## Action

Auto constraining is disabled when clock constraints and/or I/O constraints are defined in the constraint file. To use the auto constraining feature, you must remove all clock and I/O constraints from the constraint file and enable the auto constrain feature.

Auto constraining is enabled by checking the Auto Const radio button in the UI (see below) or by including a -frequency auto option (`set_option -frequency auto`) in the project file.

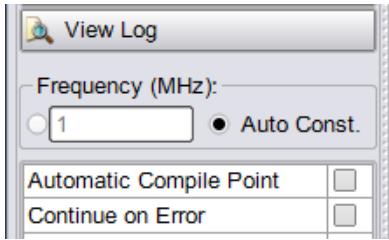


Auto constraining is supported only for some technologies.

## MT206

### **@N: Auto Constrain mode is enabled**

The auto constraint feature has been enabled for the design. When auto constraining is enabled, the mapper performs multiple iterations with different clock constraints on the critical paths to achieve the best possible result. Auto constraining is enabled by checking the Auto Constrain radio button in the UI (see below) or by including a -frequency auto option (`set_option -frequency auto`) in the project file.



Auto constraining is supported only for some technologies.

Auto constraining works on register-to-register paths unless the tool is set up to also time I/O paths by selecting the Use clock period for unconstrained IO checkbox on the Constraints tab of the Implementation Options dialog box. Auto constraining is automatically disabled as soon as the constraint file in the project includes clock or I/O constraints.

## MT246

**@W: Blackbox <bb\_name> is missing a user supplied timing model.  
This may have a negative effect on timing analysis and  
optimizations (Quality of Results)**

A user-defined black box or an unbound component was found in the design and a timing model was not provided.

### Action

Provide a timing model for each black box to ensure the tool performs correct timing analysis on paths associated with the black box and achieves the best possible timing results. It is recommended that you specify all timing information for black boxes. Black box timing models are specified as propagation delays through the black box:

- syn\_tsu – setup delay (relative to the clock) for input pins
- syn\_tco – clock-to-output delay
- syn\_tpd – combinational delay through the black box

The following example illustrates the use of black box timing models:

```
module ram32x4 (z, d, addr, we, clk)
 /* synthesis syn_black_box
 syn_tpd1="addr[3:0]->z[3:0]=8.0"
 syn_tsu1="addr[3:0]->clk=2.0"
 syn_tsu2="we->clk=3.0" */;
 output [3:0] z;
 input [3:0] d;
 input [3:0] addr;
 input we;
 input clk;
endmodule
```

## MT320

**@N: This timing report is an estimate of place and route data. For final timing results, use the FPGA vendor place and route report.**

The timing reported in the log file is an estimation of the actual timing when the design is placed and routed. The log report from the place and route tool provides the exact timing values for the design.

### Action

The timing estimates in the log file should approximate the timing reported by the place and route tool. If there is discrepancy, please report the test case with constraints to Synopsys so that further action can be taken.

## MT321

**@N: Clock constraints cover all FF-to-FF, FF-to-output, input-to-FF and input-to-output paths associated with a particular clock.**

The mapper, by default, considers only flip-flop to flip-flop paths associated with the clock when performing timing analysis. The log file shows only such paths as critical. The paths associated with the input to flip-flop, flip-flop to output, or input to output are not considered.

### Action

To override this default behavior and enable all paths to be analyzed, enable the use clock period for unconstrained IO check box in the implementations options->constraints tab, or set it in the project file with the following Tcl command:

```
implementation options
set_option -auto_constraint_io 1
```

The mapper analyzes all paths from input to flip-flop, flip-flop to output, or input to output. You can also override this default behavior by inserting I/O constraints such as `set_input_delay` and `set_output_delay` in SCOPE. When the default behavior is overridden, note MT321 appears in the log file.

## MT322

**@N: Clock constraints cover only register-to-register paths associated with the clock.**

The mapper, by default, considers only register-to-register paths associated with the clock when performing timing analysis. The log file shows only such paths as critical. The paths associated with the input to flip-flop, flip-flop to output, or input to output are not considered.

## Action

To override this default behavior and enable all paths to be analyzed, set the use clock period for unconstrained I/O options in the implementations options->constraints tab, or set it in the project file with the following Tcl command:

```
implementation options
set_option -auto constrain_io 0
```

The mapper analyzes all paths from input to flip-flop, flip-flop to output, or input to output. You can also override this default behavior by inserting I/O constraints such as `set_input_delay` and `set_output_delay` in SCOPE. When the default behavior is overridden, the note [MT321](#) appears in the log file:

# MT346

## **@W: No end clock found for end point <location>/<clock> (<rising>).**

A black box was encountered that had no associated clock.

## Action

Possible actions can be:

- Including the lower level netlist that has the “internals” of the black box. Often the tool can then read the EDIF and vendor netlists, determine the contents of the black box, and then identify the clock.
- Using the `syn_tco`, `syn_tpd`, and `syn_tsu` attributes to describe the clocks and timing arcs within the black box.

# MT403

## @W: Ignoring clock definition because instance has more than one output

A clock constraint, which is being applied to an instance instead of a net, is being ignored because there is more than one output on the clock.

Constraint File Contents:

```

Clocks
create_clock -name {i_clkgen} {i:i_clkgen} -period 10
```

## Test Case

```
module test (
 input clkin,
 input d_i,
 output d_o);
reg d_q1;
reg d_q2;
wire clk;
wire clk2;
clkgen i_clkgen (
 .clkin(clkin),
 .clk(clk),
 .clk2(clk2));

always @ (posedge clk) begin
 d_q1 <= d_i;
 d_q2 <= d_q1;
end
assign d_o = d_q2;
endmodule

module clkgen (
 input clkin,
 output clk,
 output clk2);
endmodule
```

## Action

To avoid the above warning, apply the constraint directly to the output pin of the instance by either editing the constraint file in text mode or by dragging the output net from the RTL view into the SCOPE constraints editor window and replacing the new constraints file on the Constraints panel.

The corrected create\_clock entry in the constraint file is:

```
create_clock -name {i_clkgen} {n:clk} -period 10
```

# MT418

## **@W: No clock object found for matching syn\_clock\_priority on port “<portName>”**

The port specified by a syn\_clock\_priority attribute does not have an associated clock object.

## Action

Make sure that the syn\_clock\_priority attribute is assigned to a clock port.

# MT419

## **@W: No clock object found for matching syn\_clock\_priority on net “<netName>”**

The net specified by a syn\_clock\_priority attribute does not have an associated clock object.

## Action

Make sure that the syn\_clock\_priority attribute is assigned to a clock net.

# MT420

**@W: Found inferred clock <clockName> with period <value>ns.  
Please declare a user-defined clock on object “<clockPin>”**

This warning occurs when the tool infers a clock that should have been defined but was not. It might cause a problem because the inferred frequency might not match the design requirements.

```
module ram_test (q, a, d, we, clk, clk1);
 output [7:0] q;
 input [7:0] d;
 input [6:0] a;
 input clk, clk1, we;
 reg [6:0] read_add;
 /* The array of an array register ("mem") the RAM will be
 inferred from */
 reg [7:0] mem [127:0] /* synthesis syn_ramstyle = "block_ram" */;
 assign q = mem[read_add];

 always @(posedge clk) begin
 if(we)
 /* Register RAM Data */
 mem[a] <= d;
 end

 always @(posedge clk1) begin
 read_add <= a;
 end
endmodule
```

## Action

Open the RTL view and check for the missing clock definition. If it is missing, declare the inferred clock by adding a `create_clock` timing constraint to the constraint file for the specified clock pin, and re-synthesize the design.

# MT436

**@W: Clock "<c:clk1>" has not been defined. Timing constraints using this clock will be ignored.**

A constraint references a clock object and that clock was not defined in the constraint file. For example, if the constraint file includes an I/O delay constraint referencing the rising edge of clock object c:clk, but there is no clock object defined by this name, the above message is displayed and the constraint is ignored.

## Constraint File Contents

```
Clocks
create_clock -name {clk} {p:clk} -period {10}

Inputs/Outputs
set_input_delay [all_inputs] 1.00 -clock {c:clk1}
```

## Test Case

```
module test (d,clk,q);
 input d,clk;
 output reg q;

 always @(posedge clk)
 q=d;
endmodule
```

## Action

Before referencing any clock object in the constraint file, make sure that the clock is defined properly with the same name as shown in the constraint file entries below.

```
##Constraint file ##
create_clock -name {clk} {p:clk} -period {10}
set_output_delay [all_inputs] 1.00 -clock {c:clk}
```

## MT443

**@W: <location> Timing constraint (*constraintDescriptor*) (max delay <value>) was not applied to the design because <reason>**

The mapper cannot apply the maximum path delay constraint on a given path (i.e., the path specified in the through point does not exist or the same path is constrained as a false path).

In the following test case, the constraint file includes the following entries:

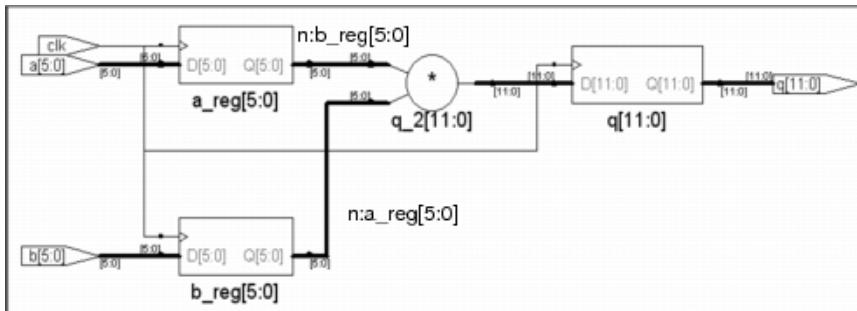
```
Clocks
create_clock -name {clk} {p:clk} -period {10}

Path Delay
set_max_delay -through {n:a_reg[5:0]} -through {n:b_reg[5:0]} 2.0
```

The path delay entry in the constraint file constrains the path through n:a\_reg[5:0] and n:b\_reg[5:0] to 2 ns. However, because there is no path through both nets in the RTL, the constraint is not applied, and the mapper displays the above warning with the reason the constraint could not be applied.

```
module mult (a,b,clk,q);
parameter k =5;
input [k:0] a,b;
input clk;
output reg [2*k +1 :0] q;
reg[k:0] a_reg,b_reg;

always @(posedge clk)
begin
 a_reg<= a;
 b_reg<= b;
 q<= a_reg * b_reg;
end
endmodule
```



## Action

Make sure that a valid path is specified for the maximum path delay constraint. In the example constraint file entries below,  $n:a\_reg[5:0]$  and  $n:b\_reg[5:0]$  are individually constrained.

```
Clocks
create_clock -name {clk} {p:clk} -period {10}

Path Delay
set_max_delay -through {n:a_reg[5:0]} 2.0
set_max_delay -through {n:b_reg[5:0]} 2.0
```

## Reason String

The reason explanation string can be any of the following based on the condition reported. Not all strings are possible for the described timing report messages.

1. higher priority constraint(s) were applied to the paths – Each timing path is checked for matching timing exceptions by comparing the starting point of the path with the -from part of the timing exception command. If two or more timing exceptions match a given path, only one exception is applied to that path. A priority scheme is used to determine which timing exception to apply. The priority depends on the commands and the order in which they are read.

If a timing exception matches one or more paths, but is never the highest priority timing exception for that path, this message is reported. The higher priority timing exception is reported in message [MT621](#).

2. no matching path had hold check – The timing exception affects or disables hold checks (e.g., `set_min_delay`). The timing exception matched one or more paths, but none of the paths ended on a cell pin or port that had a hold check. Accordingly, the timing exception has no effect and is ignored. An example would be a path ending at a black-box cell with no timing model.
3. no matching path had setup check – The timing exception affects or disables setup checks (e.g., `set_max_delay`). The timing exception matched one or more paths, but none of the paths began on a cell pin or port that had a setup check. Accordingly, the timing exception has no effect and is ignored. An example would be a path ending at a black-box cell with no timing model.
4. no matching path had start clock – None of the matching paths has a start clock which can occur when the start point is a port with a timing requirement (i.e., there is no `set_input_delay` command for the input port).
5. no matching path was synchronous – All of the paths that match this timing exception are asynchronous (i.e., the starting clock and ending clock for the path are asynchronous). These paths are considered to be false paths and are not timed.
6. none of the paths specified by the constraint exist in the design – After matching objects for every `-from`, `-through`, or `-to` in the command, no matching paths are found in the design. For example, consider the command:

```
set_multicycle_path -from [get_ports A] -to [get_ports Z]
```

If ports A and Z are totally disconnected and there is no path for data to get from A to Z, the timing exception does not apply.

7. the `from` list is incorrect: it contains no clock, primary input, sequential cell, or sequential cell clock pin – If the `-from` portion of a timing exception matches an object, but it is not the correct type of object, it does not apply. An example would be specifying `-from` on the output of a combinational gate.
8. the `through` list is incorrect: it contains no net or hierarchy pin – If the `-through` portion of a timing exception matches an object, but it is not the correct type of object, it does not apply. An example would be specifying a `-through` on a cell pin.
9. the `to` list is incorrect: it contains no clock, primary input, sequential cell, or sequential cell clock pin – If the `-to` portion of a timing exception matches an object,

but it is not the correct type of object, it does not apply. An example would be specifying -to on a pin of a combinational cell.

## MT444

### **@W: <location> Timing constraint (<constraintDescriptor>) (max delay (datapath only) <value>) was not applied to the design because <reason>**

The mapper cannot apply the max delay constraint on the specified path because another constraint with a higher priority applies, or because the path does not exist. For example, this message might be displayed if the same path also has a false path constraint which has a higher priority than a max\_delay constraint.

Path delay clock constraints apply to all paths of the registers clocked by the specified clock.

#### Action

Make sure that a valid path is specified for the max\_delay constraint.

#### Reason String

The reason explanation string can be any of the following based on the condition reported. Not all strings are possible for the described timing report messages.

1. higher priority constraint(s) were applied to the paths – Each timing path is checked for matching timing exceptions by comparing the starting point of the path with the -from part of the timing exception command. If two or more timing exceptions match a given path, only one exception is applied to that path. A priority scheme is used to determine which timing exception to apply. The priority depends on the commands and the order in which they are read.

If a timing exception matches one or more paths, but is never the highest priority timing exception for that path, this message is reported. The higher priority timing exception is reported in message [MT621](#).

2. no matching path had hold check – The timing exception affects or disables hold checks (e.g., `set_min_delay`). The timing exception matched one or more paths, but none of the paths ended on a cell pin or port that had a hold check. Accordingly, the timing exception has no effect and is ignored. An example would be a path ending at a black-box cell with no timing model.
3. no matching path had setup check – The timing exception affects or disables setup checks (e.g., `set_max_delay`). The timing exception matched one or more paths, but none of the paths began on a cell pin or port that had a setup check. Accordingly, the timing exception has no effect and is ignored. An example would be a path ending at a black-box cell with no timing model.
4. no matching path had start clock – None of the matching paths has a start clock which can occur when the start point is a port with a timing requirement (i.e., there is no `set_input_delay` command for the input port).
5. no matching path was synchronous – All of the paths that match this timing exception are asynchronous (i.e., the starting clock and ending clock for the path are asynchronous). These paths are considered to be false paths and are not timed.
6. none of the paths specified by the constraint exist in the design – After matching objects for every `-from`, `-through`, or `-to` in the command, no matching paths are found in the design. For example, consider the command:

```
set_multicycle_path -from [get_ports A] -to [get_ports Z]
```

If ports A and Z are totally disconnected and there is no path for data to get from A to Z, the timing exception does not apply.

7. the `from` list is incorrect: it contains no clock, primary input, sequential cell, or sequential cell clock pin – If the `-from` portion of a timing exception matches an object, but it is not the correct type of object, it does not apply. An example would be specifying `-from` on the output of a combinational gate.
8. the `through` list is incorrect: it contains no net or hierarchy pin – If the `-through` portion of a timing exception matches an object, but it is not the correct type of object, it does not apply. An example would be specifying a `-through` on a cell pin.
9. the `to` list is incorrect: it contains no clock, primary input, sequential cell, or sequential cell clock pin – If the `-to` portion of a timing exception matches an object,

but it is not the correct type of object, it does not apply. An example would be specifying -to on a pin of a combinational cell.

## MT445

### **@W: <location> Timing constraint (<constraintDescriptor>) (min delay <value>) was not applied to the design because <reason>**

The mapper cannot apply the min delay constraint on the specified path because another constraint with a higher priority applies, or because the path does not exist. For example, this message might be displayed if the same path also has a false path constraint which has a higher priority than a min\_delay constraint.

Path delay clock constraints apply to all paths of the registers clocked by the specified clock.

#### Action

Make sure that a valid path is specified for the min\_delay constraint.

#### Reason String

The reason explanation string can be any of the following based on the condition reported. Not all strings are possible for the described timing report messages.

1. higher priority constraint(s) were applied to the paths – Each timing path is checked for matching timing exceptions by comparing the starting point of the path with the -from part of the timing exception command. If two or more timing exceptions match a given path, only one exception is applied to that path. A priority scheme is used to determine which timing exception to apply. The priority depends on the commands and the order in which they are read.

If a timing exception matches one or more paths, but is never the highest priority timing exception for that path, this message is reported. The higher priority timing exception is reported in message [MT621](#).

2. no matching path had hold check – The timing exception affects or disables hold checks (e.g., `set_min_delay`). The timing exception matched one or more paths, but none of the paths ended on a cell pin or port that had a hold check. Accordingly, the timing exception has no effect and is ignored. An example would be a path ending at a black-box cell with no timing model.
3. no matching path had setup check – The timing exception affects or disables setup checks (e.g., `set_max_delay`). The timing exception matched one or more paths, but none of the paths began on a cell pin or port that had a setup check. Accordingly, the timing exception has no effect and is ignored. An example would be a path ending at a black-box cell with no timing model.
4. no matching path had start clock – None of the matching paths has a start clock which can occur when the start point is a port with a timing requirement (i.e., there is no `set_input_delay` command for the input port).
5. no matching path was synchronous – All of the paths that match this timing exception are asynchronous (i.e., the starting clock and ending clock for the path are asynchronous). These paths are considered to be false paths and are not timed.
6. none of the paths specified by the constraint exist in the design – After matching objects for every `-from`, `-through`, or `-to` in the command, no matching paths are found in the design. For example, consider the command:

```
set_multicycle_path -from [get_ports A] -to [get_ports Z]
```

If ports A and Z are totally disconnected and there is no path for data to get from A to Z, the timing exception does not apply.

7. the `from` list is incorrect: it contains no clock, primary input, sequential cell, or sequential cell clock pin – If the `-from` portion of a timing exception matches an object, but it is not the correct type of object, it does not apply. An example would be specifying `-from` on the output of a combinational gate.
8. the `through` list is incorrect: it contains no net or hierarchy pin – If the `-through` portion of a timing exception matches an object, but it is not the correct type of object, it does not apply. An example would be specifying a `-through` on a cell pin.
9. the `to` list is incorrect: it contains no clock, primary input, sequential cell, or sequential cell clock pin – If the `-to` portion of a timing exception matches an object,

but it is not the correct type of object, it does not apply. An example would be specifying -to on a pin of a combinational cell.

## MT446

### **@W: <location> Timing constraint (<constraintDescriptor>) (multi-path<integer>) was not applied to the design because <reason>**

The mapper cannot apply the multi-cycle path delay constraint on the specified path because another constraint with a higher priority applies, or because the path does not exist. For example, this message might be displayed if the same path also has a false path constraint or a max delay constraint, both of which have a higher priority than a multicycle constraint.

In the following test case, the constraint file includes the following entries:

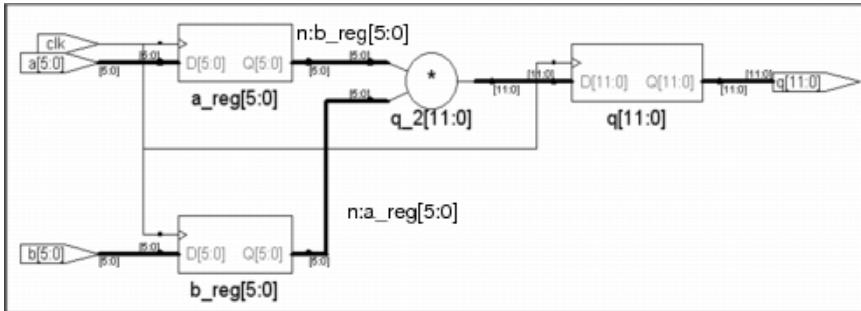
```
Clocks
create_clock -name {clk} p:clk -period 10

Delay Paths
set_multicycle_path -through {n:a_reg[5:0]}
 -through {n:b_reg[5:0]} 2
```

The multicycle path entry in the constraint file defines the path through n:a\_reg[5:0] and n:b\_reg[5:0] as a multi-cycle path that can take two cycles. However, because there is no path through both nets in the RTL, the constraint is not applied, and the warning message is displayed.

```
module mult (a,b,clk,q);
parameter k =5;
input [k:0] a,b;
input clk;
output reg [2*k +1 :0] q;
reg[k:0] a_reg,b_reg;

always @(posedge clk)
begin
 a_reg<= a;
 b_reg<= b;
 q<= a_reg * b_reg;
end
endmodule
```



## Action

First make sure that there is a valid path to apply the multicycle path constraint, and then check that there are no false paths or other higher priority constraints defined for the path.

In the following example, nets *n:a\_reg[5:0]* and *n:b\_reg[5:0]* are individually constrained:

```
Clocks
create_clock -name {clk} p:clk -period 10

Delay Paths
set_multicycle_path -through {n:a_reg[5:0]} 2
set_multicycle_path -through {n:b_reg[5:0]} 2
```

## Reason String

The reason explanation string can be any of the following based on the condition reported. Not all strings are possible for the described timing report messages.

1. higher priority constraint(s) were applied to the paths – Each timing path is checked for matching timing exceptions by comparing the starting point of the path with the -from part of the timing exception command. If two or more timing exceptions match a given path, only one exception is applied to that path. A priority scheme is used to determine which timing exception to apply. The priority depends on the commands and the order in which they are read.

If a timing exception matches one or more paths, but is never the highest priority timing exception for that path, this message is reported. The higher priority timing exception is reported in message [MT621](#).

2. no matching path had hold check – The timing exception affects or disables hold checks (e.g., `set_max_delay`). The timing exception matched one or more paths, but none of the paths ended on a cell pin or port that had a hold check. Accordingly, the timing exception has no effect and is ignored. An example would be a path ending at a black-box cell with no timing model.
3. no matching path had setup check – The timing exception affects or disables setup checks (e.g., `set_min_delay`). The timing exception matched one or more paths, but none of the paths began on a cell pin or port that had a setup check. Accordingly, the timing exception has no effect and is ignored. An example would be a path ending at a black-box cell with no timing model.
4. no matching path had start clock – None of the matching paths has a start clock which can occur when the start point is a port with a timing requirement (i.e., there is no `set_input_delay` command for the input port).
5. no matching path was synchronous – All of the paths that match this timing exception are asynchronous (i.e., the starting clock and ending clock for the path are asynchronous). These paths are considered to be false paths and are not timed.
6. none of the paths specified by the constraint exist in the design – After matching objects for every `-from`, `-through`, or `-to` in the command, no matching paths are found in the design. For example, consider the command:

```
set_multicycle_path -from [get_ports A] -to [get_ports Z]
```

If ports A and Z are totally disconnected and there is no path for data to get from A to Z, the timing exception does not apply.

7. the from list is incorrect: it contains no clock, primary input, sequential cell, or sequential cell clock pin – If the `-from` portion of a timing exception matches an object, but it is not the correct type of object, it does not apply. An example would be specifying `-from` on the output of a combinational gate.
8. the through list is incorrect: it contains no net or hierarchy pin – If the `-through` portion of a timing exception matches an object, but it is not the correct type of object, it does not apply. An example would be specifying a `-through` on a cell pin.

the to list is incorrect: it contains no clock, primary input, sequential cell, or sequential cell clock pin – If the -to portion of a timing exception matches an object, but it is not the correct type of object, it does not apply. An example would be specifying -to on a pin of a combinational cell.

## MT447

### **@W: <location> Timing constraint (<constraintDescriptor>) (false path) was not applied to the design because <reason>**

The mapper cannot apply a false path delay constraint on a given path (i.e., the path specified does not exist or the path can never become a false path).

In the following test case, the constraint file includes the following entries:

```
Clocks
create_clock -name {clk} p:clk -period 10

Delay Paths
set_false_path -from {{i:temp1}} -through {n:temp1}
 -through {n:temp2} -to {{i:q}}
```

The false path entry in the constraint file defines the path from i:temp1 to i:q through n:temp1 and n:temp2 as a false path. In the RTL view, however, nets n:temp1 and n:temp2 are parallel nets (i.e., there is no path through both nets) as indicated by the above warning.

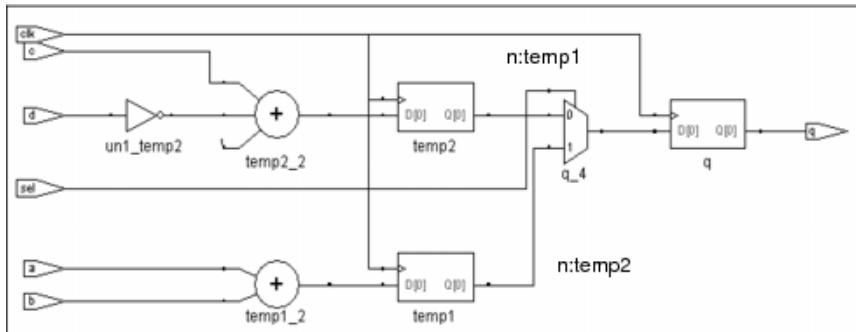
```
module false_test (a,b,c,d,sel,clk,q);
 input a,b,c,d,sel;
 input clk;
 output reg q;
 reg temp1,temp2;

 always@(posedge clk)
 begin
 temp1<= a + b;
 temp2<= c - d;
 if(sel)
```

```

 q<=temp1;
else
 q<=temp2;
end
endmodule

```



## Action

Make sure that a valid path for applying the false path constraint is specified. In the example constraint file entries below, the specified false path is valid.

```

Clocks
create_clock -name {clk} p:clk -period 10

Delay Paths
set_false_path -from {{i:temp1}} -through {n:temp1} -to {{i:q}}

```

## Reason String

The reason explanation string can be any of the following based on the condition reported. Not all strings are possible for the described timing report messages.

1. higher priority constraint(s) were applied to the paths – Each timing path is checked for matching timing exceptions by comparing the starting point of the path with the `-from` part of the timing exception command. If two or more timing exceptions match a given path, only one exception is applied to that path. A priority scheme is used to determine which timing exception to apply. The priority depends on the commands and the order in which they are read.

If a timing exception matches one or more paths, but is never the highest priority timing exception for that path, this message is reported. The higher priority timing exception is reported in message [MT621](#).

2. no matching path had hold check – The timing exception affects or disables hold checks (e.g., `set_min_delay`). The timing exception matched one or more paths, but none of the paths ended on a cell pin or port that had a hold check. Accordingly, the timing exception has no effect and is ignored. An example would be a path ending at a black-box cell with no timing model.
3. no matching path had setup check – The timing exception affects or disables setup checks (e.g., `set_max_delay`). The timing exception matched one or more paths, but none of the paths began on a cell pin or port that had a setup check. Accordingly, the timing exception has no effect and is ignored. An example would be a path ending at a black-box cell with no timing model.
4. no matching path had start clock – None of the matching paths has a start clock which can occur when the start point is a port with a timing requirement (i.e., there is no `set_input_delay` command for the input port).
5. no matching path was synchronous – All of the paths that match this timing exception are asynchronous (i.e., the starting clock and ending clock for the path are asynchronous). These paths are considered to be false paths and are not timed.
6. none of the paths specified by the constraint exist in the design – After matching objects for every `-from`, `-through`, or `-to` in the command, no matching paths are found in the design. For example, consider the command:

```
set_multicycle_path -from [get_ports A] -to [get_ports Z]
```

If ports A and Z are totally disconnected and there is no path for data to get from A to Z, the timing exception does not apply.

7. the from list is incorrect: it contains no clock, primary input, sequential cell, or sequential cell clock pin – If the `-from` portion of a timing exception matches an object, but it is not the correct type of object, it does not apply. An example would be specifying `-from` on the output of a combinational gate.
8. the through list is incorrect: it contains no net or hierarchy pin – If the `-through` portion of a timing exception matches an object, but it is not the correct type of object, it does not apply. An example would be specifying a `-through` on a cell pin.

the to list is incorrect: it contains no clock, primary input, sequential cell, or sequential cell clock pin – If the -to portion of a timing exception matches an object, but it is not the correct type of object, it does not apply. An example would be specifying -to on a pin of a combinational cell.

## MT473

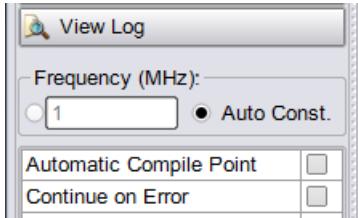
### **@W: Auto Constrain mode is disabled because the design is purely combinational.**

Auto constraining, when enabled, causes the mapper to perform multiple iterations with different clock constraints on the critical paths to achieve the best possible result. This warning occurs when the auto constraint feature is enabled for a design that is purely combinational (auto constraining works on register-to-register and I/O-to-register paths). In the following test case, the design is purely combinational (priority encoder) which causes the mapper to report the above warning.

```
module encoder3(none_on, out2, out1, out0, h, g, f, e, d, c, b, a);
 input h, g, f, e, d, c, b, a;
 output out2, out1, out0;
 output none_on;
 reg [3:0] outvec;
 assign {none_on, out2, out1, out0} = outvec;

 always @((a or b or c or d or e or f or g or h)
 begin
 if (h) outvec = 4'b0111;
 else if (g) outvec = 4'b0110;
 else if (f) outvec = 4'b0101;
 else if (e) outvec = 4'b0100;
 else if (d) outvec = 4'b0011;
 else if (c) outvec = 4'b0010;
 else if (b) outvec = 4'b0001;
 else if (a) outvec = 4'b0000;
 else outvec = 4'b1000;
 end
endmodule
```

Auto constraining is enabled by checking the Maximize radio button in the UI (see below) or by including a -frequency auto option (set\_option -frequency auto) in the project file.



Auto constraining is supported only for some technologies.

Auto constraining is disabled when clock constraints and/or I/O constraints are defined in the constraint file. To use the auto constraining feature, you must remove all clock and I/O constraints from the constraint file and enable the auto constrain feature.

## MT476

**@W: Duplicate clock definition found for clock <clockName>. Remove the redundant clock constraint from the FDC file.**

Two or more clock definitions with the same clock name are specified in an FDC file. For example:

```
create_clock -name myclock -period 10 [get_ports sysclk]
.
.
.
create_clock -name myclock -period 20 [get_ports sysclk]
```

The commands reference the same clock name; specified values may differ.

### Action

Resolve any multiple clock definitions in the FDC file. The file and line number in the warning message refer to the later clock definition. The warning is followed by note BN630, “Clock (*clockName*) previously defined at this location,” which references the first occurrence of the clock definition.

## MT481

**@W: Unable to generate clock <clockName> because no clock was found on its master pin. Make sure that a valid pin or port is defined by the -source option for the create\_generated\_clock command.**

---

**Note:** The above message can be incorrectly reported when the name of a derived clock is defined by its associated create\_generated\_clock constraint using the -name {clkName} object format. Warnings resulting from these derived clocks can be ignored; derived clocks are not used and subsequently removed by the mapper.

---

## MT518

**@E: Could not open file <fileName> for writing timing correlation**

The synthesis software cannot write to the specified file (*fileName*) for timing correlation. This can occur when:

1. The file already exists, but the permissions set for this file do not allow the synthesis software to overwrite it.
2. The directory for the file exists, but the permissions set for this file do not allow the synthesis software to write to it.
3. The directory for this file does not exist, because the synthesis software could not create it.

### Action

For the conditions mentioned above, respectively, do the following:

1. Remove the specified file or change permissions for the file so that the synthesis software can write to it.

2. Remove the directory for the specified file or change permissions for the file so that the synthesis software can write to it.
3. Check that the specified file permissions are set correctly for its parent directory.

## MT519

### **@E: Could not open file <fileName> for write**

The synthesis software cannot write to the specified file (*fileName*). This can occur when:

1. The file already exists, but the permissions set for this file does not allow the synthesis software to overwrite it.
2. The directory for the file exists, but the permissions set for this file does not allow the synthesis software to write to it.
3. The directory for this file does not exist, because the synthesis software could not create it.

### Action

For the conditions mentioned above, respectively, do the following:

1. Remove the specified file or change permissions for the file so that the synthesis software can write to it.
2. Remove the directory for the specified file or change permissions for the file so that the synthesis software can write to it.
3. Check that the specified file permissions are set correctly for its parent directory.

## MT522

### **@W: Failed to find starting point of critical path to <endingPoint>**

During report timing, the starting point for the timing path reported at the end point could not be identified. This warning can be the result of a primitive in the path that does not have a valid timing model.

#### Action

Expand the timing path back from the end point and verify that all elements in the path have valid timing models.

## MT523

### **@W: Further failures to find starting point of critical path will not be reported**

The number of starting points on critical paths that cannot be located exceeds a predefined number that is considered to be excessive.

#### Action

See warning [MT522](#) and make sure that the critical path starting points are defined for the specified end points.

# MT529

**@W: Found inferred clock <clkName> which controls <numClkPins> sequential elements including <instanceName>. This clock has no specified timing constraint which may prevent conversion of gated or generated clocks and may adversely impact design performance.**

A node that controls the clock pin of some sequential cells, but that does not have a user-defined timing constraint, was found in the design. Without a timing constraint, this clock (*clkName*) is inferred and assigned the global frequency defined in the project. The assigned frequency is displayed in the clock summary table in the log file (just before this warning message). If the frequency assigned is too fast, area may be wasted. If the frequency is too slow, the final circuit may not realize the desired performance. The design in question has one or both of the fix-gated clocks or fix-generated clocks options enabled. For these features to work properly on the clock tree, a user clock constraint must be defined at the proper point in the clock tree.

## Action

The corrective action is to define a user clock constraint at the proper point in the clock tree.

The number of sequential elements listed in the warning message (*numClkPins*) is an estimate of how many sequential instances that this node (*clkName*) controls. If there are multiple inferred clock warnings, use this number to prioritize the most important clocks to address first. The warning message lists one example sequential element that is controlled by the indicated node. Use this information to identify the clock tree as follows:

1. Open the RTL and Technology views of the design in the HDL Analyst.
2. Highlight the indicated instance (*instanceName*) in the log file.
3. Right click and select Filter in Analyst from the menu. The RTL view is filtered to show just the one sequential instance.
4. Right click on the connector of the clock pin and select Expand to Register/Ports from the menu. The resulting circuit represents the complete clock tree for *clkName*.

The following procedures are guidelines for analyzing the clock tree and determining where the source clock is to be defined. Two procedures are given; use one procedure if the clock constraint is not already defined and use the other procedure if a clock constraint is defined.

Examine the clock tree circuit to determine the actual source of the clock. This point is usually an input port, a register, or a PLL. Depending on if this point already has a clock constraint, follow the appropriate guidelines below to try and resolve this issue.

## No Clock Constraint

If the point does not already have a clock constraint:

- If the point is not a register or PLL, add the clock constraint and rerun
- If the point is a register:
  - if the register is the clock source, add the clock constraint and rerun
  - if the register is the output of a clock divider, find the source clock of the divider circuit, add a generated clock constraint, and rerun
- If the point is a PLL:
  - check that the PLL model is included; if not, add the model and rerun
  - check that the input clock to the PLL is defined; if not, define the clock and rerun
  - if the PLL model is not available or the input is already defined and is not propagating through the PLL model, define the clock on the output of the PLL and rerun

## Clock Constraint Present

If the point already has a clock constraint, analyze the components on the circuit that lie between the source clock and the clock pin of the example sequential instance to determine why the clock is not being propagated. The Synplify software can only propagate clocks forward through unate combinational cells; a non-unate cell, a black box, or another sequential cell all can prevent clock propagation.

- If there is a non-unate cell such as a MUX or an XOR cell between the source clock and the clock pin, add the clock constraint in an appropriate place downstream from the non-unate cell and rerun.

- If there is a black box or a sequential cell between the source clock and the clock pin, add the clock constraint in an appropriate place downstream from the black box or sequential cell and rerun. If the sequential cell is part of a clock divider, be sure to add a generated-clock constraint.

## MT530

**@W: Found inferred clock <clkName> which controls <numClkPins> sequential elements including <instanceName>. This clock has no specified timing constraint which may adversely impact design performance.**

A node that controls the clock pin of some sequential cells, but that does not have a user-defined timing constraint, was found in the design. Without a timing constraint, this clock (*clkName*) is inferred and assigned the global frequency defined in the project. The assigned frequency is displayed in the clock summary table in the log file (just before this warning message). If the frequency assigned is too fast, area may be wasted. If the frequency is too slow, the final circuit may not realize the desired performance.

### Action

The corrective action is to define a user clock constraint at the proper point in the clock tree.

The number of sequential elements listed in the warning message (*numClkPins*) is an estimate of how many sequential instances that this node (*clkName*) controls. If there are multiple inferred clock warnings, use this number to prioritize the most important clocks to address first. The warning message lists one example sequential element that is controlled by the indicated node. Use this information to identify the clock tree as follows:

1. Open the RTL and Technology views of the design in the HDL Analyst.
2. Highlight the indicated instance (*instanceName*) in the log file.
3. Right click and select Filter in Analyst from the menu. The RTL view is filtered to show just the one sequential instance.

4. Right click on the connector of the clock pin and select Expand to Register/Ports from the menu. The resulting circuit represents the complete clock tree for *clkName*.

The following procedures are guidelines for analyzing the clock tree and determining where the source clock is to be defined. Two procedures are given; use one procedure if the clock constraint is not already defined and use the other procedure if a clock constraint is defined.

Examine the clock tree circuit to determine the actual source of the clock. This point is usually an input port, a register, or a PLL. Depending on if this point already has a clock constraint, follow the appropriate guidelines below to try and resolve this issue.

## No Clock Constraint

If the point does not already have a clock constraint:

- If the point is not a register or PLL, add the clock constraint and rerun
- If the point is a register:
  - if the register is the clock source, add the clock constraint and rerun
  - if the register is the output of a clock divider, find the source clock of the divider circuit, add a generated clock constraint, and rerun
- If the point is a PLL:
  - check that the PLL model is included; if not, add the model and rerun
  - check that the input clock to the PLL is defined; if not, define the clock and rerun
  - if the PLL model is not available or the input is already defined and is not propagating through the PLL model, define the clock on the output of the PLL and rerun

## Clock Constraint Present

If the point already has a clock constraint, analyze the components on the circuit that lie between the source clock and the clock pin of the example sequential instance to determine why the clock is not being propagated. The Synplify software can only propagate clocks forward through unate combinational cells.

- If there is a non-unate cell such as a MUX or an XOR cell between the source clock and the clock pin, add the clock constraint in an appropriate place downstream from the non-unate cell and rerun.
- If there is a black box or a sequential cell between the source clock and the clock pin, add the clock constraint in an appropriate place downstream from the black box or sequential cell and rerun. If the sequential cell is part of a clock divider, be sure to add a generated clock constraint.

## MT531

**@W: Found signal identified as System clock which controls <numClkPins> sequential elements including <instanceName>. Using this clock, which has no specified timing constraint, can prevent conversion of gated or generated clocks and can adversely impact design performance.**

A System clock did not have a user-defined timing constraint and was assigned the global frequency defined in the project. This assigned frequency is displayed in the clock summary table in the log file (just before this warning message). If the frequency assigned is too fast, area may be wasted. If the frequency is too slow, the final circuit may not realize the desired performance. The design in question has one or both of the fix-gated clocks or fix-generated clocks options enabled. For these features to work properly on the clock tree, a user clock constraint must be defined at the proper point in the clock tree

### Action

The corrective action is to define a user clock constraint at the proper point in the clock tree.

The number of sequential elements listed in the warning message (*numClkPins*) is an estimate of how many sequential instances that the System clock controls. If there are multiple System clock warnings, use this number to prioritize the most important clocks to address first. The warning message lists one example sequential element that is controlled by the System clock. Use this information to identify the clock tree as follows:

1. Open the RTL and Technology views of the design in the HDL Analyst.
2. Highlight the indicated instance (*instanceName*) in the log file.
3. Right click and select Filter in Analyst from the menu. The RTL view is filtered to show just the one sequential instance.
4. Right click on the connector of the clock pin and select Expand to Register/Ports from the menu. The resulting circuit represents the complete clock tree for System clock.

The following procedures are guidelines for analyzing the clock tree and determining where the source clock is to be defined. Two procedures are given; use one procedure if the clock constraint is not already defined and use the other procedure if a clock constraint is defined.

Examine the clock tree circuit to determine the actual source of the clock. This point is usually an input port, a register, or a PLL. Depending on if this point already has a clock constraint, follow the appropriate guidelines below to try and resolve this issue.

## No Clock Constraint

If the point does not already have a clock constraint:

- If the point is not a register or PLL, add the clock constraint and rerun
- If the point is a register:
  - if the register is the clock source, add the clock constraint and rerun
  - if the register is the output of a clock divider, find the source clock of the divider circuit, add a generated clock constraint, and rerun
- If the point is a PLL:
  - check that the PLL model is included; if not, add the model and rerun
  - check that the input clock to the PLL is defined; if not, define the clock and rerun
  - if the PLL model is not available or the input is already defined and is not propagating through the PLL model, define the clock on the output of the PLL and rerun

## Clock Constraint Present

If the point already has a clock constraint, analyze the components on the circuit that lie between the source clock and the clock pin of the example sequential instance to determine why the clock is not being propagated. The Synplify software can only propagate clocks forward through unate combinational cells; a non-unate cell, a black box, or another sequential cell all can prevent clock propagation.

- If there is a non-unate cell such as a MUX or an XOR cell between the source clock and the clock pin, add the clock constraint in an appropriate place downstream from the non-unate cell and rerun.
- If there is a black box or a sequential cell between the source clock and the clock pin, add the clock constraint in an appropriate place downstream from the black box or sequential cell and rerun. If the sequential cell is part of a clock divider, be sure to add a generated-clock constraint.

## MT532

**@W: Found signal identified as System clock which controls <numClkPins> sequential elements including <instanceName>. Using this clock, which has no specified timing constraint, can adversely impact design performance.**

A System clock did not have a user-defined timing constraint and was assigned the global frequency defined in the project. This assigned frequency is displayed in the clock summary table in the log file (just before this warning message). If the frequency assigned is too fast, area may be wasted. If the frequency is too slow, the final circuit may not realize the desired performance.

### Action

The corrective action is to define a user clock constraint at the proper point in the clock tree.

The number of sequential elements listed in the warning message (*numClkPins*) is an estimate of how many sequential instances that the System clock controls. If there are multiple System clock warnings, use this number to prioritize the most important clocks to address first. The warning message lists one example sequential element that is controlled by the System clock. Use this information to identify the clock tree as follows:

1. Open the RTL and Technology views of the design in the HDL Analyst.
2. Highlight the indicated instance (*instanceName*) in the log file.
3. Right click and select Filter in Analyst from the menu. The RTL view is filtered to show just the one sequential instance.
4. Right click on the connector of the clock pin and select Expand to Register/Ports from the menu. The resulting circuit represents the complete clock tree for System clock.

The following procedures are guidelines for analyzing the clock tree and determining where the source clock is to be defined. Two procedures are given; use one procedure if the clock constraint is not already defined and use the other procedure if a clock constraint is defined.

Examine the clock tree circuit to determine the actual source of the clock. This point is usually an input port, a register, or a PLL. Depending on if this point already has a clock constraint, follow the appropriate guidelines below to try and resolve this issue.

## No Clock Constraint

If the point does not already have a clock constraint:

- If the point is not a register or PLL, add the clock constraint and rerun
- If the point is a register:
  - if the register is the clock source, add the clock constraint and rerun
  - if the register is the output of a clock divider, find the source clock of the divider circuit, add a generated clock constraint, and rerun
- If the point is a PLL:
  - check that the PLL model is included; if not, add the model and rerun
  - check that the input clock to the PLL is defined; if not, define the clock and rerun

- if the PLL model is not available or the input is already defined and is not propagating through the PLL model, define the clock on the output of the PLL and rerun

## Clock Constraint Present

If the point already has a clock constraint, analyze the components on the circuit that lie between the source clock and the clock pin of the example sequential instance to determine why the clock is not being propagated. The Synplify software can only propagate clocks forward through unate combinational cells; a non-unate cell, a black box, or another sequential cell all can prevent clock propagation.

- If there is a non-unate cell such as a MUX or an XOR cell between the source clock and the clock pin, add the clock constraint in an appropriate place downstream from the non-unate cell and rerun.
- If there is a black box or a sequential cell between the source clock and the clock pin, add the clock constraint in an appropriate place downstream from the black box or sequential cell and rerun. If the sequential cell is part of a clock divider, be sure to add a generated-clock constraint.

## MT536

### **@E: Found constraint file with both legacy-style and FPGA timing constraints.**

The FDC constraint file in the project contains both legacy-style and FPGA timing constraints (the mixing of timing constraint types in the same file is not supported).

#### Action

Make sure all timing constraints in a constraint file are of the same type. Legacy-style timing constraints can be replaced with their FPGA equivalents using the sdc2fdc Tcl shell command. For more information on compatible timing constraints and constraint-file conversion, see *sdc2fdc* in online help or in Chapter 5, *Specifying Constraints*, of the *User Guide*.

## MT548

**@W: Source for clock <clockName> not found in netlist. Run the constraint checker to verify if constraints are applied correctly.**

The clock object identified by the `create_clock` (or `create_generated_clock`) command in the constraint file could not be found in the netlist.

```
Clocks
create_clock -name {temp} n:temp1 -period {10}
```

Consider the following example:

```
module warningtest_1 (clk1, clk2, a, b, c);
 input clk1, clk2, a, b;
 output c;
 reg c; wire temp;
 assign temp = clk1 & clk2;

 always @ (posedge temp)
 begin
 c <= a & b;
 end
endmodule
```

### Action

Check if the net exists. Open the RTL view, and search for the net by right-clicking and selecting Find->Nets. If there is no net with the specified name, delete the constraint. If there is a net, and the warning was triggered because of a discrepancy or typo in the net name, correct the object name in the constraint.

In the above example, net `temp1` does not exist in the design. Replace `temp1` with `temp` in the constraint to eliminate the warning:

```
create_clock -name {temp} {n:temp} -period {10}
```

## MT551

### **@W: Master clock for generated clock <*clockName*> not found.**

The clock defined by the `create_generated_clock` command could not be generated because the master-clock source for the named generated clock could not be found.

#### Action

Check the `create_generated_clock` entry in the constraint file and make sure that the clock name referenced in the `-master_clock` option is a valid clock. Note that the `-master_clock` entry must be a clock alias.

## MT552

### **@W: Unable to resolve master clock for generated clock <*clockName*>**

The above warning occurs when a user-defined `create_generated_clock` constraint includes clock objects that cannot be identified or resolved.

#### Action

Check the `create_generated_clock` command entries in the constraint file and make sure that the correct parameters are applied to the objects in the constraint.

## MT571

**@W: Clock <clockName> has negative slack of <integer>ns and it is unlikely to meet timing goals; consider relaxing the constraint.**

The named clock shows a negative slack value which, if left uncorrected, can result in timing violations. The clock net and its negative slack value are reported in the warning message.

### Action

To correct a negative timing constraint, consider:

- Relaxing the clock constraint
- Setting a max-delay or false-path constraint on the path

## MT582

**@N: Estimated period and frequency not reported for given clock unless the clock has at least one timing path which is not a false or a max delay path and that does not have excessive slack**

Estimated period is an estimate of the smallest period that the clock can have without timing violations, and estimated frequency is the corresponding frequency.

To determine the estimated period for a clock, the period for each timing path that starts or ends on that clock is examined. If there is a max delay timing exception on that path, the slack for that path does not depend on the clock frequency and is ignored when making the estimate. Also, for any false path or path between asynchronous clocks, the path is similarly ignored.

A path is considered to have excessive slack when its slack is more than one effective cycle because of a negative set\_input\_delay on an input port or a large set\_output\_delay on an output port. This path has a large slack irrespective of the clock period and is ignored when making the period estimate.

If there are no timing paths for the clock, or if the only paths are ignored for any of the above reasons, the estimated period and frequency are reported as undefined (NA).

#### Action

Informative message; no action required.

## MT596

**@W: Mux <name> is driven by clocks on all data inputs. Outputs may have multiple incompatible clock records. Please use syn\_clock\_gmux\_proxy\_attribute or set\_case\_analysis to resolve this. Please check help for more details.**

For the specified circuit, the tool propagates multiple clocks through the BUFGMUX during timing analysis. However, timing analysis does not automatically treat the clock signals as mutually exclusive on the output of the mux, so paths between the clocks after the mux are timed, even though both clocks are not actually propagating simultaneously through the mux in the real circuit. This means that false paths are timed after the mux unless they are explicitly defined as false paths or asynchronous to each other in the FDC file.

#### Action

There are two ways to address this issue; pick the method best suited to your design needs:

- Use the `syn_clock_gmux_proxy` attribute to automatically generate proxy clocks at the outputs of the BUFGMUX, so that the output clocks correspond to the input clocks. You can set this as a global attribute or on an individual BUFGMUX. See the attribute syntax for details.

The disadvantage to this approach is that it generates more clocks in the design.

- Enable `set_case_analysis` on the select input pin for the BUFGMUX, to specify which clock is propagated through the mux.

The disadvantage to this approach is that it cannot be set globally; the BUFGMUX must be specified.

## MT620

### **@N: Non-applying exception: <exceptionName>**

The named timing exception was not applied as it is in contention with another timing exception (see [MT621](#))

#### Action

Informative message; no action required.

## MT621

### **@N: Applying exception: <exceptionName>**

The named timing exception is being applied. See [MT620](#) for any timing exceptions that are in contention and not being applied.

#### Action

Informative message; no action required.

## MT623

**@N: Timing exception did not apply because a higher priority timing exceptions applied on every matching timing path:**

A timing exception matches one or more paths, but not the highest priority timing exception for that path. The higher priority timing exception is reported in message [MT621](#).



## CHAPTER 30

# UI Messages 100 – 239

---

## UI109

### **@E: Implementation not found: <*implementation*>**

The project file does not have an implementation.

#### Action

Verify that the project file is open and that it contains an implementation.

## UI110

### **@E: Device family not recognized: <*technology*>**

The target technology is not valid.

#### Action

Use the implementation options dialog box or **set\_option -technology** *parameter* to set a valid technology.

## UI111

### **@E: No input file specified**

An SRP (partitioned netlist) file has not been specified.

#### Action

Set an SRP file and repeat action.

## UI113

### **@E: File not found: <file>**

The specified SRP input file for SLP generation was not found.

#### Action

Contact Synopsys support by logging in to the Synopsys website.

## UI116

### **@E: Hierarchical projects are not supported**

The Synopsys tool being used does not support hierarchical projects.

#### Action

If hierarchical project support is required, make sure you are using the appropriate tool.

## UI120

### **@E: Subproject pre-link netlist not found: <link>**

A pre-link netlist in a subproject was not found.

#### Action

Contact Synopsys support by logging in to the Synopsys website.

## UI131

### **@E: Unable to open power configuration file: <file>**

The named power analysis file cannot be found.

#### Action

Contact Synopsys support by logging in to the Synopsys website.

## UI140

### **@E: All instances must be assigned before mapping.**

All logic elements must be assigned prior to mapping the project.

#### Action

Assign all instances and repeat mapping.

## UI144

**@E: Compiled board netlist not found: <netlist>. You must first compile your board file.**

This error occurs when a board file has not been compiled.

### Action

Compile the board file and rerun RTL preparation.

## UI148

**@E: Invalid filename ignored: <file>**

The specified file cannot be used in a **run** command.

### Action

Verify that the file name is correct and repeat the action.

## UI149

**@E: Invalid filename not added to project: <file>**

The named file could not be added to the current project.

### Action

Correct the name of the file and repeat the action.

## UI155

### **@E: File: "<fileName>" not found in project: "<project>"**

The named file cannot be found in the current project.

#### Action

Verify that the path to the file is correct.

## UI161

### **@E: Cannot set "< projectName >"**

The command `set_option projectName` cannot be used to name the project.

#### Action

Name the project file when you create it.

## UI168

### **@E: Project not found: <project>**

The named project file cannot be found.

#### Action

Verify that the path to the project file is correct.

## UI169

### **@E: No opened projects**

A project file must be open before the action is taken.

#### Action

Open the project file and repeat the action.

## UI173

### **@E: There are no projects loaded.**

A project file must be open before the action is taken.

#### Action

Open the project file and repeat the action.

## UI175

### **@E: No active implementation**

An active implementation is required before the action is taken.

#### Action

Create an active implementation and repeat the action.

## UI176

### **@E: No active project.**

A project file must be open before the action is taken.

#### Action

Open the project file and repeat the action.

## UI177

### **@E: Implementation not found: '<implementation>'**

The named implementation cannot be located.

#### Action

Check the implementation name and path.

## UI186

### **@E: Invalid part "<part>" for "<family>" family**

The named part specified in the project file is invalid.

#### Action

Verify the part name.

## UI191

### **@E: Expecting: "<argument> = 0 | 1"**

This error occurs when there is a value other than **0** or **1** specified for the named argument in the project file or a Tcl script.

#### Action

Verify the value of the argument.

## UI192

### **@E: Expecting: "<argument> = "**

This error occurs when there is an unexpected value specified for the named argument in the project file or a Tcl script.

#### Action

Verify the value of the argument.

## UI194

### **@E: Could not start job. Is the current directory writeable ?**

The current directory is not write-enabled.

#### Action

Verify that the directory is write-enabled and that there is sufficient disk space.

## UI199

### **@E: Maximum number of users exceeded for feature: <feature>**

A license is currently unavailable for the named feature.

#### Action

Wait for a license to become available, then repeat the action.

## UI218

### **@E: Failed to load/create incremental instrumentation: <instrumentation>**

The software was unable to create an incremental instrumentation.

#### Action

Rerun place and route. If error recurs, contact Synopsys support by logging into the Synopsys website.

## UI219

### **@E: Design Planning requires an active implementation**

An active implementation is required to use design planning.

#### Action

To use design planning, open the project and select an implementation.

## UI220

### **@E: Design Planning is not supported for family <family>**

Design planning is not supported for the specified family.

#### Action

Use a device family that supports design planning.

## UI226

### **@E: Failed to checkout license required for job: <jobName>**

This is general error stating that the job shown in message required an additional license feature that was not available.

#### Action

Confirm license server environment variable is pointing to a license server that has available licenses of the required feature type.

## UI227

### **@E: Failed to initialize job: <jobName>**

This is general error message when a job fails to start for a number of reasons. For example:

No HDL files found to compile  
Failed to Initialize job: compiler

### Action

Depending upon the issue, this message will be preceded by another message that will give you a more detailed description on how to address the problem.

## UI228

### **@E: job not found to run: <job>**

An internal error has occurred.

### Action

Open a support ticket by visiting the Synopsys website and clicking the Support link at the top of the page.

## UI229

### **@E: Output format has to be .znl.**

The only output format allowed for the software version you are using is .znl. You are trying to open a generic project for which the output format is not .znl.

### Action

Check the software version you are using and confirm whether the software version supports the output format .znl. Open only files in .znl format while using this version of the software.

## UI230

### **@E: Output format cannot be znl.**

You are using a software version that does not support the .znl output format.

#### Action

Check the software version you are using and confirm whether the software version supports the ouput format .znl. Use a generic project output instead of .znl.

## UI233

### **@E: Unified compile flow is not supported in Windows**

The ProtoCompiler tool does not support Unified Compiler Flow (UCF) on Windows. This feature is supported only on Linux.

#### Action

Use the ProtoCompiler tool on Linux.

## UI234

### **@E: Unified compile database generation did not complete, successfully. See <logFile> for details**

This error occurs when the unified compile database generation fails.

### Action

The log file that is part of the message contains the error details and its source reference. Verify the log file and correct the source.

## UI235

### **@E: UTF file <fileName> does not exist**

The UTF file specified to launch uc command does not exist.

### Action

Verify if the name and/or path of the specified UTF file is correct.

## UI236

### **@E: Unable to create directory <directoryName>**

This error occurs when the tool is unable to create a directory while running the launch uc command.

### Action

Check if the current working directory has write permission.

## UI237

### **@E: UTF file <fileName> does not contain VCS execution command**

This error occurs when the UTF file does not contain the VCS execution command.

#### Action

The UTF file must contain the UTF command specifying the VCS execution script. For more details, see “Running the Unified Compiler” in the *HAPS Prototyping User Guide*.

## UI238

### **@E: \nFailed to set environment for VCS invocation**

The ProtoCompiler tool failed to set the environment for VCS invocation because the system resources are full.

#### Action

Reboot the system. If the error persists, please contact Synopsys support.

## UI239

### **@E: \nMultiple UTF files cannot be included in the project**

Only one UTF file can be included in a project.

**Action**

Make sure you have included only one UTF file with the added UTF commands.



## CHAPTER 31

# Z Messages 100 – 372

---

## Z100

### Compiling code

A VHDL assertion failed in the compiler. This message can be a note, warning, or error depending on how you set the severity in the code.

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
port (
 a : in integer;
 b : out integer);
end entity;

architecture behv of test is
begin
assert false report "Compiling code..." severity error;
b <= a;
end behv;
```

## Action

You can set the severity for messages generated when a VHDL assertion fails in the compiler.

# Z120

### **@E: Fix above errors to continue**

Indicates there are preceding error messages that need to be addressed before continuing.

## Action

Fix the listed errors before proceeding.

# Z198

### **@W: Unbound component <sub></sub> of instance <IO>**

For mixed-language and IP-based designs, source components must match target components. Whenever components do not match, the components are not linked and an Unbound component warning message is generated.

In the following example, the VHDL top-level entity (top) instantiates a Verilog module sub. The component sub is defined with 5-bit ports in VHDL; however, the sub module in Verilog specifies 4-bit ports. Accordingly, sub is not linked to the top-level module. A link summary compiler report file is also generated with a link from the log file.

#### **top.vhd**

```
library ieee;
use ieee.std_logic_1164.all;
```

```

entity top is
 port (in1 : in std_logic_vector(4 downto 0);
 in2 : in std_logic_vector(4 downto 0);
 dout : out std_logic_vector(4 downto 0));
end entity;

architecture behv of top is
component sub
 port (in1 : in std_logic_vector(4 downto 0);
 in2 : in std_logic_vector(4 downto 0);
 dout : out std_logic_vector(4 downto 0));
end component;
begin
 I0 : sub port map (in1 => in1, in2 => in2, dout => dout);
end behv;

```

### **sub.v**

```

module sub(input [3:0] in1, input [3:0] in2, output [3:0] dout);
 assign dout = in1 & in2;
endmodule

```

### Action

Make sure that the source and target port widths match. The following shows the warning entry in the log file and the link to the link summary report (`top_compiler.linkerlog`) shown below.

#### **Log File:**

```

W:Z198 : top.vhd(19) | Unbound component sub of instance I0
=====
For a summary of linker messages for components that did not bind,
please see log file: Linked File: top_compiler.linkerlog
=====
```

#### **top\_compiler.linkerlog**

```

Link Summary for source view work.sub.syn_black_box of instance I0:
=====
Target view link candidate 1: work.sub.verilog
=====
Interface Mismatch between View work.sub.syn_black_box and
View work.sub.verilog

Details:
=====
The following bit ports in the Source View
```

```
work.sub.syn_black_box do NOT exist in the Target View
work.sub.verilog
=====
 Bit Port in1[4]
 Bit Port in2[4]
 Bit Port dout[4]
```

## Z315

**@E: Netlist Editor: Error in <fileName> file, line <lineNumber>: <tclMessage>**

An error was encountered on the specified line in a user Tcl file. In the message, *tclMessage* describes the nature of the error.

## Z325

**@E: Multiple views found for cell <cellName>.**

The netlist editor encountered multiple views for a cell.

### Action

Make sure that the cell references only a single view. This error can occur in VHDL when more than one architecture is defined for an entity.

## Z327

**@E: Multiple views of the cell <cellName> in the library <libName>.**

The netlist editor found multiple views of the named cell in the specified library.

### Action

Make sure that the cell in the library references only a single view. This error can occur in VHDL when more than one architecture is defined for an entity.

## Z328

### **@E: The library <libName> not found in the design.**

The referenced library could not be found in the design.

### Action

Make sure that the library name is correct and that the appropriate design is loaded.

The user-assigned I/O standard does not match any of the recognized HAPS I/O standards.

Check the IOSTANDARD property value on the named port.

## Z363

### **@E: Instance <instance> constrained to <value> cannot fit as the required area exceeds available resources. Please modify the respective constraint.**

A constraint is set to a value greater than the number of available resources.

### Action

Modify the constraint and rerun the design.

## Z386

**@E: trainOnReset mismatch on FPGAs. (trainOnReset=%d, hasHSTDm=%d).**

This error occurs if two different training methods — old reset method (reset tree) and new reset method (supervisor controlled) — are used simultaneously during HSTDm training on different boards.