Lint Rules

Lintrules
Group 3

W414

Reports non-blocking assignments in a combinational block

Lintrules
Group 3

W414

Potential Issues:

Violations may arise when a non-blocking assignment is used in a combinational block

W414

Debug/Fix:

Use only blocking assignments in combinational blocks

Example Code:

```
1  module group3(set, reset, vic, lukas);
2      input set, reset;
3      output vic, lukas;
4      reg vic, lukas;
5
6      always @(posedge vic) begin
7          lukas <= set;
8      end
9
10     always @(set or reset) begin
11         vic <= set & reset; // non-blocking assignments is used in
12         lukas <= set | reset;
13     end
14
15 endmodule
```

Lintrules
Group 3

W446

Output port signal is being read

Potential Issues:

Such modules could lead to an unintended feedback path from the instantiating post-module in the post-synthesis simulation while this issue is not apparent in the pre-synthesis simulation

W446

Debug/Fix:

Such models are not recommended for some test tools that need to handle inout ports

Example Code:

```
1  module group3(
2      input in1,
3      input in2,
4      output reg out,
5      output reg feedback
6      );
7
8      assign out = in1 & in2; // The output "out" is read in the des
9      assign feedback = out | in2;
10
11 endmodule
```

W446

We cannot drive or assign reg type variables with an assign statement because a reg variable is capable of storing data and is not driven continuously.

Reg signals can only be driven in procedural blocks such as always and initial

W505

Ensure that the signals or variables have consistent value

W505

Potential Issues:

The violation is reported arise when a variable or a signal is assigned in both blocking and non-blocking manner

W505

Debug/Fix:

Example Code:

```
1  reg [3:0] kenny;
2
3  kenny <= 1;
4  kenny = 2;
5  // kenny is assigned using both blocking and nonblocking assignments.
6  // Hence, a violation is reported in the case.
```

Ensure that all assignments to the same signal or variable are made in a consistent manner

ADDITIONAL

Function/Task

Function	Task
Can**not** have time-controlling statements-delay, and hence executes in the same simulation time unit	Can contain time-controlling statement/delay and may only complete at some other time
Can**not** enable a task, because of the above rule	Can enable othertasks and functions
Should have at least one input argument and cannot have **output** or **inout** arguments	Can have zero or more arguments of any type
Can return only a **single value**	Can**not** return a value, but can achieve the same effect using **output** argument

W243

Recursive task enable

Lintrules
Group 3

Potential Issues:

This rule flags the task statement block, which has a call to task with the same name.

It also reports a message, if a task call is recursive through more than one task.
Ex: task_0 calls task_1 and task_1 calls task_0

Debug/Fix:

In the case of recursive task enable, there is always a chance that two tasks call each other infinitely. Avoid such situations by using loop constructs or some alternate method of writing the code

Example Code:

```
1  module group_3(
2      input in1,
3      input in2,
4      output reg out1
5      );
6
7      always @(in1 or in2) begin
8          MyNameIsVic(out1, in1, in2); // Recursive task calls
9      end
10
11     task MyNameIsVic;
12         output o2;
13         input in1, in2;
14
15         begin
16             MyNameIsVic( o2, in1, in2 ); // Recursive
17         end
18     endtask
19
20 endmodule
```

W428

Ensure that a task is not called inside a combinational block

W428

Potential Issues:

Since tasks may contain event controls, a task called inside a combinational always construct may turn that construct into a sequential always construct.

Violation may arise when a task is called in a combinational block.

W428

Debug/Fix:

To fix this violation, it is advisable to use functions, rather than tasks inside combinational blocks.

Example Code:

```
1  // No Violation, Task called from sequential block
2  always @(posedge clk) begin
3      MyNameIsVic(in, out1);
4  end
5
6  // Violation, Task called from combinational block
7  always @(clk) begin
8      MyNameIsVic(in, out1);
9  end
10
11 // Task
12 task MyNameIsVic;
13     input in;
14     output out1;
15     begin
16         <task body>
17     end
18 endtask
```

W429

Task called in a sequential block

Lintrules
Group 3

W429

Potential Issues:

Since tasks may contain event controls, a task called inside a sequential always construct may unexpectedly create an implicit state machine which may not be synthesizable if the task uses different edges or clocks from the block in which it is instanced.
Therefore, while it is not an error to call a task inside a sequential always block, it should be handled with extra caution.

W429

Debug/Fix:

Nothing to fix if this is done right, but does suggest need for careful review.

Example Code:

```
1  module group3(in, out, clk);
2      input [3:0] in;
3      input clk;
4      output [3:0] out;
5
6      reg [3:0] out;
7      reg [3:0] r1, r2, r3;
8
9      always @(posedge clk) begin
10         r1 = in - 1'b1;
11         MyNameIsVic(r1, r2); // Call a task in sequential always construct
12         out = r3;
13     end
14
15     task MyNameIsVic;
16         <task_body>
17     endtask
18
19 endmodule
```

Kenny

W416

The width of return type and return value of a function should be the same

Lintrules
Group 3

Potential issues

- **WIDTH MISMATCH**

  A violation is reported when the return type width is lesser or greater than the width of the return value of a function.

- **RANGE MISMATCH**

  A violation is reported when there is an order mismatch between the return type and the return value in a function.

EXAMPLE CODE

```
function logic [3:0] test_func ();
// no-violation - widths are same, range are in same order
logic [4:1] tmp;
assign test_func = tmp;
endfunction
```

```
function logic [11:0] test_func ();
// no-violation - widths are same
logic [3:0] [2:0] tmp;
return tmp;
endfunction
```

EXAMPLE CODE

```
function logic [2:0] [3:0] test_func (); // no-violation - widths are same
logic [3:0] [2:0] tmp; // rule will not flag for 'swapped ranges'
return tmp;
endfunction
```

```
function logic [5:0] test_func ();
logic [2:0] tmp;
logic [4:0] tmp_2;
return tmp + tmp_2; // violation when nocheckoverflow yes
endfunction
```

EXAMPLE CODE

```
function logic [4:0] test_func ();
logic [2:0] tmp;
logic [4:0] tmp_2;
return tmp + tmp_2; // violation when nocheckoverflow no
endfunction
```

```
function logic signed [4:0] test_func ();
logic signed [2:0] tmp;
logic [4:0] tmp_2;
return tmp + tmp_2; // violation - signed unsigned
mismatch
endfunction
```

EXAMPLE CODE

```
function logic [2:0] test_func (); // no-violation
return '0;
endfunction
```

```
function logic [0:2] test_func ();
logic [2:0] tmp;
return tmp; // violation - order mismatch - ranges are in
reverse order
endfunction
```

ASTRASTRONG APP

W126~129

do not use delay values which is non-integer, negative, non-constant or containing X or Z.

```
1  out1 = # (8'h1x) in1;
2  integer i=0;
3  out2 = #i in;
```

W18

Do not infer latches

Lintrules
Group 3

Suggested Fix

Check the inference to make sure it is what you intended. If not, prevent latch inferences by providing an explicit else clause at the end of the if statement, or default clause at the end of the case statement, to prevent inferring the latch.

```
1  process (reset, d)
2   begin
3   if (reset = '1') then
4   q <= d;
5   end if;
6  end process;
```

W107

Verilog does not provide a method to check that a connection made to the inputs of a primitive gate (and, or, etc.) is of an appropriate width.If a wider bus is attached, the function simply expands to include the additional bits. This expansion can lead to unexpected behavior if the bus width changes as the design evolves. Thus it is safer to explicitly connect, bit-by-bit, those bits, which should be gated.

Suggested Fix

Make connections bit-by-bit. For example, use and (b[0],b[1],b[2].) rather than and(b[0:3]).

W110

Use this rule to identify the width mismatch between a module port and the net connected to the port of that module instance.

```
inst IN1 (.in1(a[2:0]+b[2:0]));
// Width of expression, a[2:0]+b[2:0], will be 4 (7+7=14, 4 bits wide)
inst IN2 (.in1(a[2:0]*b[2:0]));
// Width of expression, a[2:0]*b[2:0], will be 6
inst IN3 (.in1(a1[2:0]*b1[2]));
// Width of expression, a[2:0]*b1[2], will be 3
inst IN4 (.in1({1'b0,a[2:0]}));
// Width is 3 bits (leading 0 ignored)
inst IN5 (.in1({1'b1,a[2:0]}));
// Width is 4 bits (1+3 bits)
```

W146

Use named-association rather than positional association to connect to an instance

W156

Do not connect buses in reverse order. Making reversed connections (for example, 15:0 connected to 0:15) is legal but bad design practice and may represent an error.

W210

Number of connections made to an instance does not match number of ports on master

W287a

Some inputs to the instance are not driven or unconnected

Lintrules
Group 3

Suggested Fix

If you don’t care about the input value, tie it high or low, but do not let it float.

```
1  module test(in1, in2, clk, out1);
2   input in1, in2, clk;
3   output out1;
4   wire w1, w2;
5   assign w1 = in1 & in2;
6   and a1(out1, w1, w2);
7  // Undriven instance input 'w2'
8  endmodule
```

Lukas

W287c

Inout port of an instance is not connected or connected net is hanging

Potential Issues

- The W287c flags module or gate instances where inout ports are not connected or connected net is not used anywhere in the module

- Such descriptions are allowed, they are generally design mistakes. When in the input mode, an unconnected inout port is equivalent to an undriven input which is a design error in CMOS unless it is a pull-up or pull-down.

How to debug and fix:

Tie the input high or low if it will not be used.

Lukas

W504

Integer is used in port expression

How to debug and fix:

Do not use integers in port expression. Pass integer to lower level module by parameters.

Example code

- In the following example, the W504 rule reports violations because integer variable i and constant integer 2 are used in the expression of port a

```
1  module m;
2      integer i;
3      mm1 u1(.a(i));
4      mm1 u2(.a(2));
5  endmodule
6
7  module mm1(input a);
8  endmodule
```

Lukas

Synthesis Rules

Lukas

badimplicitSM1

Identifies the sequential logic in a non-synthesizable modelling style where clock and reset cannot be inferred

Potential issues

Violation may arise when uncommon sequential logic is used, which is unsynthesizable.

How to debug and fix:

Since the implicit modeling style is uncommon, it is most likely you made a mistake in modeling a conventional synchronous block. For a standard synchronous logic, it is neither required nor a good practice to use signals other than those present in the sensitivity list to set/reset the circuit.

Example code

```
1  module bism1(set,reset,in1,in2,out1);
2
3  input in1,in2,reset,set;
4  output out1;
5  reg clk,out1;
6  always @(posedge clk or negedge set)
7      if(reset)
8          out1 = 0;
9      else if(!set)
10         out1 = 1;
11     else if(in2)
12         out1 = in2;
13     else
14         out1 = in1;
15 endmodule
```

Lukas

badimplicitSM2

Identifies the implicit sequential logic in a non-synthesizable modeling style where states are not updated on the same clock phase

Potential issues

A register which is updated on both edges of a clock in a given sequential block leads to ambiguous synthesis results.

How to debug and fix:

To resolve the violation, break the sequential logic into multiple sequential logic blocks, each of which can independently meet the requirement. If the register depends on both edges of the clock, describe the sequential nature separately and use the combinational logic to generate the final output.

Example code

In the following example code, SpyGlass reports a violation as the event control expressions use multiple edges.

```
1  always
2      begin
3          @(posedge a or negedge a) out1 <= in1;
4          @(negedge a) out2 <= in1;
5      end
6  endmodule
```

Lukas

badimplicitSM4

Identifies the non-synthesizable implicit sequential logic where event control expressions have multiple edges

Potential issues

- A violation is reported A register which is updated on both edges of a clock in a given sequential block leads to ambiguous synthesis results.
- The synthesis tool can get confused about which edge to use for updating the register.
- RTL and gate-level simulation results may not match.

How to debug and fix:

To resolve the violation, break the sequential logic into multiple sequential logic blocks, each of which can independently meet the requirement.

Example code

In the following example code, SpyGlass generates a violation as the state depends on more than one edge of the clk clock:

```
1  module test(out1,out2);
2  output out1,out2;
3  reg out1,out2,a,c,clk;
4  always
5      begin
6          @(posedge clk) out1 <= c; @(negedge clk) out2 <= a;
7      end
8  endmodule
```

Lukas

bothedges

Identifies the variable whose both the edges are used in an event control list

Potential issues

- **A violation is reported when both edges of the same variable are used in an event control list, hence making the variable not synthesizable by some synthesis tools.**

How to debug and fix:

To resolve the violation, replicate the block, one switching on the positive edge and the other on the negative edge.

Example code

```
1  module test(q);
2
3  output q;
4  reg q,d,reset;
5
6  always @(posedge reset or negedge reset)
7    begin if (reset != 0)
8      q = d;
9    end
```

Lukas

mixedsenselist

Mixed conditions in sensitivity list may not be synthesizable

Rule Description

- The mixedsenselist rule flags mixed edge and non-edge conditions in the sensitivity list of an always construct.
- Such conditions in sensitivity list may not be synthesizable by some synthesis tools.

How to debug and fix:

Decide if you really want to trigger the block on any change in the mixed signal. Check if your requirement can be met by mapping to either a positive-edge change or a negative-edge change. If both are required, consider duplicating the block, one triggering on each edge.

Example code

```
1  always @(posedge clock or reset)
2      q = d;
```