Spyglass nLint

- Verilog Linting Tool

Introduction to Spyglass nLint

- The Spyglass nLint tool is designed for the verification & analysis of digital designs with multiple powerful functions:

What we focus now!

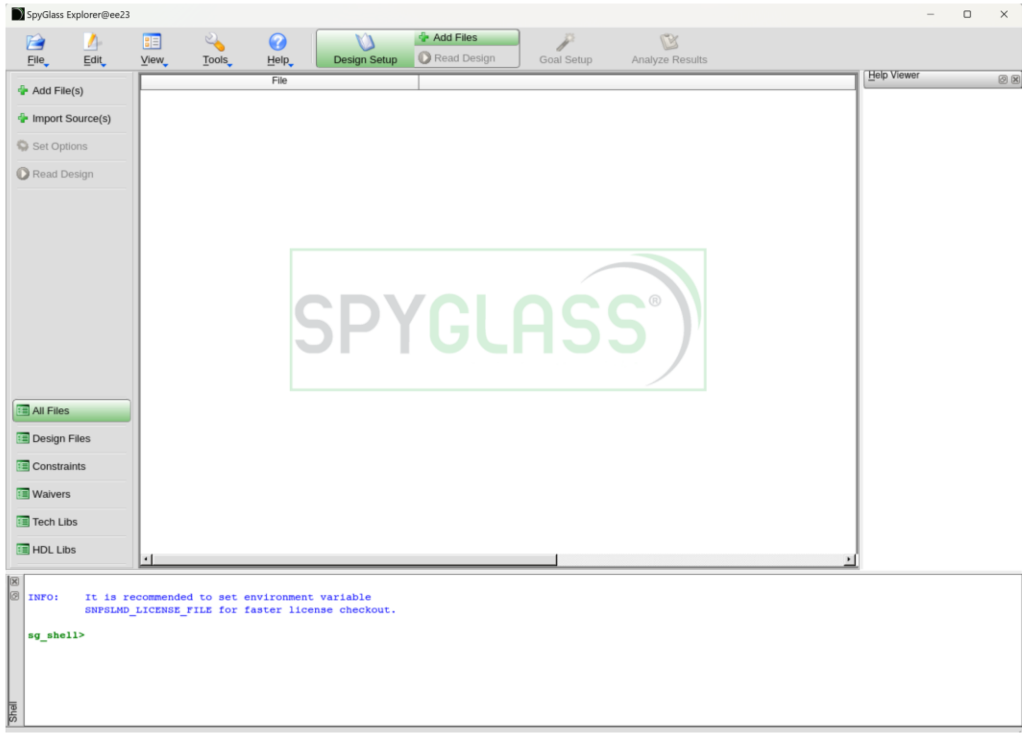
1. Coding Standard Checks in early design stages.
2. Design Rule Checks such as unintentional latches, combinational loops, multiple drivers, etc.
3. Linting for Power, Clock, and CDC Domains.

- Benefits:

1. Early Bug Detection
2. Improved Design Quality

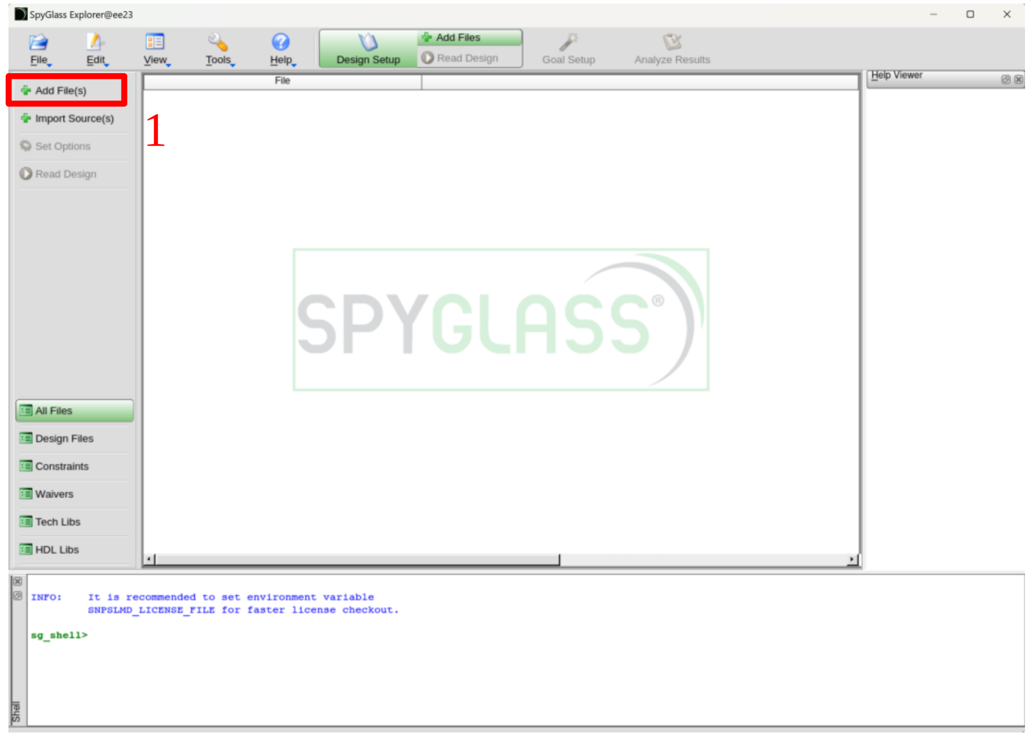
Start Spyglass nLint

- Type the following command on the terminal:
  - spyglass &
  - Also, the token “&” enable you to use the terminal while Spyglass is running in the background.

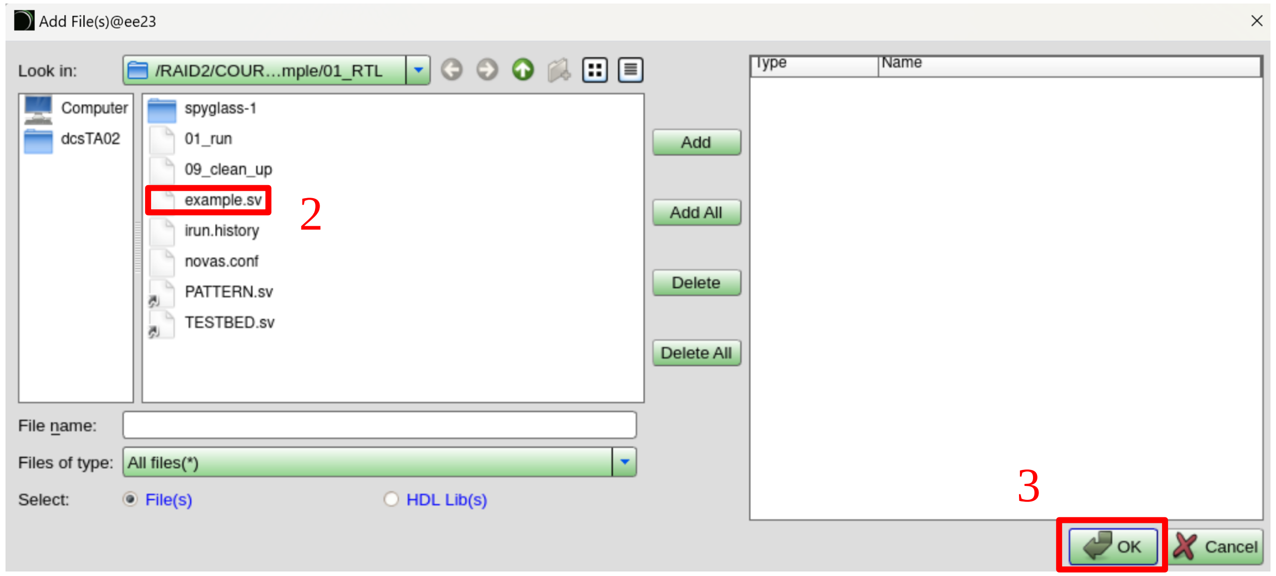


```
INFO:    It is recommended to set environment variable
         SNPSLMD_LICENSE_FILE for faster license checkout.

sg_shell>
```

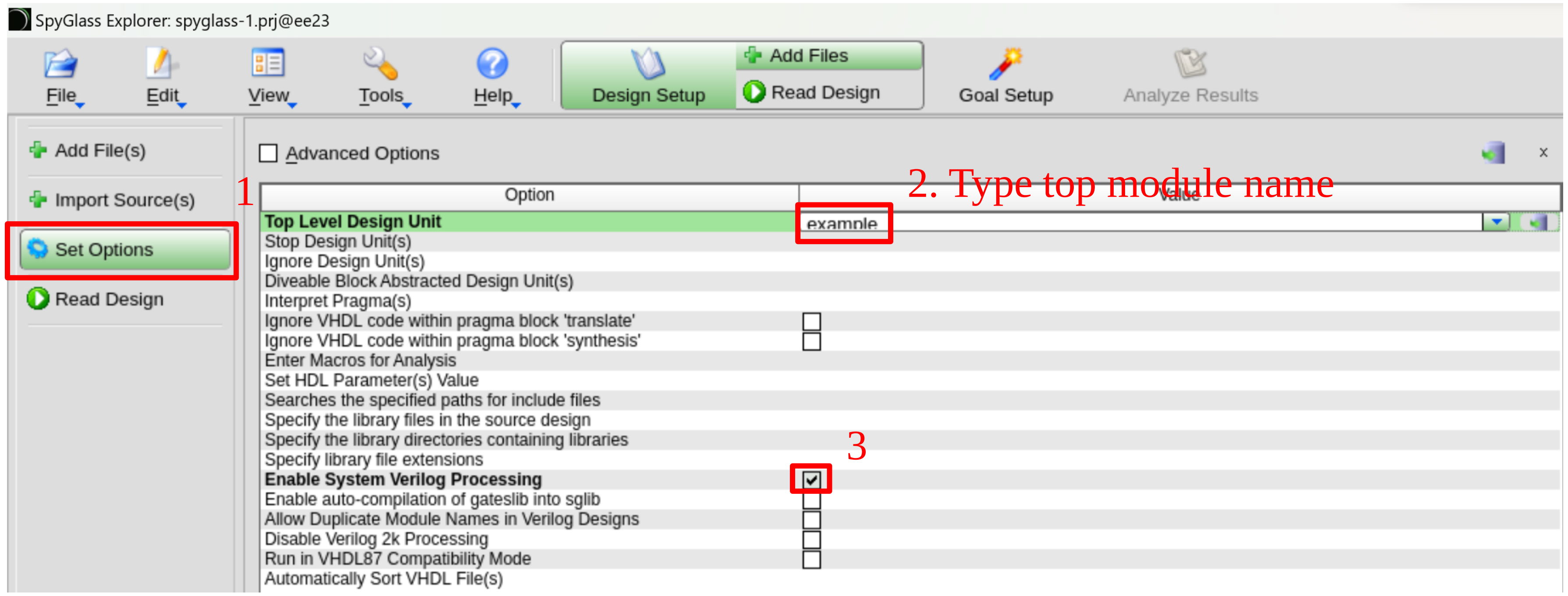
Add files



```
INFO:    It is recommended to set environment variable
         SNPSLMD_LICENSE_FILE for faster license checkout.

sg_shell>
```



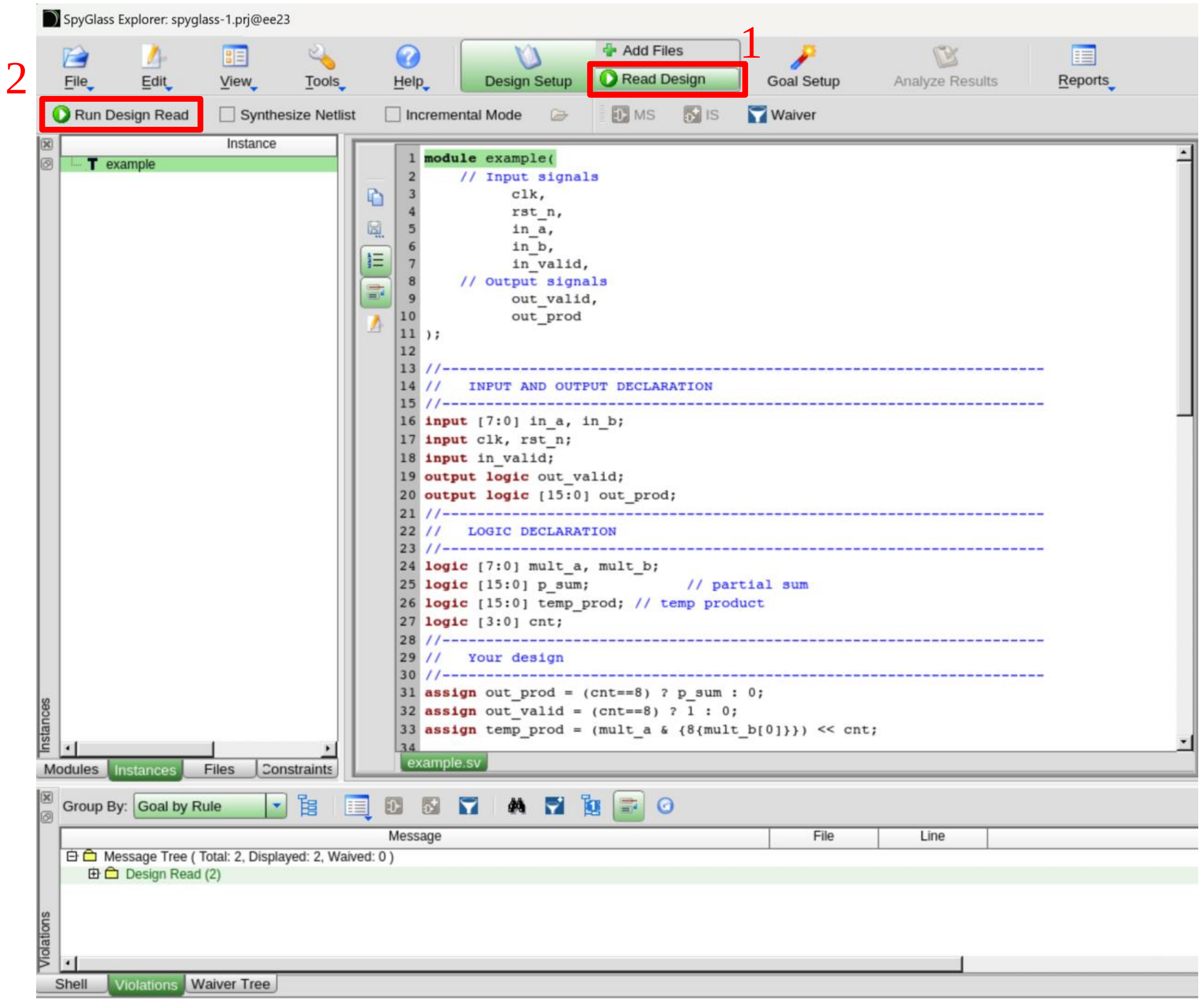
Set Options



Advanced Options

Option	Value
Top Level Design Unit	example
Stop Design Unit(s)	
Ignore Design Unit(s)	
Diveable Block Abstracted Design Unit(s)	
Interpret Pragma(s)	
Ignore VHDL code within pragma block 'translate'	☐
Ignore VHDL code within pragma block 'synthesis'	☐
Enter Macros for Analysis	
Set HDL Parameter(s) Value	
Searches the specified paths for include files	
Specify the library files in the source design	
Specify the library directories containing libraries	
Specify library file extensions	
Enable System Verilog Processing	✓
Enable auto-compilation of gateslib into sglib	☐
Allow Duplicate Module Names in Verilog Designs	☐
Disable Verilog 2k Processing	☐
Run in VHDL87 Compatibility Mode	☐
Automatically Sort VHDL File(s)	

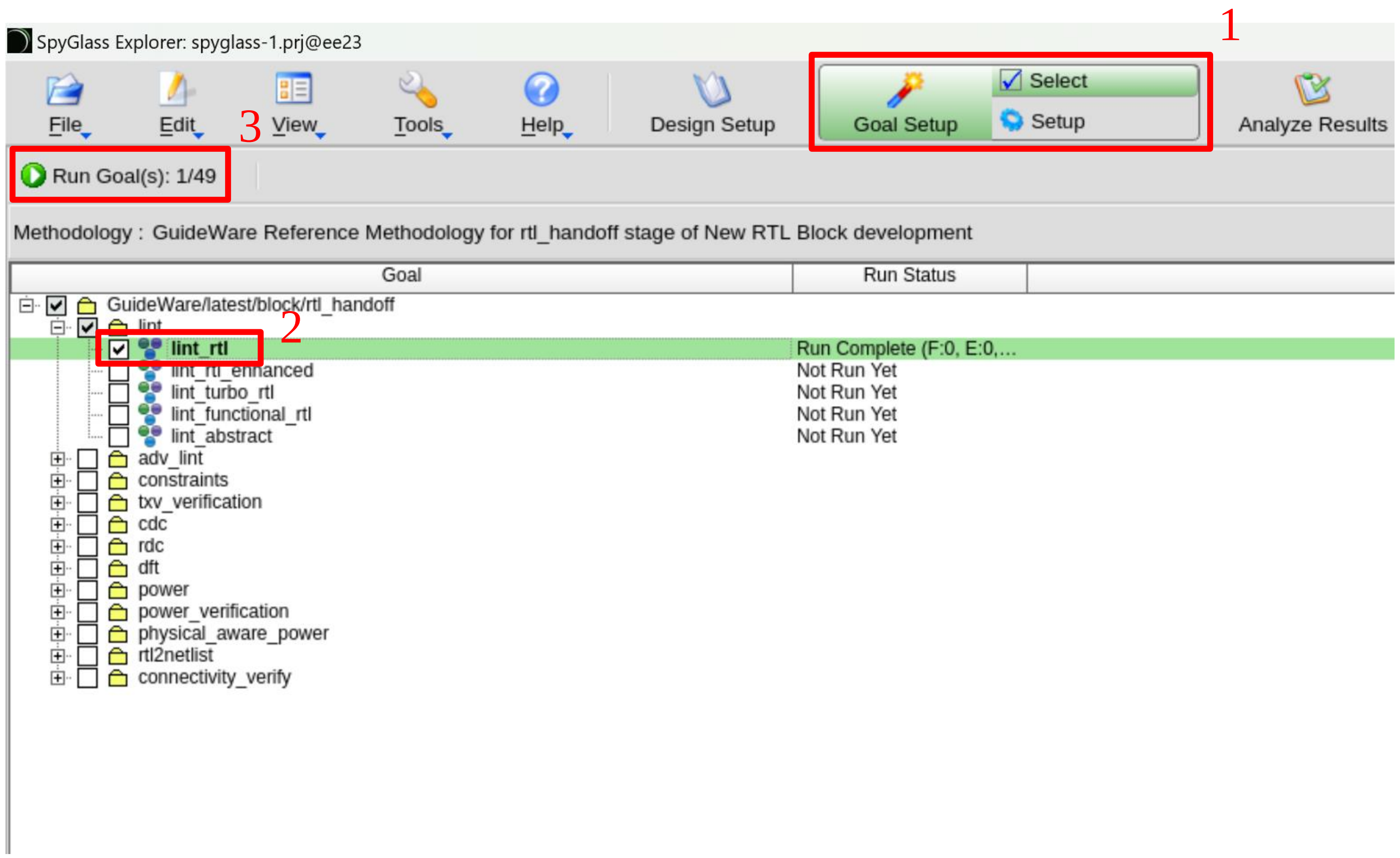
Read Design & Compile



```
module example(
    // Input signals
        clk,
        rst_n,
        in_a,
        in_b,
        in_valid,
    // Output signals
        out_valid,
        out_prod
);

//----------------------------------------------------------------------
//    INPUT AND OUTPUT DECLARATION
//----------------------------------------------------------------------
input [7:0] in_a, in_b;
input clk, rst_n;
input in_valid;
output logic out_valid;
output logic [15:0] out_prod;
//----------------------------------------------------------------------
//    LOGIC DECLARATION
//----------------------------------------------------------------------
logic [7:0] mult_a, mult_b;
logic [15:0] p_sum;           // partial sum
logic [15:0] temp_prod; // temp product
logic [3:0] cnt;
//----------------------------------------------------------------------
//    Your design
//----------------------------------------------------------------------
assign out_prod = (cnt==8) ? p_sum : 0;
assign out_valid = (cnt==8) ? 1 : 0;
assign temp_prod = (mult_a & {8{mult_b[0]}}) << cnt;
```

```
Message Tree ( Total: 2, Displayed: 2, Waived: 0 )
  Design Read (2)
```

Goal Setup



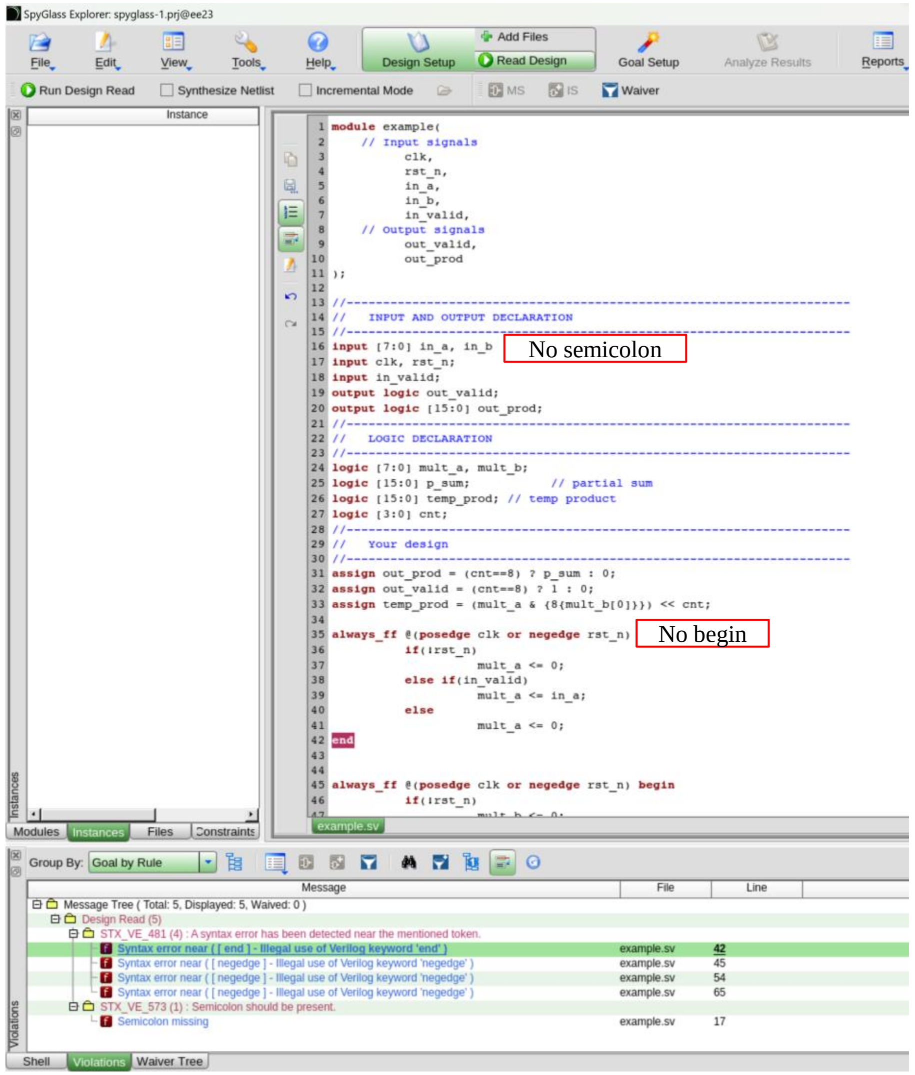
Methodology : GuideWare Reference Methodology for rtl_handoff stage of New RTL Block development

Goal	Run Status
GuideWare/latest/block/rtl_handoff	
lint	
lint_rtl	Run Complete (F:0, E:0,…
lint_rtl_enhanced	Not Run Yet
lint_turbo_rtl	Not Run Yet
lint_functional_rtl	Not Run Yet
lint_abstract	Not Run Yet
adv_lint	
constraints	
txv_verification	
cdc	
rdc	
dft	
power	
power_verification	
physical_aware_power	
rtl2netlist	
connectivity_verify	

Goal : lint/lint_rtl
Checks the design for basic Connectivity issues
Checks the design for basic Simulation issues
Checks the design for recommended design practices and Structural issues
Checks the design for basic Synthesis issues

Description:
This goal checks basic connectivity issues in the design, such as floating input, width mismatch, etc. These checks should be run after every change in RTL code prior to code check-in.
This goal checks simulation issues in the design, such as
- incomplete sensitivity list
- incorrect use of blocking/ non-blocking assignments
- potential functional errors
- possible simulation hang cases, and
- simulation race cases
These checks should be run, and reported messages should be reviewed prior to all simulation runs.
This goal identifies the structural issues in the design that affect the post-implementation functionality or performance of the design. Examples include multiple drivers, high fan-in mux, and synchronous/asynchronous use of resets.
These checks should be run once every week and before handoff to implementation.
This goal reports unsynthesizable constructs in the design and code which can cause RTL vs. gate simulation mismatch. These checks should be run twice a week, and before handoff to synthesis team.

Coding Standard Checks – Basic Syntax Error

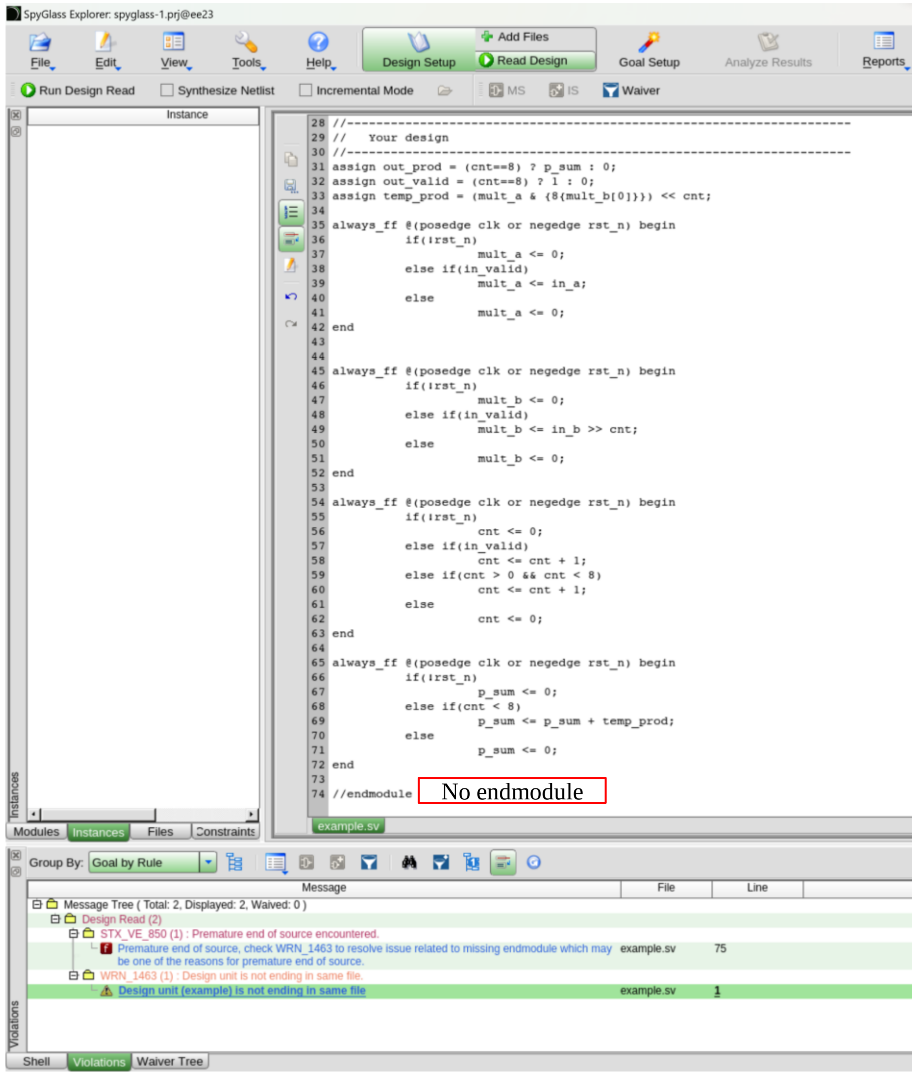


```
 1 module example(
 2     // Input signals
 3             clk,
 4             rst_n,
 5             in_a,
 6             in_b,
 7             in_valid,
 8     // Output signals
 9             out_valid,
10             out_prod
11 );
12 
13 //---------------------------------------------------------------------
14 //    INPUT AND OUTPUT DECLARATION
15 //---------------------------------------------------------------------
16 input [7:0] in_a, in_b
17 input clk, rst_n;
18 input in_valid;
19 output logic out_valid;
20 output logic [15:0] out_prod;
21 //---------------------------------------------------------------------
22 //    LOGIC DECLARATION
23 //---------------------------------------------------------------------
24 logic [7:0] mult_a, mult_b;
25 logic [15:0] p_sum;           // partial sum
26 logic [15:0] temp_prod; // temp product
27 logic [3:0] cnt;
28 //---------------------------------------------------------------------
29 //    Your design
30 //---------------------------------------------------------------------
31 assign out_prod = (cnt==8) ? p_sum : 0;
32 assign out_valid = (cnt==8) ? 1 : 0;
33 assign temp_prod = (mult_a & {8{mult_b[0]}}) << cnt;
34 
35 always_ff @(posedge clk or negedge rst_n)
36           if(!rst_n)
37                   mult_a <= 0;
38           else if(in_valid)
39                   mult_a <= in_a;
40           else
41                   mult_a <= 0;
42 end
43 
44 
45 always_ff @(posedge clk or negedge rst_n) begin
46           if(!rst_n)
```

No semicolon

No begin

```
Message                                                                          File          Line
⊟ Message Tree ( Total: 5, Displayed: 5, Waived: 0 )
  ⊟ Design Read (5)
    ⊟ STX_VE_481 (4) : A syntax error has been detected near the mentioned token.
        Syntax error near ( [ end ] - Illegal use of Verilog keyword 'end' )             example.sv    42
        Syntax error near ( [ negedge ] - Illegal use of Verilog keyword 'negedge' )     example.sv    45
        Syntax error near ( [ negedge ] - Illegal use of Verilog keyword 'negedge' )     example.sv    54
        Syntax error near ( [ negedge ] - Illegal use of Verilog keyword 'negedge' )     example.sv    65
    ⊟ STX_VE_573 (1) : Semicolon should be present.
        Semicolon missing                                                                example.sv    17
```

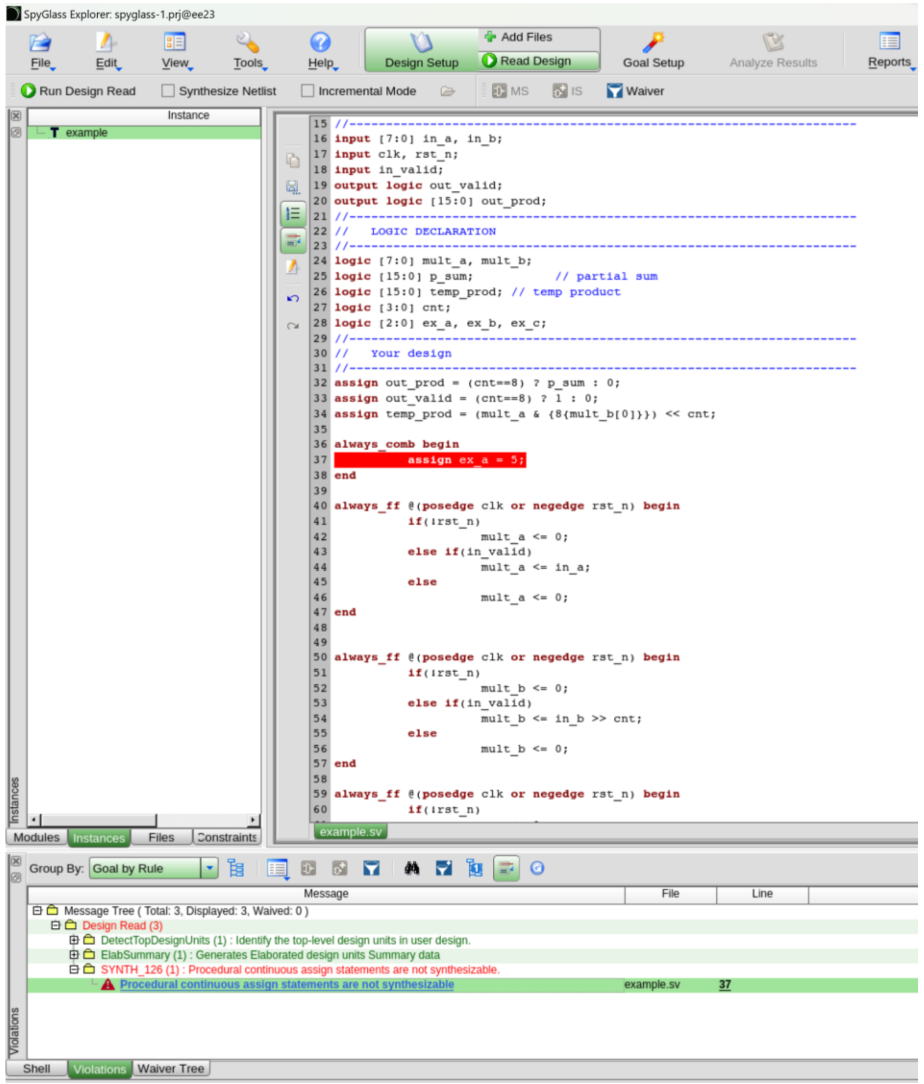


```
28 //---------------------------------------------------------------------
29 //    Your design
30 //---------------------------------------------------------------------
31 assign out_prod = (cnt==8) ? p_sum : 0;
32 assign out_valid = (cnt==8) ? 1 : 0;
33 assign temp_prod = (mult_a & {8{mult_b[0]}}) << cnt;
34 
35 always_ff @(posedge clk or negedge rst_n) begin
36           if(!rst_n)
37                   mult_a <= 0;
38           else if(in_valid)
39                   mult_a <= in_a;
40           else
41                   mult_a <= 0;
42 end
43 
44 
45 always_ff @(posedge clk or negedge rst_n) begin
46           if(!rst_n)
47                   mult_b <= 0;
48           else if(in_valid)
49                   mult_b <= in_b >> cnt;
50           else
51                   mult_b <= 0;
52 end
53 
54 always_ff @(posedge clk or negedge rst_n) begin
55           if(!rst_n)
56                   cnt <= 0;
57           else if(in_valid)
58                   cnt <= cnt + 1;
59           else if(cnt > 0 && cnt < 8)
60                   cnt <= cnt + 1;
61           else
62                   cnt <= 0;
63 end
64 
65 always_ff @(posedge clk or negedge rst_n) begin
66           if(!rst_n)
67                   p_sum <= 0;
68           else if(cnt < 8)
69                   p_sum <= p_sum + temp_prod;
70           else
71                   p_sum <= 0;
72 end
73 
74 //endmodule
```

No endmodule

```
Message                                                                          File          Line
⊟ Message Tree ( Total: 2, Displayed: 2, Waived: 0 )
  ⊟ Design Read (2)
    ⊟ STX_VE_850 (1) : Premature end of source encountered.
        Premature end of source, check WRN_1463 to resolve issue related to missing endmodule which may    example.sv    75
        be one of the reasons for premature end of source.
    ⊟ WRN_1463 (1) : Design unit is not ending in same file.
        Design unit (example) is not ending in same file                                 example.sv    1
```

Design Rule Checks – Combinational Logic

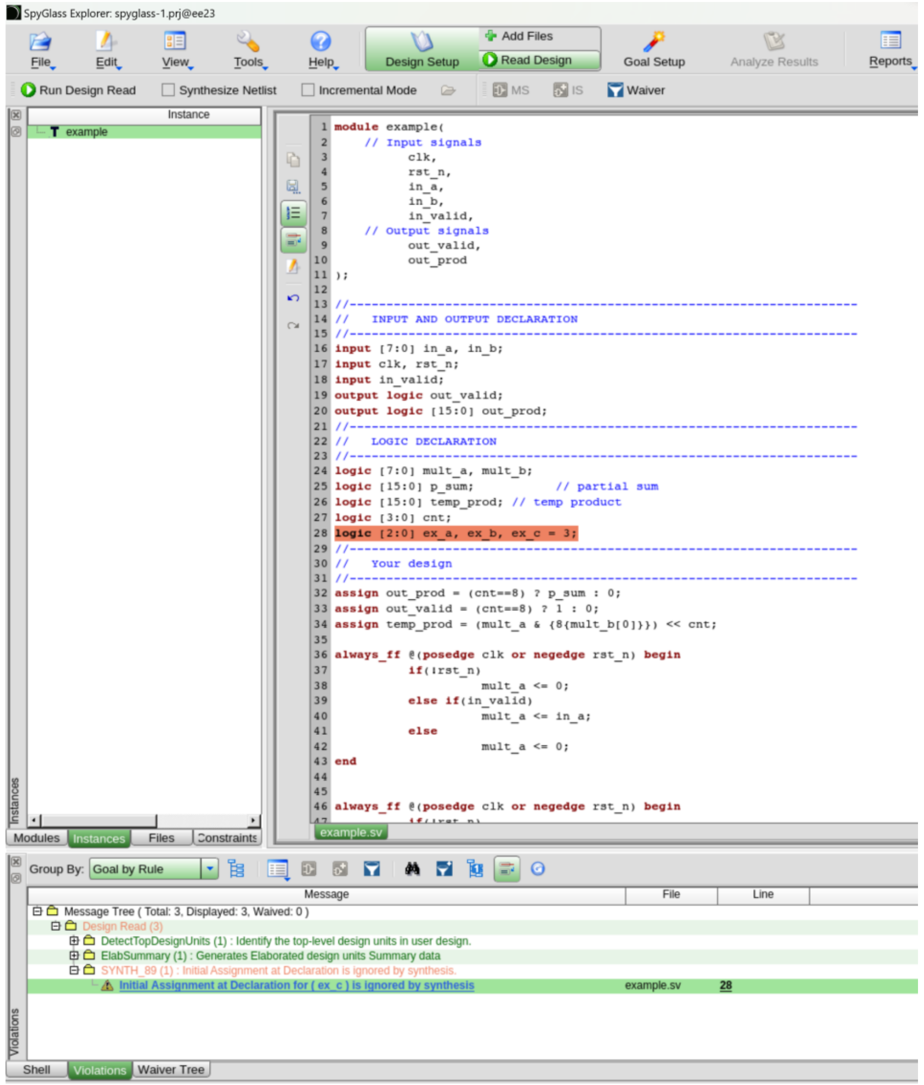


SpyGlass Explorer: spyglass-1.prj@ee23

```
15 //----------------------------------------------------------------------
16 input [7:0] in_a, in_b;
17 input clk, rst_n;
18 input in_valid;
19 output logic out_valid;
20 output logic [15:0] out_prod;
21 //----------------------------------------------------------------------
22 //   LOGIC DECLARATION
23 //----------------------------------------------------------------------
24 logic [7:0] mult_a, mult_b;
25 logic [15:0] p_sum;            // partial sum
26 logic [15:0] temp_prod; // temp product
27 logic [3:0] cnt;
28 logic [2:0] ex_a, ex_b, ex_c;
29 //----------------------------------------------------------------------
30 //   Your design
31 //----------------------------------------------------------------------
32 assign out_prod = (cnt==8) ? p_sum : 0;
33 assign out_valid = (cnt==8) ? 1 : 0;
34 assign temp_prod = (mult_a & {8{mult_b[0]}}) << cnt;
35
36 always_comb begin
37         assign ex_a = 5;
38 end
39
40 always_ff @(posedge clk or negedge rst_n) begin
41         if(!rst_n)
42                 mult_a <= 0;
43         else if(in_valid)
44                 mult_a <= in_a;
45         else
46                 mult_a <= 0;
47 end
48
49
50 always_ff @(posedge clk or negedge rst_n) begin
51         if(!rst_n)
52                 mult_b <= 0;
53         else if(in_valid)
54                 mult_b <= in_b >> cnt;
55         else
56                 mult_b <= 0;
57 end
58
59 always_ff @(posedge clk or negedge rst_n) begin
60         if(!rst_n)
```

```
Message                                                                                   File         Line
⊟ Message Tree ( Total: 3, Displayed: 3, Waived: 0 )
  ⊟ Design Read (3)
    ⊞ DetectTopDesignUnits (1) : Identify the top-level design units in user design.
    ⊞ ElabSummary (1) : Generates Elaborated design units Summary data
    ⊟ SYNTH_126 (1) : Procedural continuous assign statements are not synthesizable.
      Procedural continuous assign statements are not synthesizable                  example.sv   37
```

No assign within always block (legal but not synthesizable)



SpyGlass Explorer: spyglass-1.prj@ee23

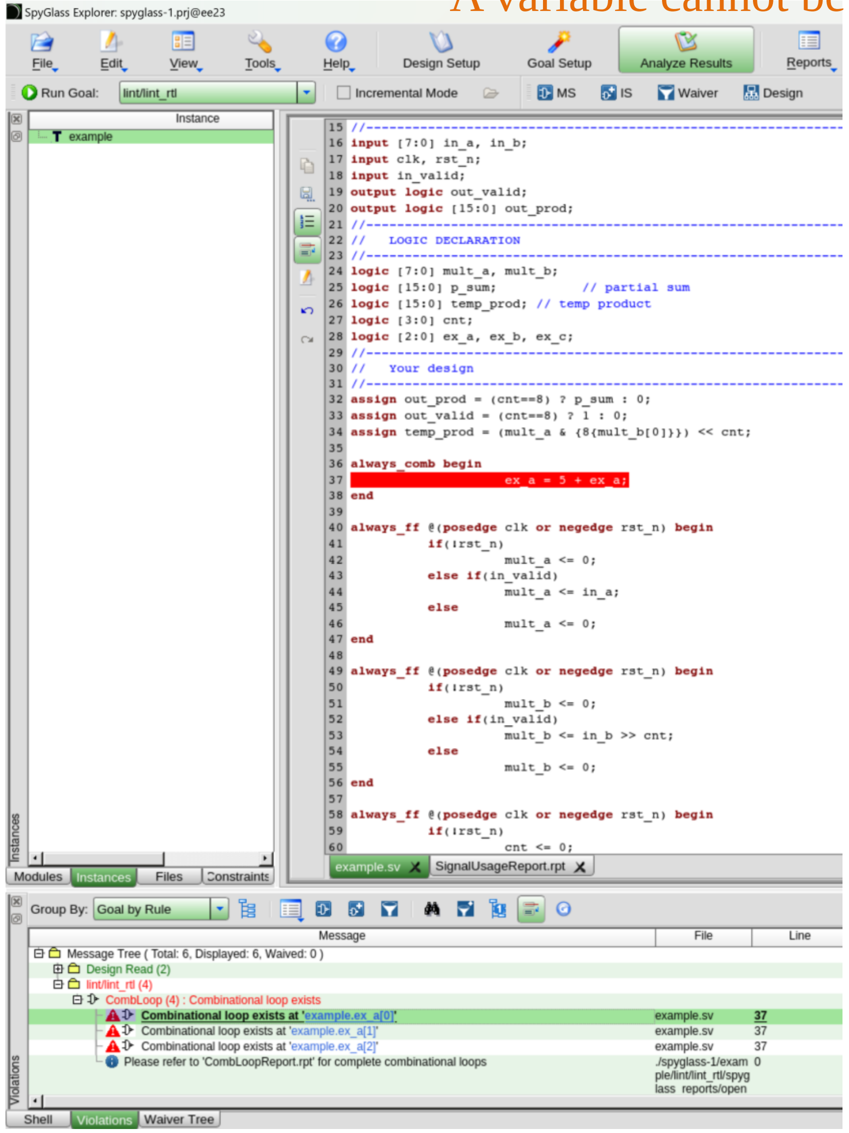
```
1  module example(
2      // Input signals
3          clk,
4          rst_n,
5          in_a,
6          in_b,
7          in_valid,
8      // Output signals
9          out_valid,
10         out_prod
11 );
12
13 //----------------------------------------------------------------------
14 //   INPUT AND OUTPUT DECLARATION
15 //----------------------------------------------------------------------
16 input [7:0] in_a, in_b;
17 input clk, rst_n;
18 input in_valid;
19 output logic out_valid;
20 output logic [15:0] out_prod;
21 //----------------------------------------------------------------------
22 //   LOGIC DECLARATION
23 //----------------------------------------------------------------------
24 logic [7:0] mult_a, mult_b;
25 logic [15:0] p_sum;            // partial sum
26 logic [15:0] temp_prod; // temp product
27 logic [3:0] cnt;
28 logic [2:0] ex_a, ex_b, ex_c = 3;
29 //----------------------------------------------------------------------
30 //   Your design
31 //----------------------------------------------------------------------
32 assign out_prod = (cnt==8) ? p_sum : 0;
33 assign out_valid = (cnt==8) ? 1 : 0;
34 assign temp_prod = (mult_a & {8{mult_b[0]}}) << cnt;
35
36 always_ff @(posedge clk or negedge rst_n) begin
37         if(!rst_n)
38                 mult_a <= 0;
39         else if(in_valid)
40                 mult_a <= in_a;
41         else
42                 mult_a <= 0;
43 end
44
45
46 always_ff @(posedge clk or negedge rst_n) begin
```

```
Message                                                                                   File         Line
⊟ Message Tree ( Total: 3, Displayed: 3, Waived: 0 )
  ⊟ Design Read (3)
    ⊞ DetectTopDesignUnits (1) : Identify the top-level design units in user design.
    ⊞ ElabSummary (1) : Generates Elaborated design units Summary data
    ⊟ SYNTH_89 (1) : Initial Assignment at Declaration is ignored by synthesis.
      Initial Assignment at Declaration for ( ex_c ) is ignored by synthesis         example.sv   28
```

Initial value for combinational logic will be ignored for synthesis

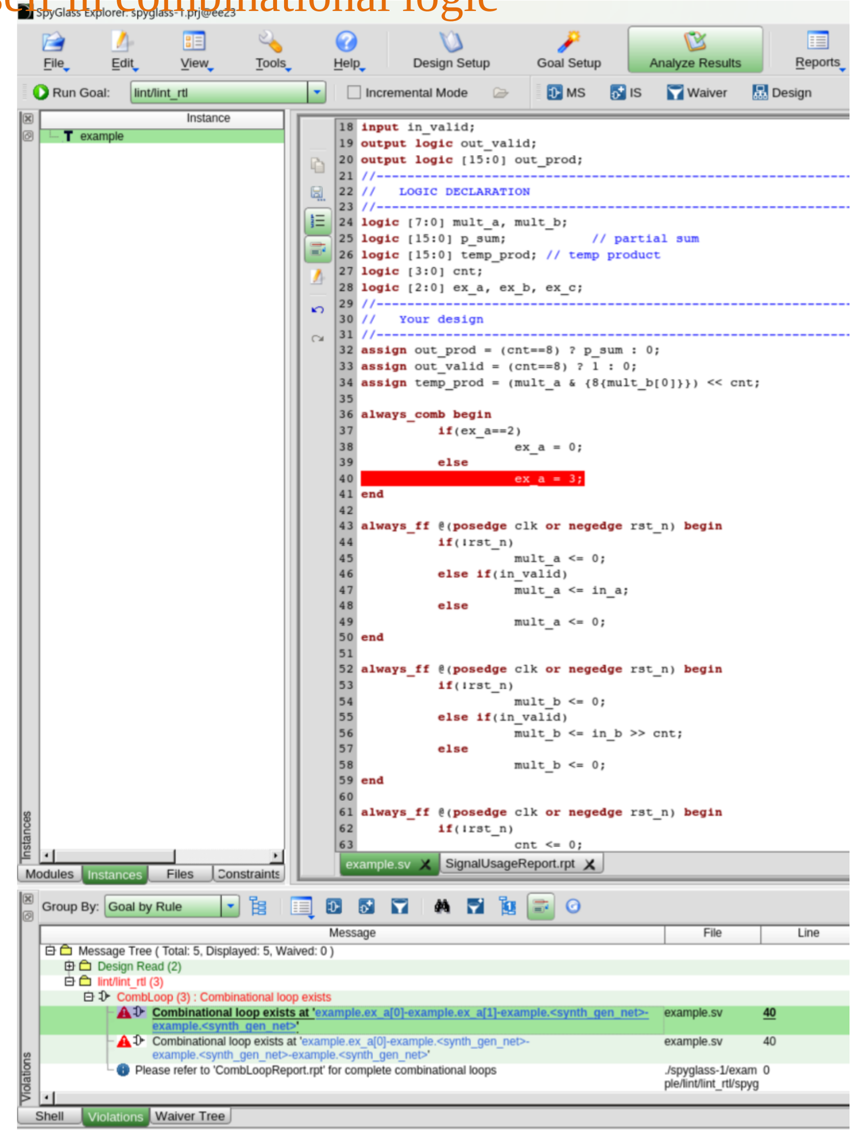
Design Rule Checks – Combinational Loop

A variable cannot be assigned by itself in combinational logic



```
15 //-----------------------------------------------------------------
16 input [7:0] in_a, in_b;
17 input clk, rst_n;
18 input in_valid;
19 output logic out_valid;
20 output logic [15:0] out_prod;
21 //-----------------------------------------------------------------
22 //   LOGIC DECLARATION
23 //-----------------------------------------------------------------
24 logic [7:0] mult_a, mult_b;
25 logic [15:0] p_sum;             // partial sum
26 logic [15:0] temp_prod; // temp product
27 logic [3:0] cnt;
28 logic [2:0] ex_a, ex_b, ex_c;
29 //-----------------------------------------------------------------
30 //   Your design
31 //-----------------------------------------------------------------
32 assign out_prod = (cnt==8) ? p_sum : 0;
33 assign out_valid = (cnt==8) ? 1 : 0;
34 assign temp_prod = (mult_a & {8{mult_b[0]}}) << cnt;
35
36 always_comb begin
37                 ex_a = 5 + ex_a;
38 end
39
40 always_ff @(posedge clk or negedge rst_n) begin
41         if(!rst_n)
42                 mult_a <= 0;
43         else if(in_valid)
44                 mult_a <= in_a;
45         else
46                 mult_a <= 0;
47 end
48
49 always_ff @(posedge clk or negedge rst_n) begin
50         if(!rst_n)
51                 mult_b <= 0;
52         else if(in_valid)
53                 mult_b <= in_b >> cnt;
54         else
55                 mult_b <= 0;
56 end
57
58 always_ff @(posedge clk or negedge rst_n) begin
59         if(!rst_n)
60                 cnt <= 0;
```

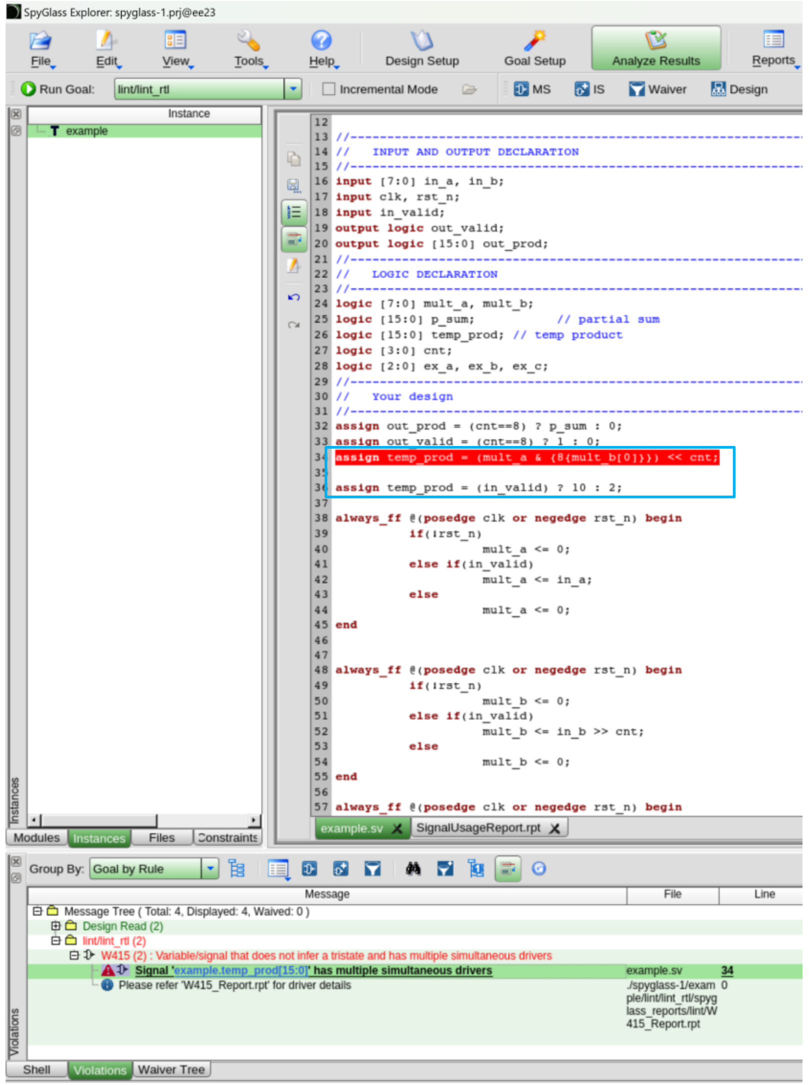
Message	File	Line
Message Tree (Total: 6, Displayed: 6, Waived: 0)		
Design Read (2)		
lint/lint_rtl (4)		
CombLoop (4) : Combinational loop exists		
Combinational loop exists at 'example.ex_a[0]'	example.sv	**37**
Combinational loop exists at 'example.ex_a[1]'	example.sv	37
Combinational loop exists at 'example.ex_a[2]'	example.sv	37
Please refer to 'CombLoopReport.rpt' for complete combinational loops	./spyglass-1/example/lint/lint_rtl/spyglass_reports/open	0



```
18 input in_valid;
19 output logic out_valid;
20 output logic [15:0] out_prod;
21 //-----------------------------------------------------------------
22 //   LOGIC DECLARATION
23 //-----------------------------------------------------------------
24 logic [7:0] mult_a, mult_b;
25 logic [15:0] p_sum;             // partial sum
26 logic [15:0] temp_prod; // temp product
27 logic [3:0] cnt;
28 logic [2:0] ex_a, ex_b, ex_c;
29 //-----------------------------------------------------------------
30 //   Your design
31 //-----------------------------------------------------------------
32 assign out_prod = (cnt==8) ? p_sum : 0;
33 assign out_valid = (cnt==8) ? 1 : 0;
34 assign temp_prod = (mult_a & {8{mult_b[0]}}) << cnt;
35
36 always_comb begin
37         if(ex_a==2)
38                 ex_a = 0;
39         else
40                 ex_a = 3;
41 end
42
43 always_ff @(posedge clk or negedge rst_n) begin
44         if(!rst_n)
45                 mult_a <= 0;
46         else if(in_valid)
47                 mult_a <= in_a;
48         else
49                 mult_a <= 0;
50 end
51
52 always_ff @(posedge clk or negedge rst_n) begin
53         if(!rst_n)
54                 mult_b <= 0;
55         else if(in_valid)
56                 mult_b <= in_b >> cnt;
57         else
58                 mult_b <= 0;
59 end
60
61 always_ff @(posedge clk or negedge rst_n) begin
62         if(!rst_n)
63                 cnt <= 0;
```

Message	File	Line
Message Tree (Total: 5, Displayed: 5, Waived: 0)		
Design Read (2)		
lint/lint_rtl (3)		
CombLoop (3) : Combinational loop exists		
Combinational loop exists at 'example.ex_a[0]-example.ex_a[1]-example.<synth_gen_net>-example.<synth_gen_net>'	example.sv	**40**
Combinational loop exists at 'example.ex_a[0]-example.<synth_gen_net>-example.<synth_gen_net>-example.<synth_gen_net>'	example.sv	40
Please refer to 'CombLoopReport.rpt' for complete combinational loops	./spyglass-1/example/lint/lint_rtl/spyg	0

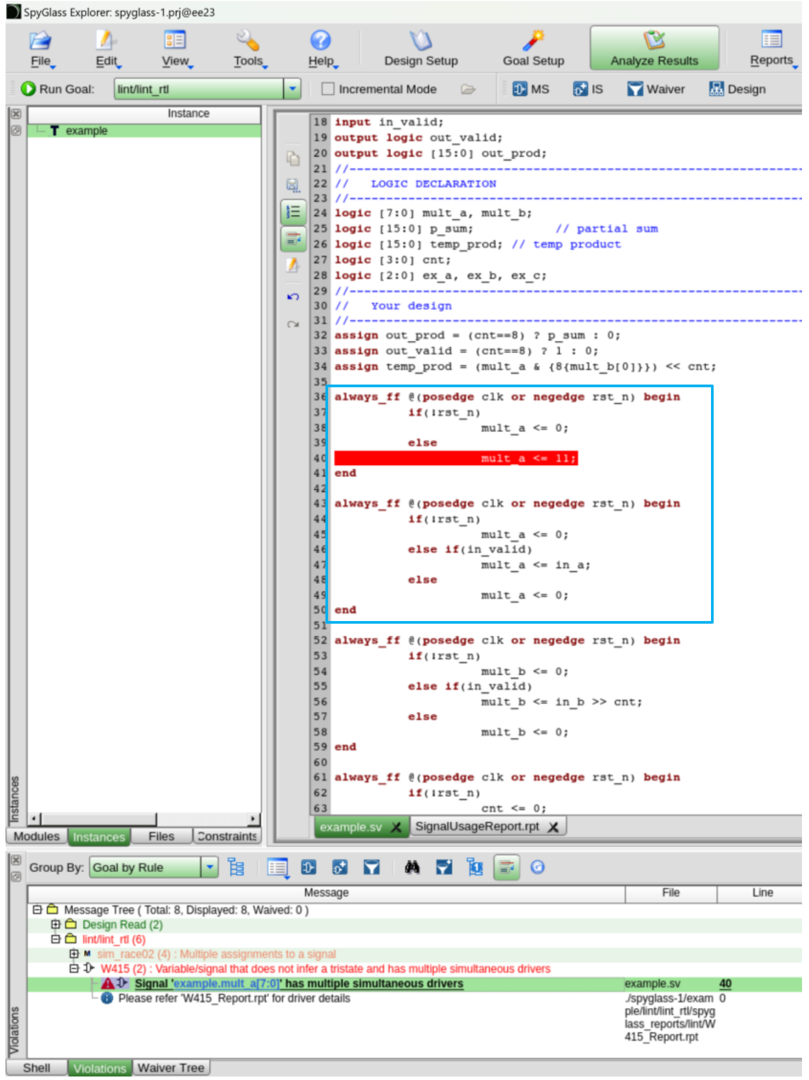
Design Rule Checks – Multiple Drives



```
12
13 //------------------------------------------------------------------
14 //   INPUT AND OUTPUT DECLARATION
15 //------------------------------------------------------------------
16 input [7:0] in_a, in_b;
17 input clk, rst_n;
18 input in_valid;
19 output logic out_valid;
20 output logic [15:0] out_prod;
21 //------------------------------------------------------------------
22 //   LOGIC DECLARATION
23 //------------------------------------------------------------------
24 logic [7:0] mult_a, mult_b;
25 logic [15:0] p_sum;            // partial sum
26 logic [15:0] temp_prod; // temp product
27 logic [3:0] cnt;
28 logic [2:0] ex_a, ex_b, ex_c;
29 //------------------------------------------------------------------
30 //   Your design
31 //------------------------------------------------------------------
32 assign out_prod = (cnt==8) ? p_sum : 0;
33 assign out_valid = (cnt==8) ? 1 : 0;
34 assign temp_prod = (mult_a & {8{mult_b[0]}}) << cnt;
35
36 assign temp_prod = (in_valid) ? 10 : 2;
37
38 always_ff @(posedge clk or negedge rst_n) begin
39         if(!rst_n)
40                 mult_a <= 0;
41         else if(in_valid)
42                 mult_a <= in_a;
43         else
44                 mult_a <= 0;
45 end
46
47
48 always_ff @(posedge clk or negedge rst_n) begin
49         if(!rst_n)
50                 mult_b <= 0;
51         else if(in_valid)
52                 mult_b <= in_b >> cnt;
53         else
54                 mult_b <= 0;
55 end
56
57 always_ff @(posedge clk or negedge rst_n) begin
```

Group By: Goal by Rule

Message	File	Line
Message Tree (Total: 4, Displayed: 4, Waived: 0)		
Design Read (2)		
lint/lint_rtl (2)		
W415 (2) : Variable/signal that does not infer a tristate and has multiple simultaneous drivers		
Signal 'example.temp_prod[15:0]' has multiple simultaneous drivers	example.sv	34
Please refer 'W415_Report.rpt' for driver details	./spyglass-1/example/lint/lint_rtl/spyglass_reports/lint/W415_Report.rpt	0



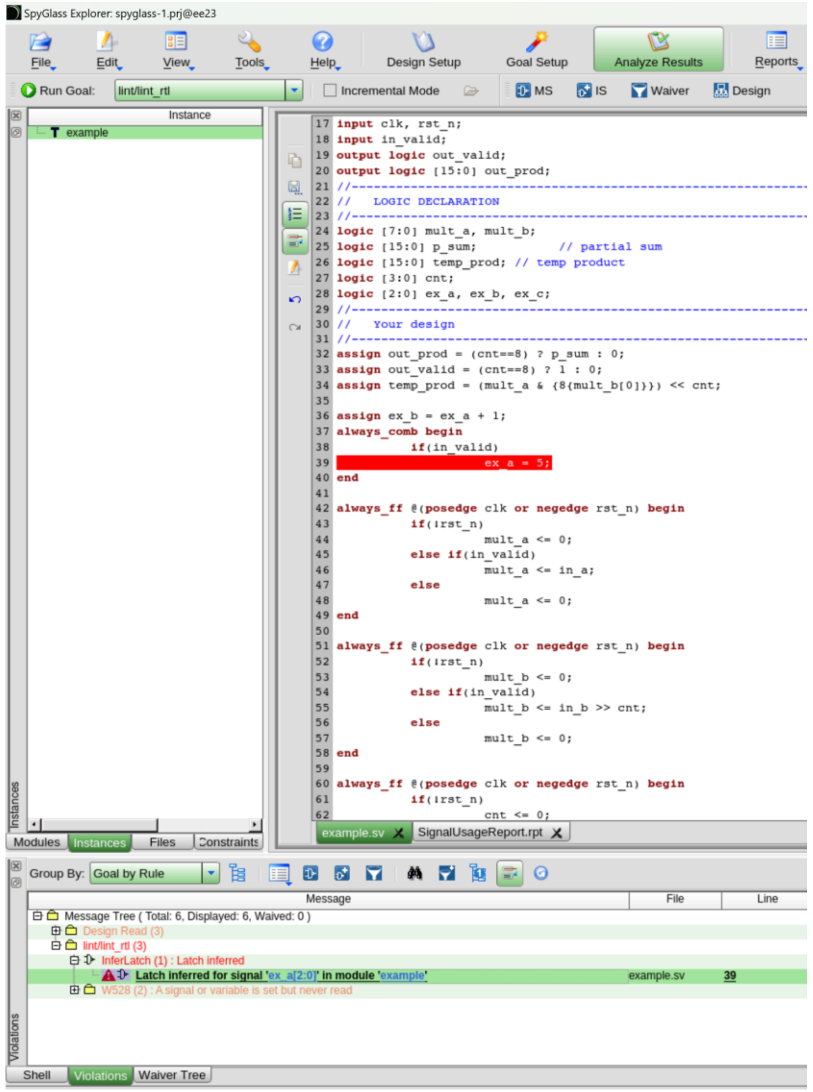
```
18 input in_valid;
19 output logic out_valid;
20 output logic [15:0] out_prod;
21 //------------------------------------------------------------------
22 //   LOGIC DECLARATION
23 //------------------------------------------------------------------
24 logic [7:0] mult_a, mult_b;
25 logic [15:0] p_sum;            // partial sum
26 logic [15:0] temp_prod; // temp product
27 logic [3:0] cnt;
28 logic [2:0] ex_a, ex_b, ex_c;
29 //------------------------------------------------------------------
30 //   Your design
31 //------------------------------------------------------------------
32 assign out_prod = (cnt==8) ? p_sum : 0;
33 assign out_valid = (cnt==8) ? 1 : 0;
34 assign temp_prod = (mult_a & {8{mult_b[0]}}) << cnt;
35
36 always_ff @(posedge clk or negedge rst_n) begin
37         if(!rst_n)
38                 mult_a <= 0;
39         else
40                 mult_a <= 11;
41 end
42
43 always_ff @(posedge clk or negedge rst_n) begin
44         if(!rst_n)
45                 mult_a <= 0;
46         else if(in_valid)
47                 mult_a <= in_a;
48         else
49                 mult_a <= 0;
50 end
51
52 always_ff @(posedge clk or negedge rst_n) begin
53         if(!rst_n)
54                 mult_b <= 0;
55         else if(in_valid)
56                 mult_b <= in_b >> cnt;
57         else
58                 mult_b <= 0;
59 end
60
61 always_ff @(posedge clk or negedge rst_n) begin
62         if(!rst_n)
63                 cnt <= 0;
```

Group By: Goal by Rule

Message	File	Line
Message Tree (Total: 8, Displayed: 8, Waived: 0)		
Design Read (2)		
lint/lint_rtl (6)		
sim_race02 (4) : Multiple assignments to a signal		
W415 (2) : Variable/signal that does not infer a tristate and has multiple simultaneous drivers		
Signal 'example.mult_a[7:0]' has multiple simultaneous drivers	example.sv	40
Please refer 'W415_Report.rpt' for driver details	./spyglass-1/example/lint/lint_rtl/spyglass_reports/lint/W415_Report.rpt	0

One variable can only be assigned in one always block or in one assign

Design Rule Checks – Unintentional latch


SpyGlass Explorer: spyglass-1.prj@ee23

Run Goal: lint/lint_rtl

Instance: example

```
17 input clk, rst_n;
18 input in_valid;
19 output logic out_valid;
20 output logic [15:0] out_prod;
21 //------------------------------------------------------------------
22 //   LOGIC DECLARATION
23 //------------------------------------------------------------------
24 logic [7:0] mult_a, mult_b;
25 logic [15:0] p_sum;           // partial sum
26 logic [15:0] temp_prod; // temp product
27 logic [3:0] cnt;
28 logic [2:0] ex_a, ex_b, ex_c;
29 //------------------------------------------------------------------
30 //   Your design
31 //------------------------------------------------------------------
32 assign out_prod = (cnt==8) ? p_sum : 0;
33 assign out_valid = (cnt==8) ? 1 : 0;
34 assign temp_prod = (mult_a & {8{mult_b[0]}}) << cnt;
35
36 assign ex_b = ex_a + 1;
37 always_comb begin
38         if(in_valid)
39                 ex_a = 5;
40 end
41
42 always_ff @(posedge clk or negedge rst_n) begin
43         if(!rst_n)
44                 mult_a <= 0;
45         else if(in_valid)
46                 mult_a <= in_a;
47         else
48                 mult_a <= 0;
49 end
50
51 always_ff @(posedge clk or negedge rst_n) begin
52         if(!rst_n)
53                 mult_b <= 0;
54         else if(in_valid)
55                 mult_b <= in_b >> cnt;
56         else
57                 mult_b <= 0;
58 end
59
60 always_ff @(posedge clk or negedge rst_n) begin
61         if(!rst_n)
62                 cnt <= 0;
```

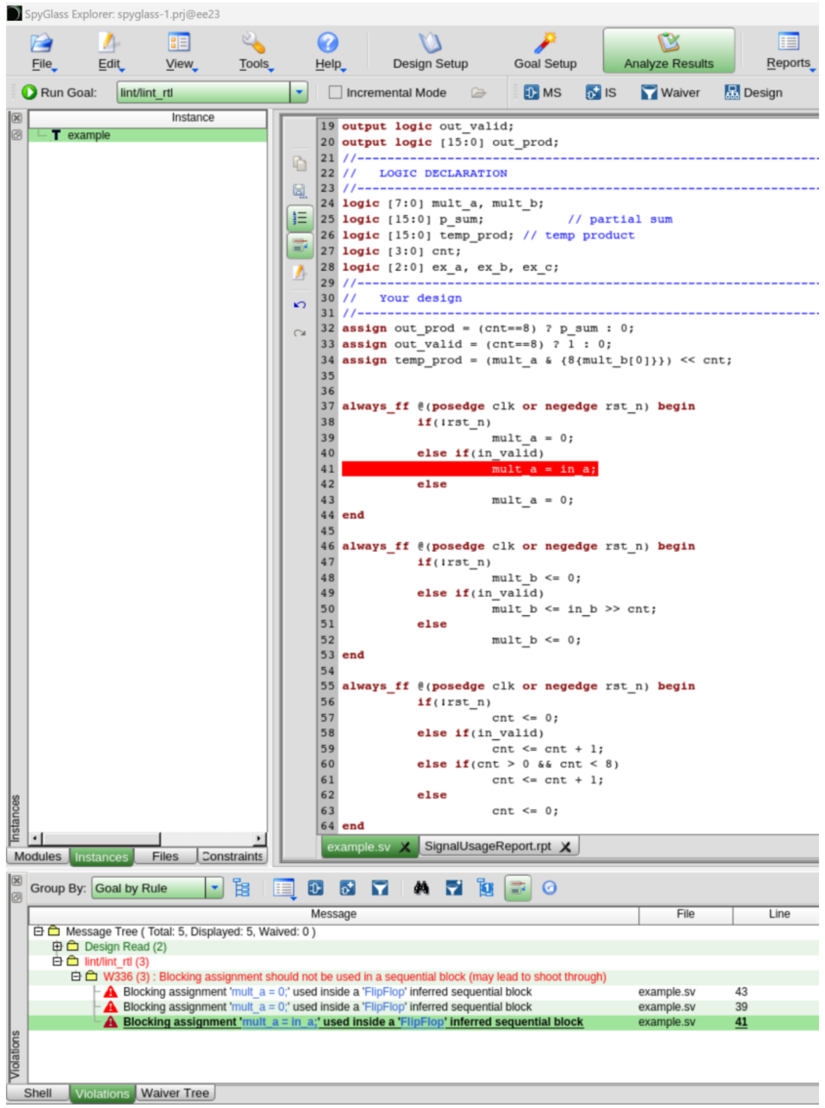
Group By: Goal by Rule

Message	File	Line
Message Tree (Total: 6, Displayed: 6, Waived: 0)		
Design Read (3)		
lint/lint_rtl (3)		
InferLatch (1) : Latch inferred		
Latch inferred for signal 'ex_a[2:0]' in module 'example'	example.sv	39
W528 (2) : A signal or variable is set but never read		

Incomplete conditional statements in combinational logic

- If/else
- Case
- Ternary operator

Design Rule Checks – Blocking assignment in sequential circuit



```
19 output logic out_valid;
20 output logic [15:0] out_prod;
21 //--------------------------------------------------------------
22 //   LOGIC DECLARATION
23 //--------------------------------------------------------------
24 logic [7:0] mult_a, mult_b;
25 logic [15:0] p_sum;            // partial sum
26 logic [15:0] temp_prod; // temp product
27 logic [3:0] cnt;
28 logic [2:0] ex_a, ex_b, ex_c;
29 //--------------------------------------------------------------
30 //   Your design
31 //--------------------------------------------------------------
32 assign out_prod = (cnt==8) ? p_sum : 0;
33 assign out_valid = (cnt==8) ? 1 : 0;
34 assign temp_prod = (mult_a & {8{mult_b[0]}}) << cnt;
35
36
37 always_ff @(posedge clk or negedge rst_n) begin
38           if(!rst_n)
39                     mult_a = 0;
40           else if(in_valid)
41                     mult_a = in_a;
42           else
43                     mult_a = 0;
44 end
45
46 always_ff @(posedge clk or negedge rst_n) begin
47           if(!rst_n)
48                     mult_b <= 0;
49           else if(in_valid)
50                     mult_b <= in_b >> cnt;
51           else
52                     mult_b <= 0;
53 end
54
55 always_ff @(posedge clk or negedge rst_n) begin
56           if(!rst_n)
57                     cnt <= 0;
58           else if(in_valid)
59                     cnt <= cnt + 1;
60           else if(cnt > 0 && cnt < 8)
61                     cnt <= cnt + 1;
62           else
63                     cnt <= 0;
64 end
```

Message	File	Line
Message Tree (Total: 5, Displayed: 5, Waived: 0)		
Design Read (2)		
lint/lint_rtl (3)		
W336 (3) : Blocking assignment should not be used in a sequential block (may lead to shoot through)		
Blocking assignment 'mult_a = 0;' used inside a 'FlipFlop' inferred sequential block	example.sv	43
Blocking assignment 'mult_a = 0;' used inside a 'FlipFlop' inferred sequential block	example.sv	39
Blocking assignment 'mult_a = in_a;' used inside a 'FlipFlop' inferred sequential block	example.sv	**41**

Blocking assignment (for combinational circuit)

- The assignment will be carried consequently.

Non-blocking assignment (for sequential circuit)

- The assignment will be carried in parallel.

Reference

- https://hackmd.io/@qpalzm60409/ryEip2A6h#伍、Spyglass

- https://blog.csdn.net/qq_30843953/article/details/109629618