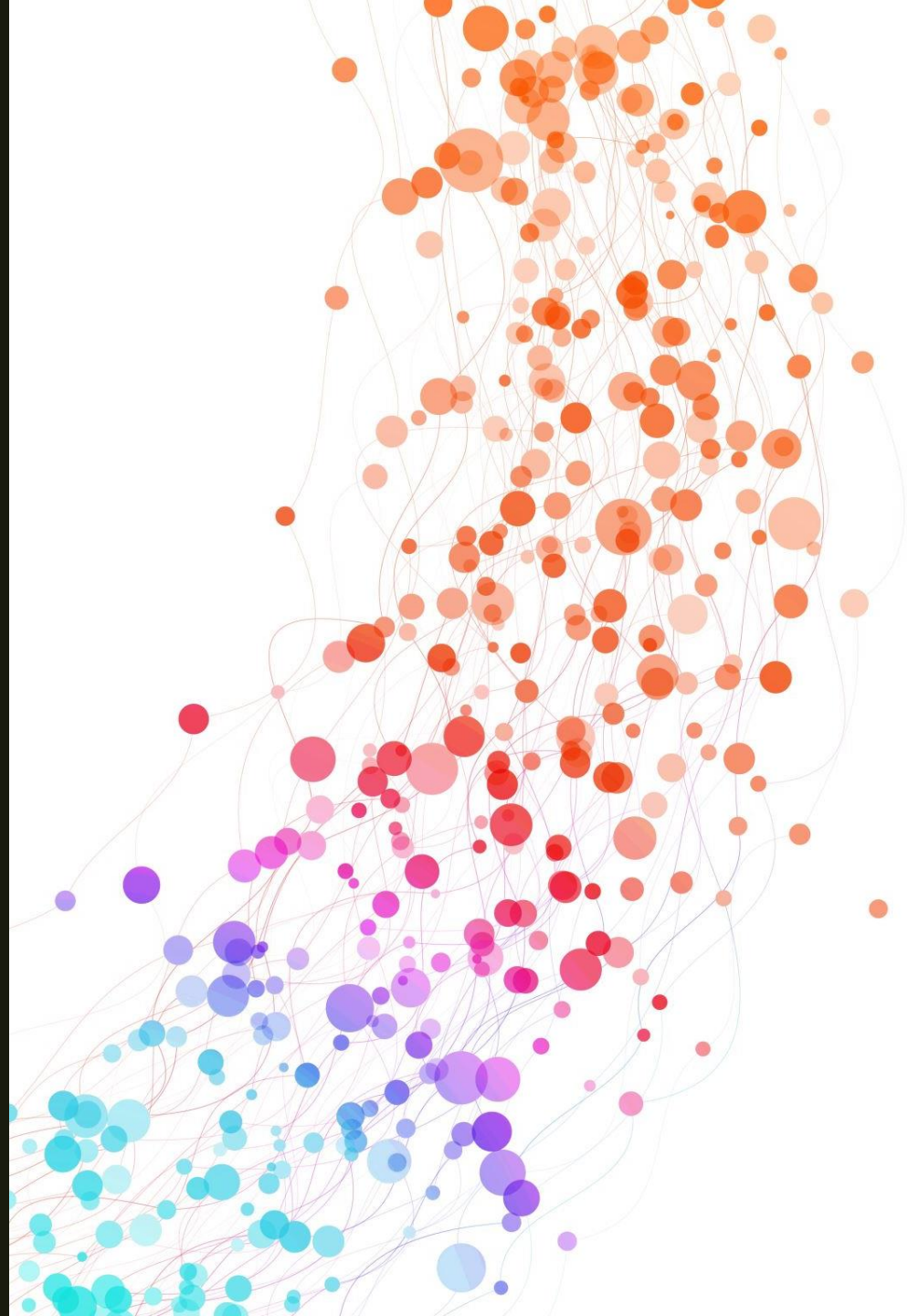
LINTRULES (523-696)

Group5:
110060021曾偉博
110060035黃振寧
110061230張仕謙



SYNTHESIS RULES PART II

Readclock(Warning)

- Unsynthesizable implicit sequential logic: clock read inside always block.
- Suggested Fix : putting the clock test inside translate_off and translate_on pragmas

```
always@ (posedge clk)
  if(clk == 1'b1)
    out1 <= in1 & in2;
```

W182c (Error)

- Identifies the time declarations which are not synthesizable
- The W182c rule reports the "time" variable declarations which are not synthesizable by some synthesis tools.

```
module test(in1, in2);
input in1, in2;
time t;
endmodule
```

W182g (Error)

- Identifies the tri0 net declarations which are not synthesizable
- The W182g rule reports violations for the tri0 net declarations used in the design.

```
module test (y,s0,d1,d0);
input s0, d1, d0;
output y;

tri0 y;

assign y = s0 ? d0 : d1;
endmodule
```

W182h (Error)

- Reports tri1 net declarations that are not synthesizable
- The W182h rule reports the tri1 net declarations even if the net is not used in the design.

```
module test(y,s0,d1,d0);
input s0, d1, d0;
output y;
tri1 y;

  assign y = s0 ? d0 : d1;
endmodule
```

W182k (Error)

- Reports trireg declarations that are not synthesizable
- The W182k rule reports trireg register declarations.

```
module test(y,s0,d1,d0);
input s0, d1, d0;
output y;

trireg y;

assign y = s0 ? d0 : d1;
endmodule
```

W182n (Error)

- Reports MOS switches, such as cmos, pmos, and nmos, that are not synthesizable
- These switches should be used for modeling only and are not synthesizable by most synthesis tools.

```
module test(out, in1, in2);
input in1,in2;
output out;
wire out1,out2;
wire n;
cmos (out,in1,in2,n);
pmos (out1,in1,in2);
nmos (out2,in1,in2);
endmodule
```

W213 (Warning)

- Reports PLI tasks or functions that are not synthesizable
- Parameter: ignore_pli_tasks_and_functions
- Suggested Fix: translate_off and translate_on pragmas

```
module test (in1, clk);

input in1, clk;
always @ (clk)
  $display ("Value of in1 %b\n", in1);

endmodule
```

W218 (Warning)

- Reports multi-bit signals used in sensitivity list
- The W218 rule reports event expressions that check for an edge on a multi-bit signal.

```
module test1(in1,clk,out1);
input [2:0] in1, clk;
output [2:0] out1;

reg [2:0] out1;

always @(posedge clk)
out1 = in1;

endmodule
```

W239 (Warning)

- Reports hierarchical references that are not synthesizable
- Hierarchical references, such as top.abx.clk , are a useful way to probe design behavior during debug.

```
module top(output [3:0] w2);
assign w2 = temp.w1;

endmodule
module temp();
wire [3:0] w1;
endmodule
```

W250 (Warning)

- Reports disable statements that are not synthesizable
- Parameter: avoid_synth_disable (default: no)
- Set this parameter to yes, to enable the rule to not report violations for synthesizable disable constructs.
- synthesizable disable constructs: tasks, blocks (sequential, whose start-end is defined by begin-end)
- Parallel blocks (fork-join) is not synthesizable disable construct.

```
module case_250 (clk1,in,cntr,out);

  input clk1,in;
  input [1:0] cntr;
  output out;

  initial begin
   f_190;
  end

  initial begin
   #10 disable f_190;
  end

task f_190;
reg clk, f_191;
reg[1:0] cntr_f;
  if ( clk == 1'b0) f_191 = cntr_f[1];
  else  f_191 = cntr_f[0];
endtask

endmodule
```

W294 (Warning)

- Reports real variables that are unsynthesizable
- The W294 rule reports real variables used in the design.

```
module test(in1, in2, z);
input in1;
input in2;
output z;
real r = 0.025;    //VIOLATION
assign z = r + in1 + in2;
endmodule
```

W295 (Error)

- Reports event variables that are not synthesizable
- Event variables have no physical equivalent and therefore are not be synthesizable

```
module test (clk1, in, cntr, out);
  input clk1, in;
  input [1:0] cntr;
  output out;

  event test;
  reg a_1, a_2;

  always @(clk1 or test)
    a_2 = in;

  always @(test)
    a_1 <= in;

  assign out = a_1 + a_2;
endmodule
```

W339

- Identity operators and non-constant divisors are not synthesizable.
- The W339 rule runs the W339a rule. (Identity operators are not synthesizable)

W339a (Warning)

- Case equal operator (===) and case not equal (!==) operators may not be synthesizable
- The case equality operators compare non-physical values (X) as well as real values
- Some synthesis tools may be able to handle these operators but reduce them to their non-case equivalents (for example === gets changed to ==).
- Suggested Fix: translate_off / translate_on pragmas

case equal operator

```
module MEM(clk, address, enable, data);
  input [7:0] data;
  input enable, clk;
  output [7:0] address;

  reg [7:0] address;

  always @(posedge clk)
    begin
      if (enable === 1)
        address <= data;
      else
        address <= 8'h00;
    end
endmodule
```

case not equal operator

```
module test(in1, in2, clk, out1);
  input [1:0] in1, in2;
  input clk;
  output [1:0]out1;

  reg [1:0]out1;

  wire [1:0] count;

  assign count = (in1 !== 1'b1)? 2'b10 : 2'b01;

  always @(posedge clk)
    begin
      out1 <= (in2 !== count) ? in1 : in2;
    end
endmodule
```

W430 (Warning)

- The "initial" statement is not synthesizable
- The W430 rule reports violation for initial constructs used in the design.
- The initial constructs have no physical equivalent.

```
module W430_mod1(in1,clk,out1,out2);
  input in1,clk;
  output reg out1,out2;

  initial
  begin
    out1 = 1'b0;
    out2 = 1'b0;
  end

endmodule
```

W442

- This rule group checks all synthesis issues related to reset
- This rule runs W442a, W442b, W442c, and W442f rules.

W442a (Error)

Ensure that for unsynthesizable reset sequence, first statement in the block must be an if statement. (except in testbench)

Reset should be at top level !

```
always@ ( posedge clk or negedge rst_n )
        if ( !rst_n)
        ......
```

W442b (Error)

Reset should only be compared with a constant.

Violation	Compliant Code

always @ (posedge reset or negedge clk)
  begin
  if (reset != b)
  ……
  end
	assign actual_reset = (reset != b);
	
	always @ (posedge actual_reset or negedge clk)
	  begin
	  if (actual_reset)
	  ……
	  end

W442c (Error)

Ensure that the unsynthesizable reset sequence are modified only by ! or ~ in the if condition

Violation	Compliant Code

always @ (posedge reset or negedge clk)
begin
if (&reset)
.....
end
	assign actual_reset = (&reset);
	
	always @ (posedge actual_reset or negedge clk)
	begin
	if (actual_reset)
	.....
	end

W496a (Warning)

Reports comparison to a tristate in a condition expression

W496b (Warning)

Reports comparison to a tristate in a case statement

Comparison to a tristate value does not have a physical equivalent in synthesis and always defaults to a false value.

EXPRESSION RULES

W116 (Warning)

Identifies the unequal length operands in the bit-wise logical, arithmetic, and ternary operators

This rule does not report a violation for integer variable expressions.

W159 (Warning)

Condition contains a constant expression

The portion of your logic may be permanently disabled/optimized out as a result.

W224 (Warning)

Multi-bit expression found when one-bit expression expected

Might cause confusion.

W341 (Warning)

Constant will be 0-extended

E.g. 8'b0x => 8'b0000000x

W342 (Warning)

Reports constant assignments that are X-extended

E.g. 8'bx0 => 8'bxxxxxxx0

W343 (Warning)

Reports constant assignments that are Z-extended

E.g. 8‘bz0 => 8'bzzzzzzz0

W362 (Warning)

Reports an arithmetic comparison operator with unequal length

Suggested Fix: explicitly comparing sub-expressions of equal width

```
module top(q,clk, d,reset);
input  clk,d,reset;
output  q;
reg q;
reg [3:0] a;
reg [7:0] b;
reg [1:0]data;

always @(posedge clk)
begin
if (data == reset)    //violation
    a <=17;
else
    b =  8'hbb;
end
endmodule
```

W443 (Warning)

'X' value used

```
module top(out, en1);
    input en1;
    output reg [1:0] out;

  always @ (en1)begin
        if (en1)
            out<=2'bxx;
        else
            out<=2'b01;
  end

endmodule
```

W444 (Warning)

'Z' or '?' value used

```
module top(out, in1, en1);
    input in1;
    input [1:0] en1;
    output out;
    reg out;

  always @ (en1 or in1)
    begin
        if (en1 == 2'b1z)
            out <= in1;
        else
            out <= 1'b?;
        end

endmodule
```

W486(Warning)

Reports shift overflow operations

```
wire [2:0] net1, net2;
assign net1 = net2 << 1;
```

```
module mod(in1, clk, sel, out1);
  input [2:0] in1;
  input clk, sel;
  output [2:0] out1;
  reg [2:0] out1;

  always @(posedge clk)
    if(sel)
      out1 = in1;
    else
      out1 = in1 << 1;     //violation(Rhs width (Expr: '(in1
<< 1)') is more than lhs Lhs width (Expr: 'out1'), this may
cause overflow)
```

W490(Warning)

A control expression/sub-expression is a constant

Suggested Fix: use the correct width and value specification, rather than depending on default extension

```
module test (in, in1, out, out1, out2, clk);
  input [3:0] in,in1;
  input clk;
  output out, out1, out2;

  reg out, out1, out2;
  wire data, node;
  parameter par = 4'b1010;

  assign data = 2 ? in : in1;
  assign data = par ? in : in1;

  always @(clk)
    begin
      if(3)
        out = node ? in1 : in;
      else
        out = in1;
      while (par < 3)
        out1 = in1 & in;
      while (par[2])
        out2 = in1 & in;
   end
endmodule
```

W491(Warning)

Reports case expression with a width greater than the specified value

E.g. 8'b?0 => 8'b???????0

W561(Warning)

A zero-width-based number may be evaluated as 32-bit number

```
`define X 0
... a = `X'h3
```

In the above example, the based number will be evaluated as 32 bit number, which may not be what you intended.

W563(Warning)	Reduction of a single-bit expression is redundant
W575(Warning)	Logical NOT operating on a vector
W576(Warning)	Logical operation on a vector

W415(Warning)

Reports variable/signals that do not infer a tristate and have multiple simultaneous drivers

W415a(Warning)

Signal may be multiply assigned (beside initialization) in the same scope

```
always @(in1 or in2)
  begin
    out = in1;
    out = in2;
  end
```

```
always @(in1 or in2 or a or b)
  begin
    if (a)
      out = in1;
    if (b)
      out = in2;
  end
```

```
a = b;
if (C1)
  begin
    a = b1; //first assignment
    if (C12)
      a = b12;  //second assignment
  end
```