Synphony C Compiler
Error Messages

Version K-2015.06, June 2015

SYNOPSYS®

Copyright Notice and Proprietary Information

Destination Control Statement

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

Disclaimer

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

Trademarks

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

Third-Party Links

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

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

Contents

COM Error Messages

COM-1001 (warning) Unroll resulted in code with %d ops. This may lead to long run times and/or out-of-memory errors.

DESCRIPTION

In some designs unrolling a loop using the unroll pragma can result in large number of operations. This can lead to long run times and in some cases out-of-memory errors.

WHAT NEXT

Use TCABs to encapsulate the operations for reducing the run time.

COM-1022 (warning) Label(s) "%s" not propagated. Dropping the constraint between (%s, %s) with relative/absolute schedule time %d.

DESCRIPTION

You can use labels (i.e. **$SCC label mylabel**) to,

• apply no-dependence constraints on memory

• control scheduling of different operations using wait constraints.

The Synphony C Compiler tool can drop labels if the source code which has label gets optimized.

For example, the following code uses labels to apply wait constraints to co-schedule two output streams. Some labels are not present in user code and constraints associated with them will be dropped.

```
ppa() {
     // user code
     for (i=0; i < 100; i++) {
       int x_1 = in_1.get();
       int x_2 = in_2.get();
       /* $SCC label lb_1*/ out_1.put((x_1*x_2));
       /* $SCC label lb_2*/ out_2.put((x_1+x_2));
     }
     // user code
   }
```

```
   // Constraints in user tcl command file
   set_params -wait "lb_1:lb_2:0"
   set_params -wait "lb_1:lb_3:0"


//Generated warning
Warning: Label(s) "lb_3" not propagated. Dropping the constraint between
(lb_1, lb_3) with relative/absolute schedule time 0.
```

WHAT NEXT

Remove wait constraints applied on unused label from your Tcl file or move the label to code that is synthesized but not optimized.

COM-1031 (error) Accesses to globals, statics, localized or externalized variables inside a channel's execute method are not supported.

DESCRIPTION

A channel is synthesized into free running hardware block. The channel interacts with other blocks only through stream or stable scalar interfaces. This ensures proper synchronization between writes and reads.

WHAT NEXT

Ensure that your channel's **execute** function does not access any globals, statics, localized or externalized variables. Try to make them local to channel or pass the values to the channel through streams.

COM-1034 (warning) Multiple label_type comments '%s' are specified for the type of the variable. Using only the first label_type comment '%s' to name the type.

DESCRIPTION

Template classes can be used for channels. To specify the template arguments to be used for synthesis, a channel declaration is labeled using **$SCC label_type channel_name**, and then **channel_name** is specified as the target for synthesis. Multiple labels on a channel declaration are ignored. The Synphony C Compiler tool only considers the first label as valid.

In the following example, the label my_chnl_object is created and used as the target for synthesis (for example, **set_params -target my_chnl_object**). Because the label is attached to a declaration with type **my_scc_channel<32>**, the channel is synthesized with template parameter n bound to 32.

```
template <unsigned int n>
class my_scc_channel: public scc_channel_if {
public:
  scc_port<scc_instream_if<unsigned_char> > datain;
  scc_port<scc_outstream_if<unsigned_int> > dataout;
  --
};

void my_ppa() {
  /* $SCC label_type my_chnl_object */
  my_scc_channel<32> my_channel;
}
```

WHAT NEXT

Remove redundant labels applied on channel declarations.

COM-1036 (error) Incorrect usage of $SCC comment %s.

DESCRIPTION

This error message is generated if there is a mistake in the application of a label identifier as per the correct specification. Note that labels are specified in comments. If the code segment corresponding to a label is commented, then the Synphony C Compiler tool cannot identify the label with the code segment and generates this error message.

WHAT NEXT

1. If you have commented the code segment along with the label, delete "$SCC" in the comment.

2. To create a label for an expression, statement, or function declaration, use a comment in the following form: /*$SCC label label_name*/ The label applies to the operator following the comment. For example, x = a + b /*$SCC label e1*/ *c - d; or x = a + /*$SCC label e1*/ b*c - d; In either of these examples, the label e1 applies to the operation b*c that immediately follows the comment.

3. To label a region of code, use the following syntax: /*$SCC label_begin label_name*/ code_region /*$SCC label_end label_name*/ For example, x = a + /*$SCC label_begin e2*/ b*c d /*$SCC label_end e2*/; The e2 label applies to both the operations (b*c) and ((b*c) d) that are enclosed by label_begin/label_end label comments.

4. For template channels, you need to provide a $SCC label_type to the channel object. For example, /* $SCC label_type my_chnl_a */ my_channel<unsigned_int> packer;

COM-1037 (warning) 'init' method for the channel is not defined in the appfiles. Not synthesizing any initialization for the channel.

DESCRIPTION

The Synphony C Compiler channel class scc_channel supports pre-defined **init** method for initialization of channel and **execute** method for describing the functionality of channel. For synthesizing the initialization of channel user should include the definition of **init** method in appfiles.

The following shows the default channel structure:

```
class my_scc_channel: public scc_channel_if {
public:
  // channel ports declaration
  scc_port<int> datain;
  scc_port<int> dataout;

  void init() {
    // Initialization of channel variables
  }
  void execute() {
    // functionality of channel
  }
};
```

For example, if you do not include **init** method in **my_scc_channel** definition, the initialization of channel will not be part of synthesized RTL.

WHAT NEXT

Define the **init** method in your channel and is included in appfiles.

COM-1038 (error) Fields of struct/class or an element of an array cannot be bound to the stable ports of a channel.

DESCRIPTION

A channel in the Synphony C Compiler tool can have stable inputs, stream inputs, and stream outputs. The Synphony C Compiler tool does not support binding a field of a struct/class or an element of an array to a stable input port of a channel. The following example binds the field of a struct type to a channel, which is not supported.

```
        void my_design (struct A &var) {
          scc_stream <uint8> ch_in;
          scc_stream_with_get_trigger<uint16> ch_out;
          my_channel ch_obj(ch_in, ch_out, var.mode); //
 binding a field of struct to channel
```

```
          // var.mode is bound to scc_in<uint1> port of the channel
          //
 declaration of scc_in makes the port stable inside the channel

          ch_out(ch_obj);
          ...
        }
```

WHAT NEXT

Modify your code to avoid binding a field of a struct/class or an element of an array to stable input ports of the channel. The following example shows an alternative coding style to avoid binding a field of a struct type to a channel:

```
        // testbench code
        mode = var.mode;
        my_design(var, mode);

        // synthesizable code
        void my_design (struct A &var, uint1 &mode) {
          scc_stream <uint8> ch_in;
          scc_stream_with_get_trigger<uint16> ch_out;
          my_channel ch_obj(ch_in, ch_out, mode);
          ch_out(ch_obj);
          ...
        }
```

COM-1039 (error) Some of the ports of the channel '%s' are not bound properly. Binding of channel ports should be done inside the PPA or in the constructor for global objects.

DESCRIPTION

All stream ports of a channel must be bound to an scc_stream, and the binding must be visible in PPA scope. For a channel declared in procedure scope, the binding must be in the procedure itself.

For a globally declared channel, the binding can also be in the constructor. For example, consider the **my_scc_channel** channel:

```
   class my_scc_channel: public scc_channel_if {
   public:
     scc_port<scc_instream_if<unsigned_char> > datain;
     scc_port<scc_outstream_if<unsigned_int> > dataout;
   };
```

The following code shows a stream binding for a channel declared in procedure scope:

```
void ppa () {
  scc_stream<unsigned char> in_stream;
  scc_stream<unsigned char> out_stream;
  my_scc_channel mychannel;
  mychannel.datain(in_stream);
  mychannel.dataout(out_stream);

  // user code
}
```

The following code shows a stream binding for a channel declared in global scope:

```
scc_stream<unsigned char> in_stream;
scc_stream<unsigned char> out_stream;
my_scc_channel mychannel(in_stream, out_stream);
void ppa () {
  // user code
}
```

WHAT NEXT

Ensure that all stream ports of channel are bound properly and binding is visible inside the PPA scope. If the channel is global then binding is allowed in the constructor.

COM-1040 (error) Stream is connected to multiple stream ports.

DESCRIPTION

An scc_stream can be connected to a channel's streaming ports. The **scc_instream_if** and **scc_outstream_if** are the interface classes provided to represent the type of input and output ports of a channel.

An scc_stream can be connected to at most one port of type **scc_instream_if** and at most one port of type **scc_outstream_if**.

This error message occurs when an scc_stream is connected to more than one port of the same type.

WHAT NEXT

Reduce the number connections for a given scc_stream to at the most one for each port type (input and output).

COM-1050 (warning) Use of hold_register pragma is only supported for scalars, external memory and output streams.

DESCRIPTION

The **hold_register** pragma is used to hold data stable on scalars, external memory and output streams. The stream input is sampled on valid handshake and does not require having a *hold_register*. Therefore, **#pragma hold_register** is not supported for input stream.

For example, the following function uses **#pragma hold_register** for all streams. In such cases a *hold_register* is generated in the RTL to hold the data stable on output stream only and the pragma is ignored for input stream.

```
scc_stream<long int> res;
#pragma hold_register res single_buffer

void calculate (scc_stream<int> &in1, scc_stream<int> &in2) {
#pragma hold_register in1 single_buffer
  res.put( (in1*in1) -2*in1*in2 +(in2*in2));
}
```

WHAT NEXT

Remove "#pragma hold_register" for the input stream.

COM-1052 (warning) Invalid usage of label "%s", ignoring.

DESCRIPTION

Labels (i.e. "$SCC label mylabel") can be used: 1) to apply no-dependence constraints on memory, 2) to control scheduling of different operations using wait constraints, or 3) to name a class member function for synthesis. Usage of a label at any other place is considered invalid and is ignored.

WHAT NEXT

USe the labels for the correct purpose. See reference manual for more details on label usage.

COM-1057 (error) Loop with "#pragma max_schedule_length" specification cannot be unrolled.

DESCRIPTION

When a loop is unrolled, it becomes straight line code. The Syphony C Compiler tool does not support controlling the schedule length of straight line code. Therefore, do not use "#pragma unroll" and "#pragma max_schedule_length" together.

WHAT NEXT

Remove "#pragma max_schedule_length". To control schedule length of straight line code, insert a dummy outer loop with iteration count=1 and use "#pragma max_schedule_length" on the dummy loop.

For example, following code generates an error message:

```
#pragma max_schedule_length 1
#pragma unroll
for(int i=0; i<2; i++) {
   // user code
}
```

This code can be re-written as follows to achieve desired constraints:

```
#pragma max_schedule_length 1
for(int j=0; j<1; j++) {
  #pragma unroll
  for(int i=0; i<2; i++) {
     // user code
  }
}
```

COM-1058 (warning) '#pragma no_memory_analysis' is deprecated, please use no_dependence pragma.

DESCRIPTION

The **no_memory_analysis** pragma is deprecated. This pragma disables all analysis of array accesses in each loop and effectively treats all distances between any accesses as infinite. However, most applications might have finite aliasing distances between such accesses.

The Synphony C Compiler tool supports specification of such finite distances using the new **no_dependence** pragma. Unlike the **no_memory_analysis** pragma, the **no_dependence** pragmas is validated during the **verify -lint** command. The syntax is:

```
#pragma no_dependence <array> [source-access:[source-region]] [sink-
access:[sink-region]] [distance]
```

Consider the following code segment:

```
int X[20][20];
#pragma no_dependence X read:e1 write:e2 3

for ( ) {
    ...
    p = /*$SCC label e1*/X[i+a][j+b];
    ...
    /*$SCC label e2*/X[i+c][j+d] = q;
    ...

}
```

This pragma usage implies that reads of array X labeled with "e1" do not alias with next three writes labeled with "e2".

WHAT NEXT

Analyze your design and understand the distances between different accesses and use the **no_dependence** pragma with finite distances as a replacement for the **no_memory_analysis** pragma specification.

COM-280 (error) Unsupported usage of pragma packed on field of a struct.

DESCRIPTION

To generate a packed data port containing all fields of the struct, specify the **#pragma packed** pragma to a struct. By default, the Synphony C Compiler tool creates multiple, field-wise data ports if you do not specify **#pragma packed** pragma. The tool does not allow packing of individual struct elements.

WHAT NEXT

Remove the **#pragma packed** pragma specification on the struct element.

COM-290 (error) stable ports of a channel cannot be bound to a local variable.

DESCRIPTION

In the Synphony C Compiler tool, a channel can have stable inputs, stream input, and stream output.The tool does not support binding a local variable to

a stable input port of a channel. The following example binds the stable input port of a channel to a local variable, which is not supported.

```
        void my_design (struct A &var) {
          scc_stream <uint8> ch_in;
          scc_stream_with_get_trigger<uint16> ch_out;
          char mode = var.mode;
          my_channel ch_obj(ch_in, ch_out, mode); //
 binding local variable mode
          // mode is bound to scc_in<uint1> port of the channel
          //
 declaration of scc_in makes the port stable inside the channel

          ch_out(ch_obj);
          ...
        }
```

WHAT NEXT

Modify the code to avoid binding local variables to stable input ports of the channel.

SEE ALSO

COM-1038

COM-342 (warning) overflow in constant expression.

DESCRIPTION

This warning message is generated when you have computations in your design that exceed the width of the datatype, after the constants involved in the expression are propagated to the variables. In the following example, a width overflow occurs because **c** is of **unsigned char** type.

```
    unsigned char a,c ;
    a = 255 ,
    c  = a + 2;
```

But, there would be no overflow when:

- "c" is of "unsigned short" type.
- if the expression was already a constant to begin with. c = 255 + 2;

This warning message is generated only for generic C/C++ types and does not come up for variables bitsized with "#pragma bitsize".

WHAT NEXT

Check your code and correct the types of the variables where overflow is indicated.

COM-515 (error) Unsupported pointer operation in the synthesizable code.

DESCRIPTION

The Synphony C Compiler tool does not support the following pointer operations in the synthesizable code:

• Global pointer or pointer declared with the **static** keyword; only procedure scope pointers are supported.

• Pointer as members of structs or classes.

• Pointer to a pointer or array of pointers.

• Conversion of a pointer type. For example, the following usage is not allowed:

```
int *temp;
  char *temp2;
  temp = (int*)temp2; //This is not allowed.
```

When you use a pointer in your synthesizable code, the pointer must resolve statically, that is, the location or variable that the pointer is referring must be known at compile time. For example,

```
char bufA[1024], bufB[1024];
...
char *ptr; // procedure scope declaration
...
if (parity)
  ptr = bufA;
else
  ptr = bufB;
...= ptr[addr];
```

In this example, the **ptr** pointer is bound to **bufA** or **bufB** and the tool synthesizes the required multiplexer logic to choose the appropriate buffer based on parity.

WHAT NEXT

Modify the synthesizable code to avoid the unsupported pointer operations. For more details about the supported pointer operations, see the *Synphony C Compiler Reference Manual*.

COM-516 (error) Synphony C Compiler does not support usage of non synthesizable fields inside synthesizable code.

DESCRIPTION

This error message is generated when the tool finds an unsupported syntax containing a field of a struct or class. In the following example, if you use pointer member inside a struct or class in the synthesizable code, the error message is generated.

```
class D3
{
public:
    int arr[4];
    int *ptr;
    int sum() {
       return arr[0]+arr[1]-arr[2]+arr[3];
    }
};
D3 obj;
int array[4] = {0};

int my_design() {
  obj.ptr = array;
  #pragma unroll
  for (int i = 0; i < 4; i++)
    obj.arr[i] = i;
  return obj.sum() - array[2];
}
```

WHAT NEXT

Avoid the unsupported syntax for fields of a struct or class in your synthesizable code.

COM-522 (error) Variable %s is of type float, which is not supported.

DESCRIPTION

The Synphony C Compiler tool does not support floating point operations in the synthesizable code. However, the tool supports floating point operations in

the testbench code. You can assign a floating point constant to a variable of type **s_fixed** or **s_unfixed**. Alternatively, you can initialize a constant fixed point array to floating point constants. For example, the following code is synthesizable:

```
const s_fixed<12,0,S_RND,S_SAT> taps[NTAPS] = {
  -0.032226562500,
  -0.052978515625,
  -0.044921875000,
   0.000000000000,
   0.074951171875,
   0.159179687500,
   0.225097656250,
   0.250000000000,
   0.225097656250,
   0.159179687500,
   0.074951171875,
   0.000000000000,
  -0.044921875000,
  -0.052978515625,
  -0.032226562500
};
```

You can also use floating point operations in constant expressions as shown in the following example:

```
const int a = 3.14159 * 2048;
```

WHAT NEXT

• Do not use floating point types and operations in your synthesizable code.

• Convert floating point operations to fixed point operations. Consider using Synphony C Compiler tool’s fixed point library for this conversion.

COM-523 (error) Found a typecasting operation to float/double type. Float/double typecasting is not supported.

DESCRIPTION

The Synphony C Compiler tool does not support floating point operations in the synthesizable code. This error message is generated when there is an implicit or explicit typecast to a float type in the synthesizable code. The example shows an implicit typecast to a float type.

```
int val = 1.0 * t5; // t5 is int type
```

This type of implicit typecasting to a float type is not supported.

WHAT NEXT

To avoid this error message,

• Do not use float types and operations in the synthesizable code.

• Convert the floating point operations to fixed point operations. Consider using the tool’s fixed point library for this conversion.

SEE ALSO

COM-522

COM-537 (error) The bitsize and types declaration are not compatible for ’#pragma %s %s ...’.

DESCRIPTION

The size that you specified in the bitsize pragma must be less than or equal to the bitsize of the variable’s native data type. In the following example, the bit size of the unsigned short variable var is declared as 64 bits, while C/C++ uses 16 bits for unsigned short variable:

```
unsigned short var;
#pragma bitsize var 64
```

WHAT NEXT

To avoid this error message, do one of the following:

• Modify the size specified in the bitsize pragma declaration.

• Use the appropriate C/C++ native type for the variable declaration.

COM-556 (error) Definition for function "%s" is missing from synthesizable application files.

DESCRIPTION

This error message is generated when the target function to be synthesized calls a function whose definition is missing in the application files that you specified using the **set_params -appfiles** command.

In some cases, using the C/C++ library functions can cause this error message. For example, using the **printf** statement in the synthesizable code.

WHAT NEXT

Ensure that the function definition is present in the specified application files. If you have used library functions for debugging, enclose them in **#ifndef** and **#endif** statements to avoid compiling them as synthesizable code, as shown in the following example:

```
#ifndef SCC
   // Put non-synthesizable code here
   printf (...);
#endif
```

COM-568 (error) Synphony C Compiler does not support data shape changes during function calls. Actual argument not compatible for function '%s' argument %d.

DESCRIPTION

You can pass an entire array to a TCAB or importable PPA. Passing of arrays to a TCAB or importable PPA is not allowed when the addressing mechanism in container is different from addressing mechanism in the TCAB or importable PPA. This is not allowed even if the array is banked. For example,

```
// declaration seen by container
int A[1][MAX];
// calling TCAB / importable PPA from container
block (A[0]);
// block prototype is: void block(int x[MAX])
// block is synthesized as a TCAB or importable PPA
```

The above example will result in this error. In the container you access the element of the array "A" using the operation A[0][i] and in the TCAB or importable PPA you would access an element of "A" using A[i].

Another example using struct is given below.

```
typedef struct {
  int m[MAX_SIZE];
}mem_type;
mem_type mem[1]; // declaration seen by container

block(mem[0].m);
// block prototype is: void block(int mem[MAX_SIZE])
// block is synthesized as TCAB or importable PPA
```

WHAT NEXT

Avoid the coding style where the addressing mechanism is different in container and TCAB or importable PPA. In the example, you can change the declaration of array "A" and struct mem_type as below.

```
int A[MAX]; // declaration changed from int A[1][MAX];
mem_type mem; // declaration changed from mem[1]
```

SEE ALSO

COM-915

COM-639 (error) %s named "%s" is not found in the input files.

DESCRIPTION

This error message indicates that the target specified for building the implementation is not present in one of the application files.

WHAT NEXT

To avoid the error message, do the following:

1. Check that the **target** parameter is set correctly.

2. Confirm that the target is present in one of the files specified in the **appfiles** parameter.

3. If the **appfiles** parameter is set to *all*, confirm that the target is present in one of the files specified by the **sources** parameter.

4. If you are using a class member function as your target, use **$SCC label <name>** to name the member function and set the **target**=*<name>* parameter.

5. If you are using a template function or overloaded function, use **$SCC label <name>** to name the specific instance of the function and then set the **target**=*<name>* parameter.

6. If you are synthesizing a channel, make sure that you specified the name of the channel class in the **target** parameter.

7. If you are synthesizing a template channel, use **$SCC label_type <name>** to name the specific instance of the template channel and set the **target**=*<name>* ***parameter.***

COM-6435 (information) Loop %d has only dead operations whose computed values are not used. %s.

DESCRIPTION

If any loop in your design has some functional code, which is not being used for producing actual output of the design, it is considered as dead code. The tool can not analyze or synthesize such loops.

WHAT NEXT

Remove the unused loops to avoid this error message.

COM-6545 (error) Shared stall domain TCAB cannot be synthesized with the specified %s constraint of %d %s. Try synthesizing the TCAB by either setting 'set_params -allow_latency_violation yes' or by increasing the %s specification.

DESCRIPTION

The tool issues this error when you specify **set_params -allow_latency_violation no** command and any one of the following constraints could not be met for a shared stall domain TCAB:

```
set_params -output_production_time num
set_params -resource_last_use_time num
set_params -architectural_pipelinability num
```

WHAT NEXT

If you do not intend to use a shared stall domain TCAB, use the **set_params -allow_latency_violation yes** command. If the tool cannot synthesize a shared stall domain TCAB using the specified latency constraints, it synthesizes an independent stall domain TCAB.
Or
Remove or relax the latency constraint for the TCAB. This can result in a shared stall domain TCAB.
Note:
To synthesize a shared stall domain TCAB, the code must contain fixed bound loops and the execution of any loop must be unconditional. Or
Use report_feedback_path.txt file or the Feedback Path Viewer in the GUI to find the feedback path that violates the latency constraints.

COM-658 (error) At least one loop failed unroll %s.

DESCRIPTION

For unrolling a loop using the unroll pragma, the Synphony C Compiler tool must know the maximum number of iterations the loop will execute. This error message is generated when the tool cannot determine the maximum number of iterations required to execute the loop. In the following example, the tool cannot unroll the loops as it cannot determine the maximum number of iterations required to execute the loops.

```
        #pragma unroll
        while (condition) {
          //
 the number of times the loop will execute is not known at compile time
          // Loop Body
        }

        #pragma unroll
        for (int i = 0; i < n; i++) {
          // n is not known at compile time
          // Loop Body
        }
```

WHAT NEXT

Specify the maximum number of iterations using the **num_iterations** pragma as shown in the following example:

```
       #pragma unroll
       #pragma num_iterations(,,10)
       while (condition) {
         // Loop Body
       }

       #pragma unroll
       #pragma num_iterations(,,5)
       for (int i = 0; i < n; i++) {
         // Loop Body
       }
```

COM-6636 (warning) interface_type " memory_interrupt" is deprecated

DESCRIPTION

This warning message occurs when you specify a deprecated option for the **interface_type** parameter.

WHAT NEXT

The supported values for the **interface_type** parameter are *handshake* and *memory_mapped_handshake*.

COM-6670 (error) The access window of resource %s cannot be constrained to %d cycles in loop %d.

DESCRIPTION

For designs with **continuous_processing** parameter set to *always*, the tool constrains the resource access window of the PA to be equal to the initiation interval (II) of the PA. This error is issued when the tool cannot meet this constraint for a resource in the PA.

WHAT NEXT

Use the report_feedback_path.txt file or the Feedback Path Viewer in the GUI to analyze the feedback path that prevents the resource access window from being equal to the II of the PA. Make the necessary modifications to avoid the feedback path. For more information about the Feedback Path Viewer, see the *Synphony C Compiler User Guide*.

COM-6687 (error) User given constraint "%s" failed at II=%d

DESCRIPTION

The **build** command generates this error message when a hard constraint cannot be met at the specified II. The Synphony C Compiler tool relaxes this constraint to generate RTL.

WHAT NEXT

Use the report_feedback_path.txt file, the Feedback Path Viewer, or the Operation Schedule Viewer in the GUI to understand why the constraint could not be satisfied. Update the constraints or modify your code to satisfy the performance requirements of the design.

COM-6688 (warning) Ignoring parameter "%s" since it is set to "%s" which is a reserved Verilog identifier.

DESCRIPTION

Parameters that control names in the generated Verilog must not use reserved Verilog identifiers. The Synphony C Compiler ignores such settings.

The list of reserved Verilog identifiers is: **timescale**, **define**, **ifdef**, **ifndef**, **endif**, **include**, **module**, **parameter**, **defparam**, **input**, **output**, **inpout**, **supply0**, **strong0**, **pull0**, **weak0**, **highz0**, **supply1**, **strong1**, **pull1**, **weak1**, **highz1**, **large**, **medium**, **small**, **wire**, **tri**, **tri0**, **tri1**, **wand**, **triand**, **wor**, **trior**, **trireg**, **reg**, **integer**, **time**, **real**, **assign**, **buf**, **not**, **and**, **nand**, **or**, **nor**, **xor**, **xnor**, **bufif0**, **bufif1**, **notif0**, **notif1**, **nmos**, **pmos**, **cmos**, **rnmos**, **rpmos**, **rcmos**, **tran**, **tranif0**, **tranif1**, **rtran**, **rtranif0**, **rtranif1**, **pullup**, **pulldown**, **primitive**, **table**, **endtable**, **endprimitive**, **always**, **initial**, **begin**, **end**, **posedge**, **negedge**, **if**, **else**, **case**, **casex**, **casez**, **default**, **enadcase**, **for**, **forever**, **while**, **repeat**, **wait**, **fork**, **join**, **task**, **endtask**, **function**, **endfunction**, **force**, **release**, **specify**, **specparam**, and **endspecify**.

WHAT NEXT

Change the parameter’s value so that it does not collide with reserved Verilog identifiers.

COM-6690 (warning) Ignoring parameter "%s" since the key "%s" is invalid.

DESCRIPTION

Some Synphony C Compiler parameters use keys and must be set with the syntax **"key:signal_name ..."**. You have specified an invalid key for the parameter.

The **stream_port_polarity** parameter keys are **get_req**, **can_get**, **put_req**, and **can_put**.

The **system_port_name** parameter keys are **get_req**, **can_get**, **put_req**, and **can_put**.

The **system_port_polarity** parameter keys are **enable**, **abort**, **error**, **start_valid**, **start_ready**, **done_valid**, and **done_ready**.

The **system_port_production** parameter keys are **scan_in**, **scan_en**, **scan_out**, **error**, **start_ready**, and **done_valid**.

WHAT NEXT

Make sure that the parameter’s value has valid keys.

COM-6701 (error) The value "%s" for parameter "system_port_name" is a reserved word.

DESCRIPTION

Reserved words are default system interface port names used by the Synphony C Compiler tool. The value of parameter **system_port_name** should not collide with any reserved words.

The list of reserved words is: **clk**, **reset**, **enable**, **start_task_init**, **start_task_final**, **clear_init_done**, **clear_task_done**, **psw_livein_frames_in_use**, **psw_liveout_frames_in_use**, **psw_released**, **psw_busy**, **psw_idle**, **psw_init_done**, **psw_task_done**, **err**, **abort**, **psw_aborting**, **aborting**, **start_valid**, **start_ready**, **done_valid**, **done_ready**, **host_addr**, **host_rdata**, **host_wdata**, **host_req**, **host_mode**, **host_ack**, **scan_in**, **scan_out**, and **scan_en**.

WHAT NEXT

Change your system port names so that they do not collide with reserved words.

COM-6714 (error) User given constraint "%s" was violated on the following set of ops due to their latency: %s.

DESCRIPTION

This error message is generated when a hard user constraint cannot be met because of the operation’s latency. The Synphony C Compiler tool generates this error message and relaxes this constraint to generate the RTL.

WHAT NEXT

Analyze the operators in the Operation Schedule Viewer to understand the conflict. Do one of the following:

- Update your constraints to match possible performance of the design

- If the operation is a TCAB, see if you can reduce the latency of the TCAB.

COM-6720 (error) Parameter "design_intent" is set to "import" which is not supported for %s implementations.

DESCRIPTION

Importable PPAs do not have support for autostart, infinite run, PSW interfaces, or disabling of the PPA on host access. Such features are only available for deployable PPAs.

WHAT NEXT

If you intend to deploy this PPA, then set the **design_intent** parameter to *deploy*. If you intend to import this PPA, then disable the features specific to deployable PPAs.

COM-680 (warning) Potentially uninitialized reference of variable ’%s’.

DESCRIPTION

The Synphony C Compiler tool produces this warning in build phase when it is unable to prove that a variable declared inside a loop is always written before it is used.

In the following example, a **tmp** variable is uninitialized in some cases before it is used resulting in the warning message.

```
int my_design() {
  int sum;
  for (int j = 0; j < 10; j++) {
    int tmp;
    if (j == 0) {
      tmp = 0;
    } else {
      sum += tmp;
    }
  }
  return sum;
}
```

WHAT NEXT

This warning message occurs during the build flow. In a few cases, the design might avoid that path during a simulation. If the **verify -lint** command does not generate a warning (with a testcase with good code coverage) message about an uninitialized variable, ignore this warning message generated during the build phase.

COM-712 (warning) The internal stream %s feeds back from loop %d to loop %d in program order. This may cause the golden model to mismatch the RTL execution.

DESCRIPTION

Your code has an internal stream with a read in one loop and a write in another loop that appears later in program order. For example:

```
scc_stream <int> x;
for() { // loop 0
  x.get();  // Read from stream in an earlier loop.
}

for () { // loop 1
  x.put();  // Write to stream in a later loop
}
```

Loop 0 is executed first in program order for a particular task; therefore, the Synphony C Compiler tool generates this warning message.

WHAT NEXT

This warning message is only a guard against application errors so that you can check your design for correctness. If the usage is correct, ignore this warning message.

COM-788 (warning) "#pragma no_hostlm_access" is deprecated. Please use "#pragma host_memory_access %s none" to disable host access.

DESCRIPTION

WHAT NEXT

Replace "#pragma no_hostlm_access" with "#pragma host_memory_access arrayName none" for the arrays which you do not want to have host access.

COM-794 (warning) Pragma specification is ignored for parameters of an inlined function. It should be specified on the actual parameter in the calling function. Ignoring pragma '%s' for the parameter '%s' in the inlined function '%s'.

DESCRIPTION

Any function in your design is imported (as pre-synthesized hierarchical blocks) or inlined into a top-level design. The Synphony C Compiler tool ignores any pragma specification on formal parameters of inlined functions. In such cases, only pragmas specified on the variables at the call point of such inlined function are accepted by the tool.

Consider the following example:

```
      void RLD(short tmp_zzin[64]) {
         #pragma internal_fast tmp_zzin
 ...
      }


      void DeQuantizer(short tmp_zzin[64]) {
         #pragma user_supplied tmp_zzin
 ...
      }

      void IDCT(short outBlock[64]) {
         #pragma user_supplied outBlock
 ...
      }

      void MotionComp(short tmp_dctout[64]) {
         #pragma user_supplied outBlock
 ...
      }

      void decode(void) {
        // only used locally
        short tmp_zzin[64];
        short tmp_block[64];

// Properties of arrays at call points
        #pragma user_supplied tmp_zzin
        #pragma user_supplied tmp_block

        RLD(tmp_zzin);           // Inlined
        DeQuantizer(tmp_zzin);   // Inlined
        IDCT(tmp_block);         // PPA
        MotionComp(tmp_block);   // TCAB
      }
```

In the example, the **decode()** design calls four functions which accept array parameters. Different memory implementation properties (**internal_fast** vs **user_supplied**) are applied inside all the functions. Because the **RLD()** and **DeQuantizer()** functions are inlined into the **decode()** function, the tool ignores the pragma on the parameter array **tmp_zzin[64]** in such cases and applies only the **user_supplied** pragma from the call location in **decode()** function on that array.

However, because the **IDCT()** and **MotionComp()** functions are targeted for synthesis as PPA and TCAB respectively, the tool applies the memory implementation pragma in each of the **IDCT()** and **MotionComp()** functions functions when the functions are synthesized as PPA or TCAB.

Since the RLD() and DeQuantizer() functions are inlined into the decode() function, the pragma on the parameter array tmp_zzin[64] is ignored in such cases and only the user_supplied pragma from the call location in decode() function is applied on that array.

In the scope of the **decode()** function, the **tmp_block** is of **user_supplied** type and the synthesis flow might error out because the **MotionComp()** TCAB might expect an **internal_fast** array to be passed to the function.

WHAT NEXT

Ignore this warning message if you have already applied the same pragmas on the variables being passed to the functions. If you want to completely remove the warnings, remove the pragma specification in the function body.

COM-820 (warning) Revising the upper bound on trip count from %d iterations to %d iterations based on the upper bound expression of the loop.

DESCRIPTION

The **analyze** command analyzes **num_iterations(MIN,TARGET,MAX)** specifications for each loop against its trip count variable. When the specified **MAX** value of a **num_iterations(,,MAX)** pragma specification is less than loop bound variable of that loop, the **analyze** command generates this warning message.
For example, in following code the **MAX** value of the **num_iterations** specification is 8, but the actual trip count is 10.

```
#pragma num_iterations (1,,8)
for (int i=0;i<10;i++) {
    // Loop body
}
```

The **analyze** command generates this warning message and revises the **MAX** value from 8 to 10 to avoid inconsistency.

WHAT NEXT

Specify a **MAX** value which is not less than actual trip count of that loop. Note that the tool uses the **MAX** value in a **num_iterations** specification to decide the bitsize of the loop counter and related logic.

COM-824 (error) Usage of channel member '%s' in a non-channel design is not allowed.

DESCRIPTION

A channel communicates using ports declared in the channel class. The members of a channel class can be used only by the channel design. This error message is issued when you access the members of a channel class in a non-channel design, as shown in the following example:

```
void my_design(uint1 &mode) {
  scc_stream <uint8> ch_in1;
  scc_stream <uint8> ch_in2;
  scc_stream_with_get_trigger<uint16> ch_out;

  my_channel ch_obj(ch_in1, ch_in2, ch_out, mode);
  ch_out(ch_obj);

  ch_obj.val = 1; // Not allowed
  ...
}
```

WHAT NEXT

Do not access the member of a channel class in a non-channel design. To put or get data from the channel, use the ports of the channel.

COM-825 (error) Array of pointers in type/subtype of '%s' is not allowed in the synthesizable code.

DESCRIPTION

The Synphony C Compiler tool does not support the following pointer operations in the synthesizable code:

• Global pointer or pointer declared with **static** keyword; only procedure scope pointer is supported.

• Pointer as a member of struct or class type.

• Pointer to a pointer or an array of pointers.

• Conversion of pointer type.:w
When you use a pointer in the synthesizable code, the pointer should resolve statically, that is, the location or the variable the pointer is referring to must be known at compile time.

WHAT NEXT

Modify the synthesizable code to avoid using an array of pointers. For more details about the supported pointer operations, see the *Synphony C Compiler Reference Manual.*

SEE ALSO

COM-515

COM-826 (error) Pointer to pointer in type/subtype of '%s' not allowed in the synthesizable code.

DESCRIPTION

The Synphony C Compiler tool does not support the following pointer operations in the synthesizable code:

• Global pointer or pointer declared with **static** keyword; only procedure scope pointer is supported.

• Pointer as a member of struct or class type.

• Pointer to a pointer or an array of pointers.

• Conversion of a pointer type.
When you use a pointer in the synthesizable code, the pointer should resolve statically, that is, the location or the variable the pointer is referring to must be known at compile time.

WHAT NEXT

Modify the synthesizable code to avoid using a pointer to a pointer.

SEE ALSO

COM-515

COM-827 (error) Global or static pointer '%s' is not allowed in the synthesizable code.

DESCRIPTION

The Synphony C Compiler tool does not support the following pointer operations in the synthesizable code:

• Global pointer or pointer declared with **static** keyword; only procedure scope pointer is supported.

• Pointer as a member of struct or class type.

• Pointer to a pointer or an array of pointers.

• Conversion of a pointer type.
When you use a pointer in the synthesizable code, the pointer should resolve statically, that is, the location or the variable the pointer is referring to must be known at compile time.

WHAT NEXT

Modify the synthesizable code to avoid global pointers or pointers declared with the **static** keyword.

SEE ALSO

COM-515

COM-842 (warning) Ignoring pragma %s on pointer variable '%s'.

DESCRIPTION

This warning is generated when you specify a pragma that has no effect on pointer variables. For example,

```
unsigned short mem[100];
#pragma bitsize mem 12
int my_design(int index) {
  unsigned short * pin = 0;
  #pragma bitsize pin 12
  pin = mem;
#pragma user_supplied pin
  return pin[index] * pin[index];
}
```

The tool ignores the **#pragma user_supplied pin** and accepts the **#pragma bitsize** pragma.

WHAT NEXT

Specify the pragmas that are applicable to pointer variables. In the above example, you can specify the **#pragma user_supplied** on the **mem** array.

COM-878 (information) A flush pragma on the outer-most loop of a loop nest has no effect.

DESCRIPTION

For nested looping code, the Synphony C Compiler tool builds a pipeline of operations to achieve higher performance. At any given cycle, there can be many loop iterations in flight. The **flush** pragma can be used to flush iterations corresponding to a particular loop. Because the outermost loop is flushed automatically after the loop completed, a **flush** pragma on an outermost loop has no effect.

For example, the following doubly nested loop processes an HxW image frame in video processing. The input is received row-by-row in burst mode inside the inner loop. With pipelined operation, when inner loop is processing few last pixels of previous row, the next iteration can start in parallel and start accessing the next row from the input. If the next row is not available, the hardware stalls and inturn freezes the processing of the previous row.

```
for(r=0; r<H; r++) {
  for(c=0; c<W; c++) {
    int pix = inpix.get();
    int new_pix = process_pix(pix); // processing has (n>1) latency
```

```
      outpix.put(new_pix);
    }
  }
```

To avoid the stalling of inner loop, use the **#pragma flush** pragma on the inner loop to ensure that the pipeline is flushed before the next iteration of the outer loop.

```
  for(r=0; r<H; r++) {
    #pragma flush
    for(c=0; c<W; c++) {
      int pix = inpix.get();
      int new_pix = process_pix(pix); // processing has (n>1) latency
      outpix.put(new_pix);
    }
  }
```

Application of the flush pragma on outer loop has no effect.

WHAT NEXT

Remove the **#pragma flush** pragma from the outer loop. If you intended to flush a different loop, specify the pragma to that loop.

COM-881 (warning) Use of "input_stream", "output_stream" or "connect" pragma on class methods is obsolete. Use the Synphony C Compiler scc_stream objects instead.

DESCRIPTION

The **input_stream**, **output_stream** or **connect pragmas**, which were used to convert any given class method into stream interface are now obsoleted. To get a similar stream interface, you can use the Synphony C Compiler scc_stream class.

WHAT NEXT

Replace your class methods marked **input_stream**, **output_stream** or **connect** with an scc_stream class object. For example, if your design has class methods treated as a stream interface using these obsoleted pragmas, rewrite them as:

```
// BEFORE
  class mydesign {
  public:
    unsigned int x;
    void send(unsigned int x);
    #pragma output_stream send
    unsigned int receive();
    #pragma input_stream receive
```

```
    void myprocess() {
      x = receive();
      send(x);
    }
  };

// AFTER
  class mydesign {
  public:
    unsigned int x;
    scc_stream <unsigned int> send;
    scc_stream <unsigned int> receive;

    void myprocess() {
      x = receive.get();
      send.put(x);
    }
  };
```

COM-896 (error) Unsupported usage of virtual classes .

DESCRIPTION

This error message is generated when you use virtual inheritance in synthesizable code. For example,

```
class Base
{
public:
  int arr[4];
  void set(int i, int val) {
    arr[i] = val;
  }
  virtual int sum() = 0;
};

class D1: virtual public Base
{
public:
    int get(int i) {
       return arr[i];
    }
  virtual int sum() = 0;
};

class D2: virtual public Base
{
public:
  virtual int sum() = 0;
  void init(int x){
```

```
    arr[0] = arr[1] = arr[2] = arr[3] = 0;
  }
};

class D3: public D1, public D2
{
public:
    int sum() {
       return arr[0]+arr[1]+arr[2]+arr[3];
    }
};

int my_design() {
  D3 obj;
  obj.set(0,1);
  obj.set(1,1);
  obj.set(2,1);
  obj.set(3,1);
  return obj.sum();
}
```

In the above code, class D3 uses virtual inheritance which is not supported.

WHAT NEXT

Avoid using virtual inheritance in the synthesizable code.

COM-899 (error) Objects of classes with virtual functions or virtual base classes cannot be declared inside synthesizable code.

DESCRIPTION

The Synphony C Compiler tool does not support virtual functions or virtual inheritance inside synthesizable code. For example,

```
class Base
{
public:
  int arr[4];
  void set(int i, int val) {
    arr[i] = val;
  }
  virtual int sum() { return 0;};
};

class D3: public Base
{
public:
    int sum() {
```

```
        return arr[0]+arr[1]+arr[2]+arr[3];
    }
};

int my_design() {
  D3 obj;
  obj.set(0,1);
  obj.set(1,1);
  obj.set(2,1);
  obj.set(3,1);
  return obj.sum();
}
```

WHAT NEXT

Avoid using objects with virtual functions or virtual inheritance in the synthesizable code.

COM-915 (error) Cannot pass array slice of '%s' as function argument to the TCAB function '%s'.

DESCRIPTION

You can pass an entire array to a TCAB or importable PPA. Passing a part of the whole array to a TCAB or importable PPA is not allowed. Even in cases where the array is banked, this is not allowed. For example,

```
// declaration seen by container
int A[2][MAX];
// calling TCAB/ importable PPA from container
block1 (A[0]);
block2 (A[1]);
// block1 prototype is: void block1(int x[MAX])
// block2 prototype is: void block2(int x[MAX])
// block1 and block2 are synthesized as a TCAB or importable PPA
```

The above example will result in this error.

Another example where a part of single dimensional array is passed is given below.

```
// declaration seen by container
int A[MAX+5];
// calling TCAB from container
block (&A[5]);
// block prototype is: void block(int x[MAX])
```

WHAT NEXT

Avoid the coding style of passing a part of the array to the TCAB. In the example, you can declare two separate arrays "A" and "B".

```
int A[MAX];
int B[MAX];
block1 (A);
block2 (B);
```

In the second example above, you can pass the entire array to block and access the locations starting from index 5 inside block.

```
block(A); // changed from block (&A[5]);
```

SEE ALSO

COM-568

COM-987 (warning) '#pragma host_access' is deprecated.

DESCRIPTION

This warning message is generated when you use **host_access** pragma in the synthesizable code.

WHAT NEXT

Use the **host_memory_access** pragma instead of the **host_access** pragma.

SIM Error Messages

SIM-2000 (warning) Width overflow for "%s", type %d signed bits (i%d), value %d (%#x).

DESCRIPTION

This warning message is generated when a larger bitsized value is assigned to a lower bitsized variable. For example, if you define a variable having 6 bit width and assign value larger than 6 bit width results in the overflow warning message.

```
s_uint<6> out;
out = 73;
```

WHAT NEXT

Fix your source code by increasing the bitwidth the of the variable so that it can handle the full range of value assigned to it. Observe the filename or the line number in warning message and find the occurrence of that variable in your source code.

SIM-2001 (warning) Width overflow for "%s", type %d unsigned bits (ui%d), value %u (%#x).

DESCRIPTION

Bitsize pragmas are used on native C datatypes to convey that bits over a certain width are not required (i.e. are don’t care) in the generated RTL. Since bitsize pragmas do not affect the semantics of the golden model, the Synphony C Compiler tool provides a simulation step that verifies these pragmas dynamically. The **verify -lint** command checks for overflow of variables with bitsize pragmas, generates this warning message when overflow occurs, randomizes the overflow bits, and continues with the simulation.

```
unsigned int out;
#pragma bitsize out 6
out = 173;
```

WHAT NEXT

One option is to increase the bitwidth of the variable so that it can handle the full range of value assigned to it. Another option is to add masking

operations to prevent the overflow from occurring. For example, following code restricts **out** to six bits and prevents overflow. out = 173 & 0x3f

SIM-2003 (warning) Uninitialized reference "%s".

DESCRIPTION

When you run the **verify -lint** command, the tool found a variable that was read before it was written. If the final results of your code rely on implicit initialization done by the C/C++ compiler, then there could be simulation mismatches in the **verify -lint** or **verify -rtl** command execution.

For example, a procedure scope static variable is implicitly initialized to zero by the C/C++ compiler. If the **reset_data_registers** parameter is set to *no*, then the register associated with this variable will have no initial value in the generated RTL.

WHAT NEXT

To avoid this warning message, modify your code to explicitly initialize variables requiring initial values. If your design does not depend on an initial value for this variable, then your can ignore this warning message. You can also ignore this warning message for uninitialized global and procedure scope static variables if the **reset_data_registers** parameter is set to *yes*.

SIM-2038 (warning) Shift value %lld out of legal range (%lld,%lld).

DESCRIPTION

Your code contains a bitwise shift operation using **'<<'** or **'>>'** operators and the shift amount is larger than the width of the datatype. For example:

```
int data = yin.get();
int left_shift = data >> 32;
x.put(left_shift);
```

The variable **data** is shifted by 32 bits. The result of such an operation is always 0.
NOTE: This warning is not printed in cases of overflows due to bitsize pragmas. It is only printed when the native datatype overflows.

WHAT NEXT

Most applications do not require shifts by such large amounts. Check your application to see if the shift values are correct. The position (file:line) of the possible illegal shift is indicated in the warning message itself.

SIM-2042 (error) Actual number of iterations (%d) of loop at %s:%d is less than the minimum number of iterations (%d) specified by "#pragma num_iterations".

DESCRIPTION

The **verify -lint** command compares the **#pragma num_iterations(MIN,TARGET,MAX)** values against the dynamic trip counts found during simulation. The specified **MIN** value is greater than the dynamic value for at least one execution of the loop.

In the following example, the **MIN** value of the **num_iterations** pragma is 10, but the actual trip count (based on the testbench code) is 5.

```
void foo(unsigned char N) {
  #pragma num_iterations (10,100,100)
  for (int i=0;i<N;i++) {
    // Loop body.
  }
}

void main() {
  foo(5);
}
```

The **verify -lint** command flags an error message that the actual number of iterations (5) is less than the **MIN** number of iterations (10).

WHAT NEXT

Remove the test cases that are inconsistent with the constraints of your design, or decrease the **MIN** value to allow all test cases to pass.

SIM-2043 (error) The actual number of iterations %d of the loop at %s:%d is more than the maximum number of iterations %d specified by "#pragma num_iterations".

DESCRIPTION

The **verify -lint** command compares the **#pragma num_iterations(MIN,TARGET,MAX)** values against the dynamic trip counts found during simulation. The specified **MAX** value is less than the dynamic value for at least one execution of the loop.

In the following example, the **MAX** value of the **num_iterations** pragma is 8, but the actual trip count (based on the testbench code) is 10.

```
void foo(unsigned char N) {
#pragma num_iterations (1,,8)
  for (int i=0;i<N;i++) {
    // Loop body.
  }
}
void main() {
  foo(10);
}
```

The **verify -lint** command flags an error that the actual number of iterations (10) is more than the **MAX** number of iterations (8).

WHAT NEXT

Either remove test cases that are inconsistent with the constraints of your design, or increase the **MAX** value to allow all test cases to pass.

SIM-2045 (error) Stream input call found no data in stream %s.

DESCRIPTION

This error is issued in golden simulation when you try to get a value from an empty FIFO.

WHAT NEXT

Use printfs or any debugging technique you are familiar with to identify the coding error.

SIM-2052 (error) Deadlocked due to insufficient inter-loop FIFO depth. All the loops are either stalling or inactive since last 1000 cycles. See the performance report for details.

DESCRIPTION

When scc_stream is used as an inter-loop FIFO between two loops (L0->L1), it allows parallel execution of both loops by enabling automatic flow control of streaming data between producer loop (L0) and consumer loop (L1). The consumer of the FIFO would stall if data is not available. Similarly, producer will stall if the FIFO is full. The situation where producer is stalled due to FIFO full and consumer is waiting for some event (scalar input) from producer before reading data from FIFO can cause deadlock.

For example, the following code can cause deadlock where L0 is writing 4 data to inter-loop FIFO before L1 starts reading data from FIFO. Deadlock can happen due to in-sufficient FIFO depth.

```
scc_stream<unsigned char> inter;
#pragma fifo_length inter 1
for(int i=0; i<4; i++) {
  inter.put(i);
  if(i==3)
    loop_end =1;
}
for(int i=0; i<4; i++) {
  if(loop_end==1)
    inter.get(i);
}
```

WHAT NEXT

In case deadlock occurred due to insufficient FIFO depth, you can see vlogsim performance report in "Reports" directory of the implementation and could find out which FIFO is causing the deadlock. For example, the following information in vlogsim performance indicates that **inter** FIFO needs more FIFO depth.

```
"Potential FIFOs for resize to achieve targeted performance:
ils_fifo_inter_len"
```

You can increase the FIFO depth to avoid deadlock. In the above example, L1 can not start reading data from inter-loop FIFO until L0 sets loop_end=1 while L0 has to write 4 data before it sets loop_end=1 which required inter-loop FIFO depth to be 4 to hold those 4 incoming data from L0. It means FIFO **inter** requires at least 4 depth to finish L0 and start L1. Changing FIFO depth to 4 will avoid the deadlock. Use the **fifo_length** pragma to specify FIFO depth.

```
#pragma fifo_length inter 4
```

SIM-2056 (warning) FIFO '%s' is NOT empty at the end of a task, %d value(s) remaining at the end of task %d. 'verify -rtl' may fail.

DESCRIPTION

This warning message is generated during lint simulation when a stream writes more data than it has consumed in a task. The FIFOs used in your code can carry data across tasks if they are global or static and the **verify -rtl** command would pass. If the FIFOs are used to carry data across tasks and are not global or static, then you would get a failure during the **verify -rtl** command.

WHAT NEXT

To avoid this warning message, you can,

- Make the FIFOs that could carry data across tasks global or static.

- If the FIFOs are not meant to carry data across tasks, use printfs or any debugging technique you are familiar with to identify the coding error.

SIM-2078 (error) Reference to index %d at address "%s" is beyond the end of the array.

DESCRIPTION

The **verify -lint** command checks for out-of-bounds array accesses. Each array dimension is checked independently based on the array declaration. For example, for an array declared as **int X[20][20]**, both **X[25][0]** and **X[0][25]** would be considered as out-of-bounds.

The following example contains a 16 element array **shift_reg** that is accessed out-of-bounds:

```
bool shift_reg[16];
scc_stream<bool> X;

for (int i=0;i<16;i++) {
shift_reg[i] = shift_reg[i+1];
}
shift_reg[15] = X.get();
```

WHAT NEXT

Increase the size of the array if it was wrongly sized or modify the access pattern to eliminate out of bound access.

Here is a code fix for the example case shown:

```
bool shift_reg[16];
scc_stream<bool> X;

for (int i=0;i<15;i++) {              // changed from 16
shift_reg[i] = shift_reg[i+1];
}
shift_reg[15] = X.get();
```

SIM-2080 (error) Dependence violation. From: %s at %s at %s:%d:%s. To: %s at %s at %s:%d:%s. Actual distance is %d; user specified distance is %d %s.

DESCRIPTION

During scheduling, the Synphony C compiler tool analyzes the memory accesses in the design. In some cases, the tool makes conservative decisions with respect to aliasing memory accesses. The **no_dependence** pragma specifies the distance between aliasing accesses.

This error message is issued during lint simulation, if there are aliasing accesses within the distance specified by the **no_dependence** pragma. The error message has the detailed trace of the loop iterations and the function calls where the aliasing access occurs.

WHAT NEXT

Specify the correct distance using the **no_dependence** pragma.

SIM-2084 (error) get() on an empty FIFO.

DESCRIPTION

This error is issued in golden simulation when you try to get a value from an empty FIFO.

WHAT NEXT

Use printfs or any debugging technique you are familiar with to identify the coding error.

SIM-5000 ERROR: lint and golden produced different output.

DESCRIPTION

This error indicates that the **verify -lint** and **verify -golden** simulations have produced different outputs.

WHAT NEXT

Examine and fix the warnings from **analyze** and **verify -lint**. Generally, mismatches are caused by application coding issues such as:

- bitwidth overflow
- out-of-bounds array access
- dependence on undefined C behavior
- uninitialized variables.

The warnings that are produced usually indicate the source position of the offending code. Fixing these warnings should get you past the **verify -lint** failure.

Even if the simulation passes, you should either fix **verify -lint** warnings or confirm that they are expected.

SIM-5001 (error) "verify -golden" returned non-zero (1) status

DESCRIPTION

The Synphony C Compiler tool requires your C/C++ simulation to return 0 for the simulation to be considered successful. For an nonzero issue, the following can be reasons:

1. Your testbench code does not have a return 0 at the end,

2. Your **verify -golden** simulation is getting a segmentation fault or is crashing

WHAT NEXT

Verify that the **verify -golden** command returns 0 on success. You can check this by compiling your code with a C/C++ compiler and running a debugger such as gdb.

SYN-575 (error) Cannot build a PPA which can sustain the Task II %d that you specified. Try a Task II of at least %d.%s.

DESCRIPTION

Task II (Task Iteration Interval) is a specification for minimum number of cycles between successive calls to the function which is mapped to hardware. When task II is specified, tool tries to achieve it. Achievable task II for the target function depends on the achievable II (iteration interval) of all the loops inside the function.
For example, the following target function has one loop having N iterations and specified Task II is N. The minimum N cycles between successive calls to foo can be achieved only if the internal loop can be finished in N cycles and that can happen only if loop can achieve II=1.

```
void foo() {
    for(int i=0; i<N; i++) {
        x[i] = x[i]*coeff1 + b[i]*coeff2;
    }
}
```

The loop is doing two accesses to memory x each iteration. If memory x has only one port in hardware, each loop iteration will require at least 2 cycles which can result in achievable II=2, and hence the function can achieve Task II = N*2. Specified constraints for Task II (**set_params -task_ii n**) is not achievable. Task II should be updated to N*2.

WHAT NEXT

To achieve task_II=N in this example, it is necessary to achieve loop II=1. This can be possible by increasing number of ports to 2 for memory x. For more information about Task II, refer to Performance Metric section under Programming Model Basics chapter in the Synphony C Compiler Reference Manual.

SYN-6432 (error) <%d>-ported native memory <%s> will be generated for array[s] '<%s>', but not supported in RTL simulation

DESCRIPTION

When an array is ported to external memory for synthesis and the design has many accesses to same array in one cycle, synthesis can generate more number of ports for accessing that memory. However the tool does not contain a

Verilog macrocell to support RTL simulation of such memory with the higher number of ports. The warning is generated to indicate the design issue and users should try to reduce the number of ports on the memory.

WHAT NEXT

Try to reduce the number of ports on the memory. There are different ways you can reduce the number of ports for memory.

1. Reduce the number of ports with the following pragmas and restrict it to limited number #pragma read_write_ports <array_name> combined [<n>] #pragma read_write_ports <array_name> separate[<r> readonly [<w> writeonly]]

2. Increase the loop II specification. This will spread the number of different accesses across different II phases causing a reduction in the number of simultaneous accesses to the array. This would lead to fewer ports for the array.

3. Reduce the number of ports by modifying code. If possible do common sub-expression elimination manually and remove multiple references with same indexes. i.e. Following code change will translate two accesses in to one access:

```
if(condition1) a = x[i];
   if(condition2) b += x[i];
```

it does two accesses to same index and can be re-written as follow to reduce access to one.

```
   tmp = x[i];
   if(condition1) a = tmp;
   if(condition2) b += tmp;
```

4. Change the implementation of array from external SRAM to a register based implementation using internal_fast pragma
a. For array X, add #pragma internal_fast X
b. When you define a memory as internal_fast, the tool implements it as registers and there is no limit on number of ports for register based memory. This allows the tool more flexibility to schedule operations, but the number of ports increases

SYN-6437 (error) The "build" command was unable to schedule loop %d in the specified number of attempts (%d).

DESCRIPTION

The Synphony C Compiler tool attempts to schedule the operations in a loop a fixed number of times. When it is not possible to schedule the operations in an attempt, the tool relaxes one or more constraints from a previous attempt and reschedules the operations. For example, in an attempt, the tool can choose to use more functional units or undo the sharing decisions. When the

tool cannot schedule the operations after a specific number of attempts, it generates this error message.

WHAT NEXT

Use the report_feedback_path.txt file or the Feedback Path Viewer in the GUI to analyze the critical timing paths. Even if the feedback paths do not show negative slack, enhance the most critical feedback paths. For more information about using the Feedback Path Viewer in the GUI, see the *Synphony C Compiler User Guide.*

SYN-6461 (error) Specified II %d for %s can not be met. Achievable II = %d.

DESCRIPTION

The Synphony C Compiler tool uses the initiation interval (II) specified by you as a constraint to schedule the operations inside a loop. In some cases, it may not be possible for the tool to satisfy this constraint and schedule the operations. In those cases, the tool issues this error and relaxes this constraint in order to generate the RTL.

WHAT NEXT

Use report_feedback_path.txt or the Feedback Path Viewer in GUI to understand feedback path. Make necessary modifications to avoid the feedback path.

SYN-6482 (error) The '%s' functional unit (%s) has an estimated delay of %g ns (including muxing, interconnect, register setup/delay etc.). Conservative timing estimation does not satisfy the required clock period of %g ns at II %d. Details: %s.

DESCRIPTION

When your design's clock frequency is pushing the limits of a technology library, slow functional units (such as multipliers or dividers), can have an estimated delay that is higher than the available clock period. In such cases, the Synphony C Compiler tool generates the RTL with very aggressive timing estimates, and the backend tools may not meet timing.

WHAT NEXT

Lower the clock frequency or adding bitsize pragmas to avoid the timing issue. For example, if following function is synthesized at 1000 MHz with a specific technology library where 32-bit multiplier unit's estimated delay is 1.394ns, then the tool cannot fit this functional unit in 1ns clock period (1000 MHz).

If the inputs a and b were bitsized to values lower than 32-bits, then the estimate delay for the multiplier reduces.

```
unsigned long long calculate (unsigned int a, unsigned int b) {
  return (a*b);
}
```

SYN-6484 Synthesis has run out of memory. This is most likely because of the size of the design. Please try partitioning the design using TCABs.

DESCRIPTION

This message is issued when the design is too large to be run at once in memory.

WHAT NEXT

You can do either of the following to get past this issue:

- If you were using the 32-bit version of Synphony C Compiler, rerun the design with the 64-bit version of the tool. This can be done by invoking the tool without the -32bit option.

- Partition your design into smaller functions. Such lower level functions can be built independently as TCABs or PPAs and then imported at the build of your current design. Please see the User Guide for more detailed information.

SYN-6486 (warning) The '%s' functional unit (%s) has an estimated delay of %g ns (including muxing, interconnect, register setup/delay etc.). Conservative timing estimation does not satisfy the required clock period of %g ns at II %d. Details: %s.

DESCRIPTION

When your design's clock frequency is pushing the limits of a technology library, slow functional units (such as multipliers or dividers), can have an estimated delay that is higher than available clock period. In such cases, the Synphony C Compiler tool generates the RTL with very aggressive timing estimates, and the backend tools might not meet timing.

WHAT NEXT

Lower the clock frequency or adding bitsize pragmas to avoid the timing issue. For example, if following function is synthesized at 1000 MHz with a specific technology library where 32-bit multiplier unit's estimated delay is 1.394ns, then the tool cannot fit this functional unit in 1ns clock period (1000 MHz). If the inputs a and b were bitsized to values lower than 32-bits, then the estimate delay for the multiplier reduces.

```
unsigned long long calculate (unsigned int a, unsigned int b) {
  return (a*b);
}
```

SYN-6538 (warning) Since the TCAB is single-cycle, shared stall domain unit is being synthesized in spite of the user specification to force independent stall domain unit.

DESCRIPTION

A TCAB can be synthesized as a shared stall domain or independent stall domain. Typically, the Synphony C Compiler tool creates a shared stall domain TCAB if the pipelinability constraint is achievable. The tool generates an independent stall domain TCAB under the following conditions:

- if your specified pipelinability constraints are not achievable,
- if the TCAB has an unknown number of iterations, or
- if you explicitly specify the **set_params -force_independent_stalldomain_tcab yes** constraint.

If your TCAB can be scheduled in a single cycle (e.g. it is combinational), there is no performance benefits of an independent stall domain and hence a shared stall domain is always created.

WHAT NEXT

Ignore the warning message or remove the **set_params -force_independent_stalldomain_tcab yes** constraint.

SYN-6549 (error) The Synphony C Compiler tool was not able to synthesize a non-stalling TCAB with the specified architectural constraints.

DESCRIPTION

WHAT NEXT

The failing path/feedback paths are available in report_feedback_path.txt and can be visualized in the Feedback Path Viewer in the GUI.

If all cycles/paths in the report_feedback_path.txt are reported with a positive slack, investigate the first few cycles/paths with least slack, and try to eliminate or increase the slack of these feedback paths by modifying the application code. You could also try synthesizing the TCAB by setting the **allow_latency_violation** parameter to *yes* or by increasing the **output_production_time**, **resource_last_use_time**, and/or **architectural_pipelinability** parameters.

SYN-6648 (warning) Loop %d doesn't meet timing at frequency %d MHz, trying %d MHz.

DESCRIPTION

When the **build** command cannot meet the target frequency for a particular loop, the Synphony C Compiler tool attempts to generate RTL by lowering the frequency.

WHAT NEXT

Use the report_feedback_path.txt file or the Feedback Path Viewer in the GUI to understand the feedback path that prevented the tool to achieve the frequency.

SYN-6649 (error) Loop %d failed user specified TASK II/II. Failing II is %d, Minimum achievable II=%d

DESCRIPTION

The II for a loop could not be achieved. The II is specified by the **ii** parameter or inferred by the Synphony C Compiler tool during the **build** command.

The tool might relax this constraint and try to generate RTL by increasing II. The achieved II is shown in the report_summary.txt file and the Unified Report in the GUI.

WHAT NEXT

Use the report_feedback_path.txt file or the Feedback Path Viewer in the GUI to understand the feedback path that prevents the tool to achieve II.

SYN-6704 (warning) Your design has scalar that is both an input and an output to the design. Such designs can't have task overlap.

DESCRIPTION

If a scalar I/O that is both input and output to the design, the Synphony C Compiler tool cannot generate hardware that starts the next task with different value at the input while the previous task is in progress.

WHAT NEXT

To prevent this warning message, remove the **set_params -task_overlap always** design constraint or change your design so that all scalars I/Os are either input or output, but not both.

SYN-6707 (error) No arbitration constraint failed. %s ports on the memory for array '%s' (buffer-%d) are %d:%d arbitrated because the user specified porting requirement of %d could not be achieved otherwise. This could degrade dynamic performance.

DESCRIPTION

This error message is issued when the tool cannot meet the specified performance and porting constraints without instantiating an arbitrator. You can specify the memory porting constraints using TCL commands or the **read_write_ports** pragma:
Specifying memory porting constraints using TCL commands:

```
  set_params -
internal_blockram_memory_read_write_ports separate | combined | no_arbi
tration | separate,no_arbitration | combined,no_arbitration]
  set_params -
user_supplied_memory_read_write_ports separate | combined | no_arbitrat
ion | separate,no_arbitration | combined,no_arbitration
  set_params -
user_supplied_fpga_memory_read_write_ports separate | combined | no_arb
itration | separate,no_arbitration | combined,no_arbitration
```

Specifying memory porting constraints using **read_write_ports** pragma:

```
  #pragma read_write_ports arrayName combined [n] [no_arbitration]
  #pragma read_write_ports arrayName separate [n] [r readonly [w writeo
nly]] [no_arbitration]
```

By default, the Synphony C Compiler tool instantiates an arbitrator if the number of simultaneous accesses is greater than the number of ports specified by the **read_write_ports** pragma. When you specify the **no_arbitration** constraint with the **read_write_ports** constraint, the tool attempts to synthesize the design without instantiating an arbitrator. If the tool cannot synthesize the design without an arbitrator, even after relaxing other possible constraints, the tool issues this error message.

WHAT NEXT

Analyze your design to see if an arbitrator was instantiated because of the tool's conservative analysis. Rewrite your code in a more analyzable way.

SYN-6718 (error) PA%d failed to meet timing at frequency %d MHz. Relaxing the frequency to %d MHz.

DESCRIPTION

The Synphony C Compiler tool uses the frequency constraint specified by the **set_params clock_freq** parameter for scheduling operations in a PA. This error message is issued when the tool cannot meet the hard frequency constraint. Synphony C Compiler issues this error message, but relaxes the specified constraint to generate the RTL.

WHAT NEXT

Use the report_feedback_path.txt file or Feedback Path Viewer in the GUI to analyze the feedback path causing the failure. Make necessary modifications to avoid the feedback path failure. For more information about the Feedback Path Viewer, see the *Synphony C Compiler User Guide.*