mirafra
TECHNOLOGIES

Lint - Basics

Dr. Ipsita Biswas Mahapatra
Member of Technical Staff
Mirafra Technologies Pvt. Ltd.

Introduction

mirafra TECHNOLOGIES

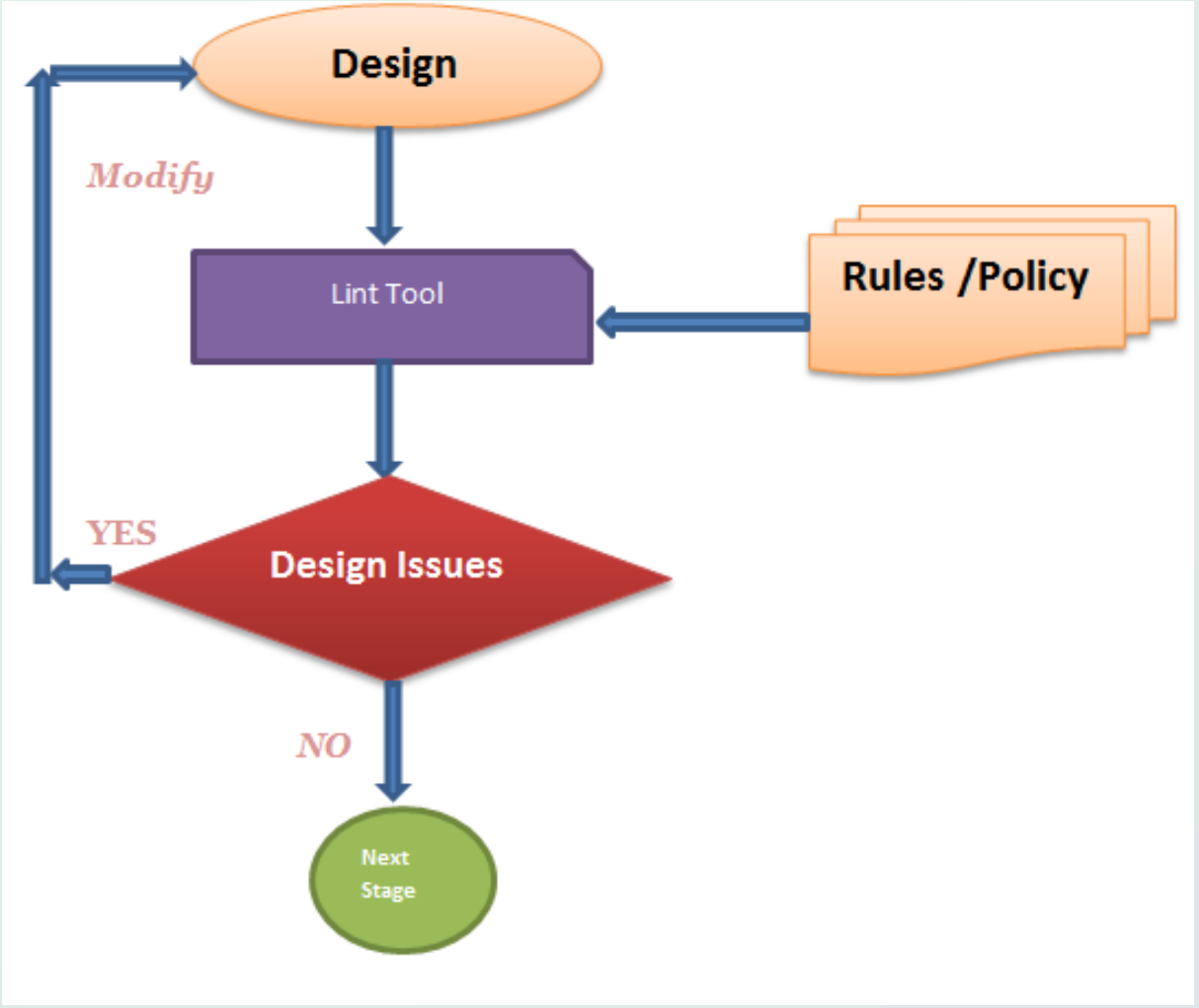
- Static code analysis checks used to flag
  - Programming errors
  - Stylistic errors
  - Suspicious constructs
- Need for linting?
  - Synthesized output of RTL code is input to multiple stages of VLSI design flow
    - Any coding issues, can result multiple iterations of RTL update, synthesis
- Linting checks the cleanliness and portability of HDL code for the later stages of VLSI design flow.

How does it work?

mirafra TECHNOLOGIES

STEPS:

- Change configuration/Define rule set.
- User can change the configuration settings and modify the rules set by enabling or disabling as per their requirement.
- Invoke tool and Source rules
- Analyse Design
- Elaborate
- Generate report



Few Sample Rules

mirafra TECHNOLOGIES

- COMBO_LOOP : It reports if there is a combinational loop in the design.
- TERMINAL_STATE : It reports, if a state in a FSM that once entered never reaches to another state via next state assignment.
- COMBO_NBA : It reports NBA reg assignment from a combo block.
- MULTI_DEFINES : It reports, if there are more than one macros with the same name.
- INC_SENS_LIST : If a signal is referenced and not used in the sensitivity list of the block, it reports the failure.

lint_rtl goals

- **This goal checks basic connectivity issues in the design, such as floating inputs, 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.

Important issues addressed in Lint

mirafra
TECHNOLOGIES

- Un-synthesizable constructs
- Unintentional latches
- Unused declarations
- Multiple driver and undriven signals
- Race conditions
- Incorrect usage of blocking & non-blocking assignments
- Incomplete assignments in sub-routines
- Case statement style issues
- Set & Reset conflicts
- Out-of-range indexing

Important issues addressed in Lint

mirafra TECHNOLOGIES

- Differences between simulation and synthesis semantics.
- Opportunities to improve simulation performance.
- Probable simulation errors.
- Chances of matching gate level simulations with RTL simulations.
- Coding guidelines.
- FSM state reachability and coding issues.
- Network and connectivity checks for clocks, resets, and tri-state driven signals.
- Module partitioning.
- Tool flow issues in the upcoming design cycle stages.
- Possible synthesis issues. (e.g., unintended latches inferred or combo loops).

Linting tools

mirafra
TECHNOLOGIES

- Spyglass
- Realintent (Ascentlint, IIV,Meridian)
- LEDA
- SureLint
- Jasper gold (Cadence)
- Most of the formal verification tools (Onespin, IFV etc)

Rules in Spyglass Lint

mirafra
TECHNOLOGIES

- Array Rules
- Case Rules
- Reset Rules
- Clock Rules
- Usage Rules
- Tristate Rules
- Assign Rules
- Function-Task Rules
- Delay Rules
- Latch Rules
- Instance Rules
- Synthesis Rules
- Expression Rules
- Multiple Driver Rules
- Simulation Rules
- Event Rules
- Loop Rules
- Elab Rules

mirafra
TECHNOLOGIES

Examples of some major violations

Example: 1

mirafra
TECHNOLOGIES

```
module example_1 (a, b, c);
    input a;
    input b;
    output c;
    assign c = a + b;
endmodule
```

- This example is completely ok with compiler. Compiler does not generate warning.
- In this code by default the output variable ‘c’ is a wire. It is not necessary to declare it as a wire.
- But some (not every) synthesis tools may need it to be declared it as a wire.
- If Lint tool applied on the code before the synthesis , it points the above mistake.
- This is just simple example.

Example: 2


```
module example_2 (out, in);
output [1:0] out;
input [2:0] in;
assign out = in[1:0];
endmodule
```

- In this example, the Spyglass Lint tool reports a violation because all bits of the array in are not read.

W111: Not all elements of an array are read

Example 3: Latch inference

mirafra TECHNOLOGIES

```
always@(in1 or in2 or in3 or mem)
begin
case(in3)
2'b00 : out[0] = in1 || mem[0];
2'b01 : out[1] = in1 && mem[1];
2'b11 : out[2] = in2 && mem[3];
endcase
end
```

❑Fix:

- add case clauses (or a default clause) for those cases that are not covered and specify the appropriate behavior in that default clause.

- add pragma full_case to the case statement, implying you know that the remaining cases can never happen.

- In this example where the case construct does not have all possible cases described (case in3=2'b10 is not described) and also does not have a default clause

❑ ***Consequences of Not Fixing:***

- A latch may be inferred, if the case construct does not describe all possible states and also does not have a default clause.
- While you may have intended to infer a latch, it is advisable to examine such cases to avoid creation of unexpected logic.

W69: Ensure that a Case statement specifies all possible cases & has a default clause

Inferred latches (contd ….)

mirafra TECHNOLOGIES

- It is a major issue that can cause a design to malfunction.
- This happens when a Verilog code does not account for the value held by a variable at certain times.
- It occurs due to incomplete usage of certain constructs.
- Example: ***If condition without else block***

```
always @ (posedge clk)
if (a > b)
c <= a;
```

- If this is all the given code, how do we know what happens to “c” when “a” > “b” condition fails?
- This is called ***inferred latch.*** When the tool infers a latch under such circumstances, then the previous value of the variable is maintained.

Inferred latches (contd …)

mirafra TECHNOLOGIES

- To avoid the tool making such an inference, it is always desirable for a designer to explicitly mention what happens in such a case.
- Even if the value is maintained, it can be explicitly coded as shown:

```
always @ (posedge clk)
if (a > b)
c <= a;
else
c <= c;
```

- This will ensure that the latches are not 'inferred' by the tool.

Example 4: Set & Reset conflicts

mirafra TECHNOLOGIES

```
always @( posedge C_SCLK1 or posedge
    C_SRST1 )
begin
if (C_SRST1)
Q1 = 2'b11;
else
Q1 = DataIn[1:0];
end
always @( posedge C_SCLK1 or negedge
    C_SRST1 )begin
if (!C_SRST1)
Q2 = 2'b11;
else
Q2 = DataIn[1:0];
end
```

- ***Consequences of Not Fixing***

- When both polarities of reset/set signal are used, one logic block always remain in a reset/set state. If a reset/set signal is high, the logic that operates on positive reset/set polarity would be reset/set.

- However, when the reset/set signal is de-asserted, some other portion of logic would be reset/set. The usage leads to mutually exclusive blocks of logic.

W392: Reports reset or set signals used with both positive & negative polarities within the same design unit

Example 5: Pos-edge & Neg-edge conflicts

mirafra TECHNOLOGIES

```
module example_6 (out1, out2, in1, in2, clk);
input in1, in2, clk;
output out1, out2;
reg out1, out2;
always @(posedge clk)
out1 = in1;
always @(negedge clk)
out2 = in2;
endmodule
```

❑ *Consequences of Not Fixing*

- As a result of using both the edges, the behavior of that module gets dependent on the duty cycle of the clock.

W391: Reports modules driven by both edges of a clock

Example 6: Undriven nets

mirafra TECHNOLOGIES

```
module example_7(in1, clk, out1);
input [2:0] in1;
input clk;
output [2:0] out1;
reg [2:0] out1;
wire [2:0] net;
always@(posedge clk)
out1 <= in1 && net;
endmodule
```

- ***Consequences of Not Fixing***

- Reading an uninitialized value may result in undriven nets in post synthesis.

W123: Identifies the signals and variables that are read but not set

Example 7: Data Loss

```
module example_8(clk, out);
input clk;
output out;
reg out;
always @(posedge clk) begin
out = 1'b101;
if(out == 2'b0101) begin
end
end
endmodule
```

W19: Reports the truncation of extra bits

❑ ***Consequences of Not Fixing***

- When constant value is wider than the width of the constant, it results in truncation of extra bits. This in turn leads to data loss and unexpected code behavior.
- //Constant 1'b101 will be truncated
- //Constant 2'b0101 will be truncated

Example 8: Latch inference

mirafra TECHNOLOGIES

```
process (reset, d)
begin
if (reset = '1') then
q <= d;
end if;
end process;
```

- Latch inferred for signal 'q'

W18: Do not infer latches

Example 9: Width mis-match

mirafra TECHNOLOGIES

```
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)
```

- ***Consequences of Not Fixing***
  - Connection of a bus which is wider than the port: Excess bits are ignored for the input bus and floated for the output bus.
  - Connection of a bus which is narrower than the port: Missing bits are driven unknown for the input bus and ignored for the output bus.

W110: Identifies a module instance port connection that has incompatible width as compared to the port definition

Example 10: Width mis-match (contd ….)

mirafra TECHNOLOGIES

- One of the more common issues that can be discovered only by lint - ***Mismatches in the width of signals being used on LHS as well as RHS. Example:***

```
wire [3:0] a;
wire [7:0] b;
assign a = b;
```

  - This will cause a width mismatch.
  - The width of signal “a” is 4-bits while the width of “b” is 8-bits.
  - When we assign “b” to “a”, “a” will be able to store only the least significant 4-bits of “b”, ie., b[3:0].
  - The remaining 4-bits of “b” will be lost.
  - Solution – To avoid potential data loss, we should ensure that the signals used in the LHS as well as RHS of an expression is of the same width.

Example 11: Non-synthesizable constructs

mirafra TECHNOLOGIES

```
module example_13(in1, in2, z);
input in1;
input in2;
output z;
real r = 0.025;
assign z = r + in1 + in2;
endmodule
```

❑ ***Consequences of Not Fixing***

- Objects with real values have no physical equivalent and therefore may not be synthesizable by some synthesis tools.
- Real variables should only be used in testbenches and simulation models.

W294:Reports real variables that are unsynthesizable

Example 12: Non-synthesizable constructs (contd ….)

mirafra TECHNOLOGIES

- Case equal operator (===) and case not equal (!==) operators may not be synthesizable

```
module example_14(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
```

W339a:
Case equal operator (===) and case not equal (!==) operators may not be synthesizable

Example 13: Non-synthesizable constructs (contd ….)

mirafra TECHNOLOGIES

```
module example_15(in1,clk,out1,out2);
input in1,clk;
output reg out1,out2;
initial
begin
out1 = 1'b0;
out2 = 1'b0;
end
endmodule
```

- ***Consequences of Not Fixing***

- The initial constructs have no physical equivalent. Therefore, such constructs are Unsynthesizable by some synthesis tools.

W430: The "initial" statement is not synthesizable

Summary

mirafra
TECHNOLOGIES

- Unwanted latch inference
- Set and reset conflicts
- Pos-edge and Neg-edge conflicts
- Undriven nets
- Data loss
- Width mis-match
- Non-synthesizable constructs

mirafra
TECHNOLOGIES

Thank You