VERILOG 설계언어 중급 2일차

반도체설계교육센터
IC DESIGN EDUCATION CENTER

김두영

doo0@hanyang.ac.kr

2016. Aug. 11

강의 시간표 (2일차)

2일차 (8월 11일)

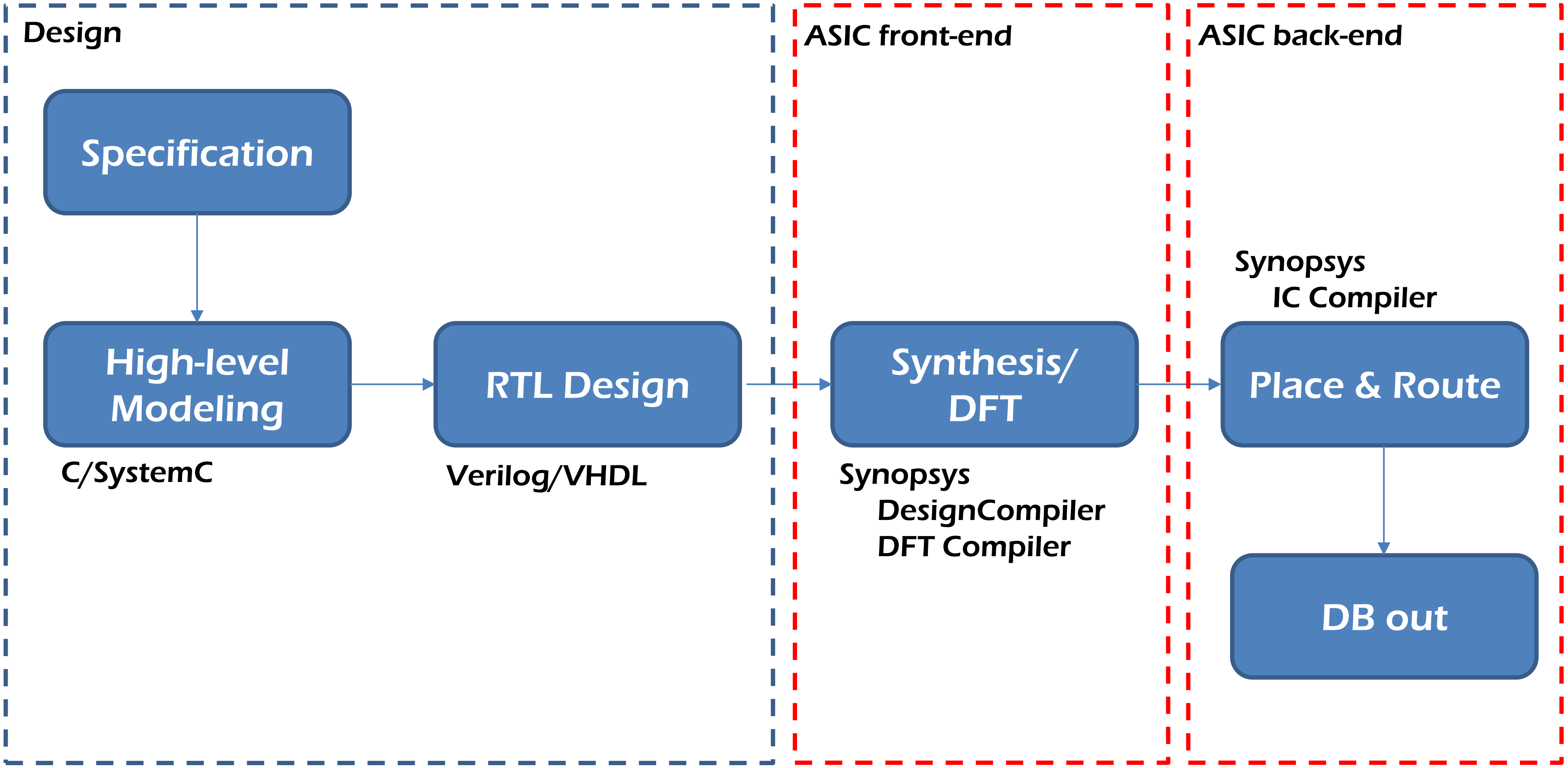
10:00 ~ 11:20	ASIC 개발과 VERILOG HDL
11:40 ~ 13:00	ASIC 개발과 VERILOG HDL (계속)
13:00 ~ 14:00	점심시간
14:00 ~ 15:20	ASIC 개발 실습 – 합성, Timing Closure
15:40 ~ 17:00	ASIC 개발 실습 – LEC, DFT

ASIC 개발과 VERILOG HDL 파트 1

반도체설계교육센터
IC DESIGN EDUCATION CENTER

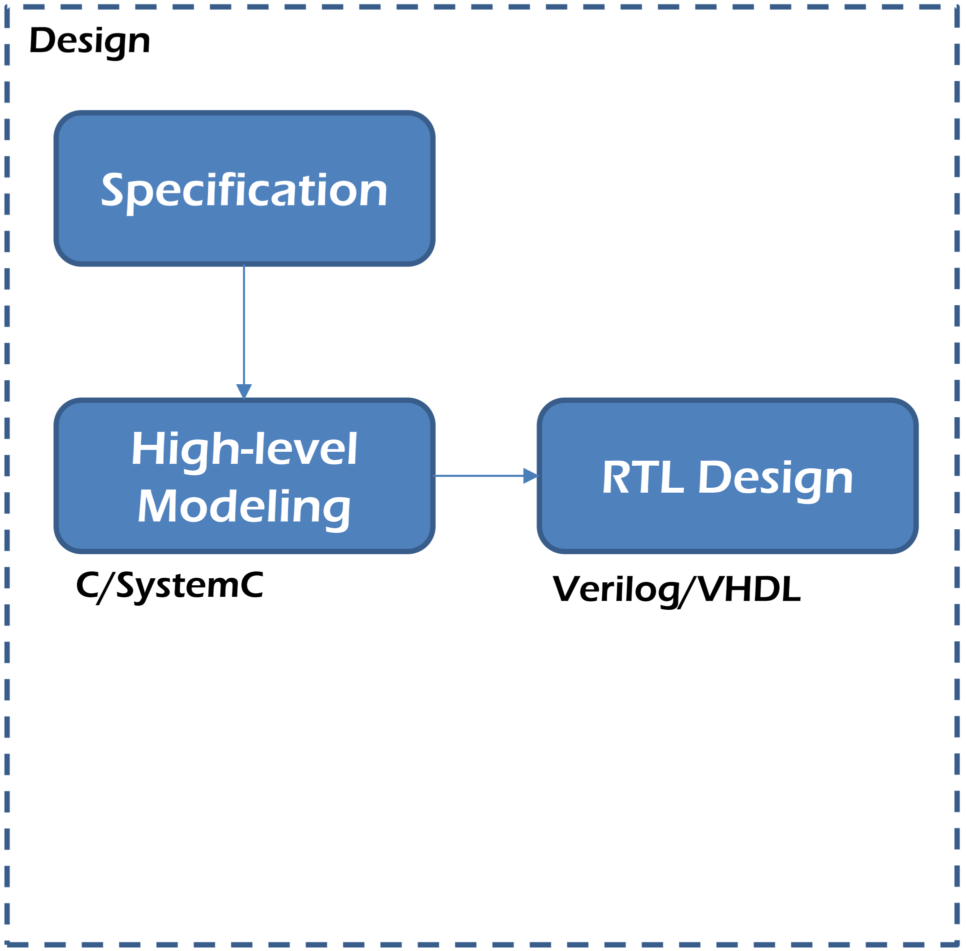
Design Flow - ASIC

1. Design flow for ASIC



Design with Verilog

1. How to verify my RTL design?



* Functional verification

- Simulation : testbench

  How to make a good test-bench for my design?

  Is it enough?

- Formal verification

  High complexity

- Code coverage

  The quality of testbench

- Lint rule check

  Is it synthesizable?

Design with Verilog

2. Code coverage

* Code coverage

- Code coverage tells how well your HDL code has been exercised by your testbench
- Statement coverage
- Block coverage
- Condition/Expression coverage
- Branch/Decision coverage
- Toggle coverage
- FSM coverage

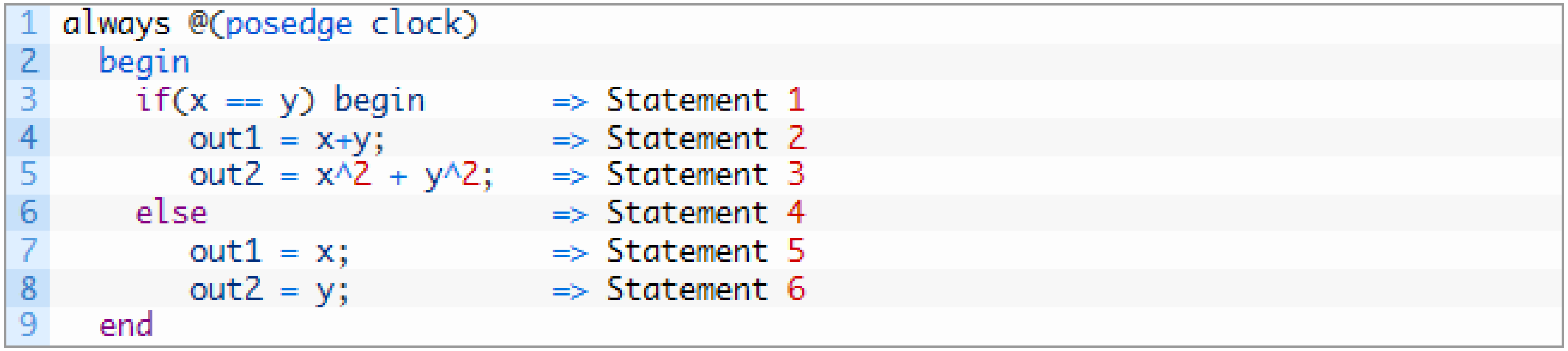
Design with Verilog

2. Code coverage

* **Statement coverage**

- **How many statements(lines) are covered in the simulation, by excluding lines like module, endmodule, comments, timescale.**

- **Example**



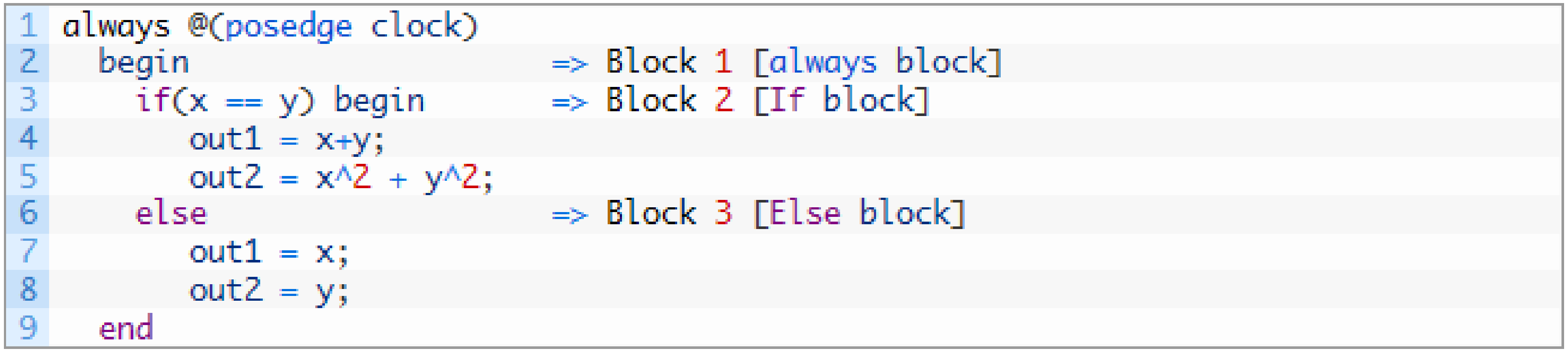
```
always @(posedge clock)
  begin
    if(x == y) begin          => Statement 1
       out1 = x+y;            => Statement 2
       out2 = x^2 + y^2;      => Statement 3
    else                      => Statement 4
       out1 = x;              => Statement 5
       out2 = y;              => Statement 6
  end
```

Design with Verilog

2. Code coverage

* Block coverage

- Block: a group of statements which are in the begin-end or if-else or case or while loop or for loop
- Block coverage gives the indication that whether these blocks are covered in simulation or not
- Example



```
always @(posedge clock)
  begin                        => Block 1 [always block]
    if(x == y) begin           => Block 2 [If block]
       out1 = x+y;
       out2 = x^2 + y^2;
    else                       => Block 3 [Else block]
       out1 = x;
       out2 = y;
  end
```

Design with Verilog

2. Code coverage

* Conditional/Expression coverage

- This gives an indication how well variables and expressions in conditional statements are evaluated

- Example

```
out = (x xor y) or (x and z);
```

All the possible cases would be available as truth table and uncovered expression can be easily identified from the table

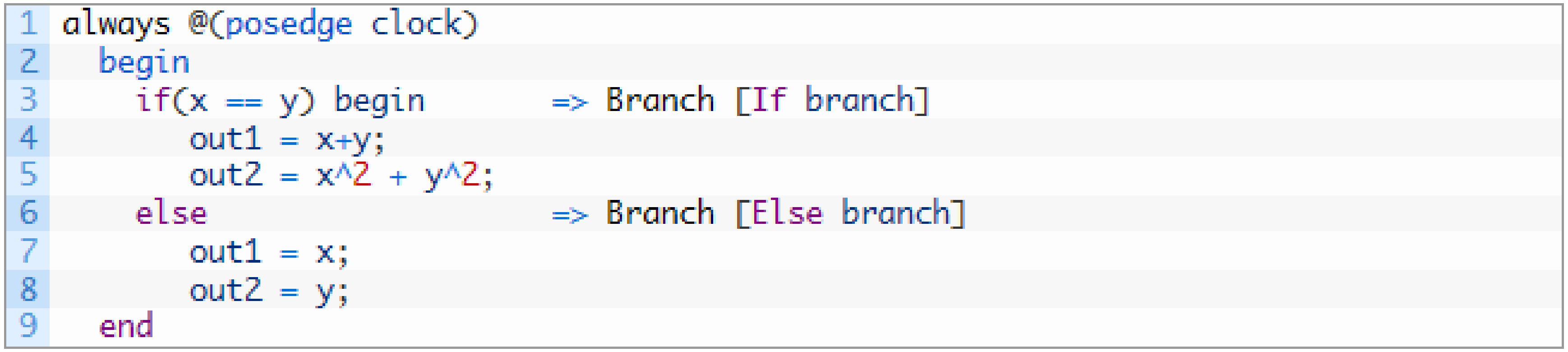
Design with Verilog

2. Code coverage

* Branch/Decision coverage

- Conditions like if-else, case and the ternary operator(?:) statements are evaluated in both true and false cases

- Example



```
always @(posedge clock)
  begin
    if(x == y) begin          => Branch [If branch]
       out1 = x+y;
       out2 = x^2 + y^2;
    else                      => Branch [Else branch]
       out1 = x;
       out2 = y;
  end
```

Design with Verilog

2. Code coverage

* Toggle coverage

- It gives a report that how many times signals and ports are toggled during a simulation run
- It measures activity in the design, such as unused signals or constant signals or less value changed signals

* FSM coverage

- It reports that whether the simulation run could reach all of the states and cover all state transitions in a given FSM
- It is complex coverage type as it works on behavior of the design

Design with Verilog

3. Lint

* Definition

- The generic term given to design verification tools that perform a static analysis of software based on a series of rules and guidelines that reflect good coding practice
- In the hardware-design space, linting is typically applied to hardware description languages (HDLs) such as Verilog, SystemVerilog and VHDL prior to simulation
- the goal is increasingly to clean RTL before entering that increasingly lengthy process
- used to check for potential mismatches between simulation and synthesis

Design with Verilog

3. Lint

* Typical Lint targets

- Unsynthesizable constructs
- Unintentional latches
- Unused declarations
- Driven and undriven signals
- Race conditions
- Incorrect usage of blocking and non-blocking assignments
- Incomplete assignments in subroutines
- Case statement style issues
- Set and reset conflicts
- Out-of-range indexing

Design with Verilog

3. Lint

* **Spyglass Synthesis Rule – badimplicitSM1**

- Identifies the sequential logic in a non-synthesizable modelling style where clock and reset cannot be inferred
- Unsynthesizable
- Severity level: Error

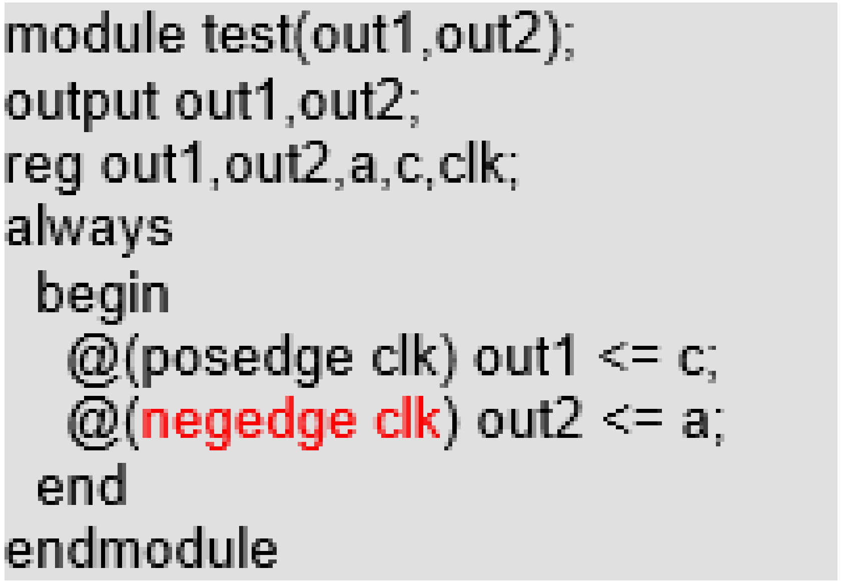
```
module bism1(set,reset,in1,in2,out1);

input in1,in2,reset,set;
output out1;
reg clk,out1;
always @(posedge clk or negedge set)
  if(reset)
    out1 = 0;
  else if(!set)
    out1 = 1;
  else if(in2)
    out1 = in2;
  else
    out1 = in1;
endmodule
```

Design with Verilog

3. Lint

* Spyglass Synthesis Rule – badimplicitSM2

- Identifies the implicit sequential logic in a non-synthesizable modeling style where states are not updated on the same clock phase
- The synthesis tool can get confused about which edge to use for updating the register
- RTL and gate-level simulation results may not match
- Severity level: Error



```
module test(out1,out2);
output out1,out2;
reg out1,out2,a,c,clk;
always
  begin
    @(posedge clk) out1 <= c;
    @(negedge clk) out2 <= a;
  end
endmodule
```

Design with Verilog

3. Lint

* Spyglass Synthesis Rule – badimplicitSM4

- Identifies the non-synthesizable implicit sequential logic where event control expressions have multiple edges
- The synthesis tool can get confused about which edge to use for updating the register
- RTL and gate-level simulation results may not match
- Severity level: Error

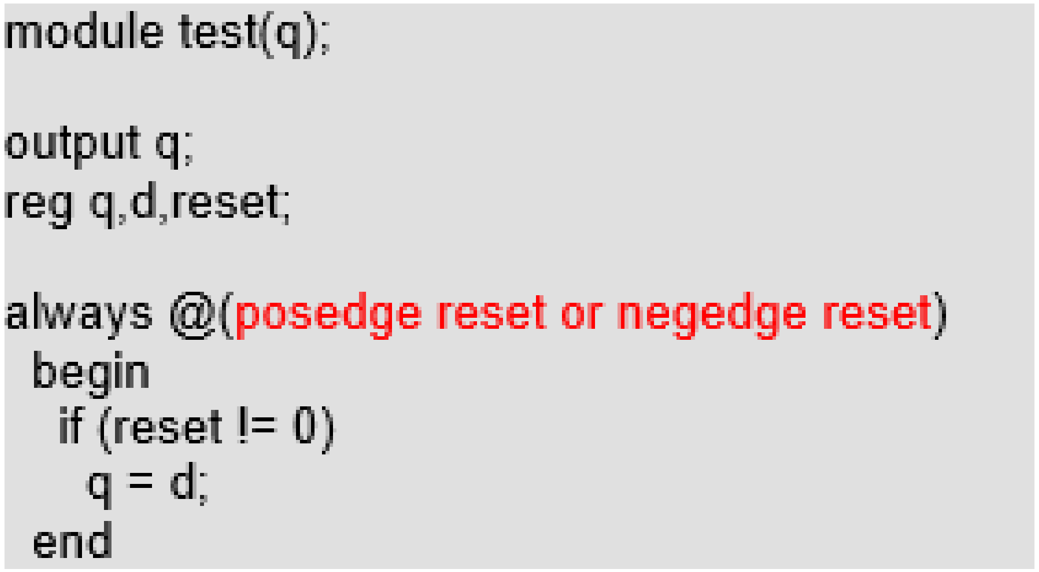
```
always
  begin
    @(posedge a or negedge a) out1 <= in1;
    @(negedge a) out2 <= in1;
  end
endmodule
```

Design with Verilog

3. Lint

* Spyglass Synthesis Rule – bothedges

- Identifies the variable whose both the edges are used in an event control list
- Synthesis tools do not allow both edges of the same variable in an event control list
- Severity level: Error



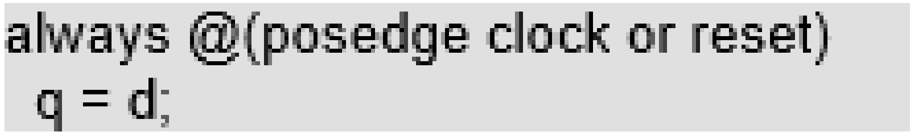
```
module test(q);

output q;
reg q,d,reset;

always @(posedge reset or negedge reset)
  begin
    if (reset != 0)
      q = d;
  end
```

Design with Verilog

3. Lint

* Spyglass Synthesis Rule – mixedsenselist

- Mixed conditions in sensitivity list may not be synthesizable
- It flags mixed edge and non-edge conditions in the sensitivity list of an always construct.
- Severity level: Error



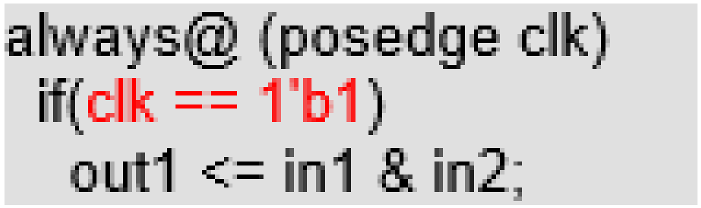
```
always @(posedge clock or reset)
  q = d;
```

Design with Verilog

3. Lint

* **Spyglass Synthesis Rule – readclock**

- Unsynthesizable implicit sequential logic: clock read inside always block
- It flags sequential descriptions where the clock signal is read inside the always construct
- Severity level: Warning



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

Design with Verilog

3. Lint

* Spyglass Synthesis Rule – W182g, W182h, W182n

- Identifies the tri0/tri1 net declarations which are not synthesizable
- The tri0 and tri1 net declarations represent connections with resistive pull-down or pull-up
- Some technologies may not support tristate operations
- Severity level: Error

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

tri0 y;

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

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

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

Design with Verilog

3. Lint

* Spyglass Synthesis Rule – W182n

- Reports MOS switches, such as cmos, pmos, and nmos, that are not synthesizable
- Except for custom or analog design, transistor-level design is generally discouraged because behavior and timing are difficult to predict under all possible circumstances
- Severity level: Error

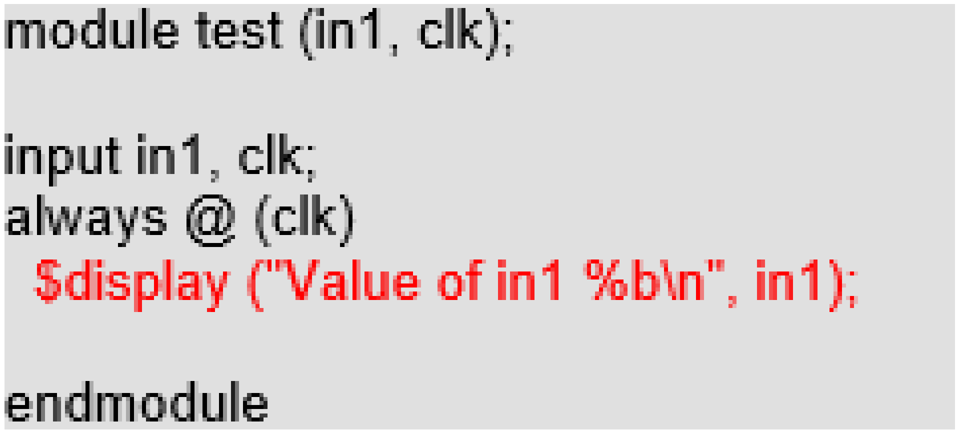
```
module test(out, in1, in2);
input in1,in2;
output out;
wire out1,out2;
wire n;
cmos (out,in1,in2,n);
pmos (out1,in1,in2);
nmos (out2,in1,in2);
endmodule
```

Design with Verilog

3. Lint

* Spyglass Synthesis Rule – W213

- Reports PLI tasks or functions that are not synthesizable
- The PLI tasks or functions, such as $display, have no physical meaning and therefore are not synthesizable
- translate_off & translate_on
- Severity level: Warning



```
module test (in1, clk);

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

endmodule
```

Design with Verilog

3. Lint

* Spyglass Synthesis Rule – W218

- Reports multi-bit signals used in sensitivity list
- Edge specifications for multi-bit expression is semantically incorrect
- In such cases, only the changes on least significant bit are important
- Severity level: Warning

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

reg [2:0] out1;

always @(posedge clk)
out1 = in1;

endmodule
```

Design with Verilog

3. Lint

* Spyglass Synthesis Rule – W239

- Reports hierarchical references that are not synthesizable
- Synthesis tools, in general, do not create connections corresponding to these references
- Severity level: Warning

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

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

Design with Verilog

3. Lint

* Spyglass Synthesis Rule –W294

- Reports real variables that are unsynthesizable
- Objects with real values have no physical equivalent and therefore may not be synthesizable
- Severity level: Warning

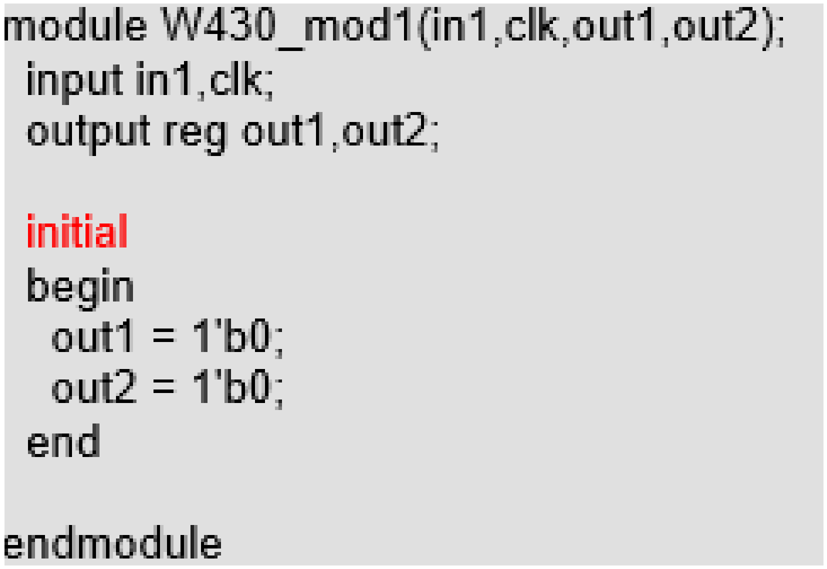
```
module test(in1, in2, z);
input in1;
input in2;
output z;
real r = 0.025; //VIOLATION
assign z = r + in1 + in2;
endmodule
```

Design with Verilog

3. Lint

* **Spyglass Synthesis Rule – W430**

- *The "initial" statement is not synthesizable*
- *The initial constructs have no physical equivalent*
- *Severity level: Warning*



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

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

endmodule
```

Design with Verilog

3. Lint

* Spyglass Synthesis Rule – W442a

- Ensure that for unsynthesizable reset sequence, first statement in the block must be an if statement
- *In general, synthesis tools expect that the first statement inside an asynchronously reset block is an if statement*
- Severity level: Error

```
module DFF(D, clk, R, Q);
 output Q;
 input D, clk, R;
 reg Q;

 reg d1, d2;
 always@(posedge clk or posedge R)

  begin
   d2 = d1;
   if(R) Q = 0;
   else Q = D;
  end
endmodule
```

Design with Verilog

3. Lint

* Spyglass Synthesis Rule –W442b

- Ensure that for unsynthesizable reset sequence, reset condition is not too complex
- Violation may arise when a reset signal is compared with any other signal or variable or a non-constant expression
- Severity level: Error

```
module mod(in1, clk, reset, set, out1);
  input [1:0] in1;
  input clk, reset, set;
  output [1:0] out1;
  reg [1:0] out1;

  always @(posedge clk or posedge reset)
    begin
      if(reset == !set)
        out1 <= 2'b00;
      else
        out1 <= in1;
    end
endmodule
```

Design with Verilog

3. Lint

* Spyglass Synthesis Rule –W442c

- Ensure that the unsynthesizable reset sequence are modified only by ! or ~ in the if condition
- Violation may arise when a reset signal is being modified by an operator other than logical inverse (!) and bit-wise inverse (~) operators
- Severity level: Error

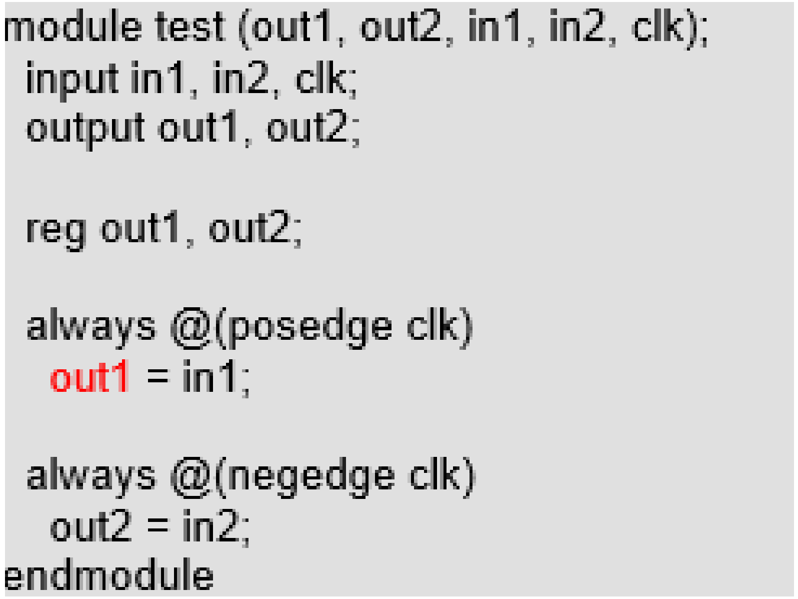
```
module test(reset,q);
output q;
input reset;
reg q,clk,d;

always @(posedge reset or negedge clk)
  begin
    if (&reset)
      q = 1'b0;
    else
      q = d;
  end
endmodule
```

Design with Verilog

3. Lint

* Spyglass Lint_Clock Rule – W391

- Reports modules driven by both edges of a clock
- As a result of using both the edges, the behavior of that module gets dependent on the duty cycle of the clock
- Severity level: Warning



```
module test (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
```

Design with Verilog

3. Lint

* **Spyglass Lint_Clock Rule – W401**

- Clock signal is not an input to the design unit
- Localize clock generation and gating to a single module if possible
- Timing and test issues can be managed carefully with respect to that one module rather than in many locations in the design
- Severity level: Warning

```
module test(in1, in2, in3, out1);
  input in1, in2, in3;
  output out1;

  my_clock mod1(in1, in2, in3, out1);
endmodule

module my_clock(in1, in2, clock, out1);
  input in1, in2, clock;
  output out1;

  reg DFF_clk, out1;

  always@(posedge clock)
    begin
      DFF_clk <= in1;
    end

  always@(posedge DFF_clk)
    begin
      out1 <= in2;
    end
endmodule
```

Design with Verilog

3. Lint

* Spyglass Lint_Clock Rule – W422

- Unsynthesizable block or process: event control has more than one clock
- It reports violation for potentially un-synthesizable block
- Severity level: Warning

```
module mod(in1, in2, clk1, clk2, out1);
input in1, in2;
input clk1, clk2;
output out1;
reg out1;

  always@(posedge clk1 or posedge clk2)
    out1 = in1 ^ in2;

endmodule
```

Design with Verilog

3. Lint

* Spyglass Lint_Reset Rule – W392

- Reports reset or set signals used with both positive and negative polarities within the same design unit
- When both polarities of reset/set signal are used, one logic block always remain in a reset/set state
- Violation may arise when two different IP blocks are connected together at a SoC level
- Severity level: Warning

```
module test3( Q1, Q2, DataIn, C_SCLK1, C_SRST1 );
input [2:0] DataIn;
input C_SCLK1;
input C_SRST1;
output [2:0] Q1, Q2;
reg clk;
reg [2:0] Q1,Q2,Q3;
always @( posedge C_SCLK1 or posedge C_SRST1 )
begin
if (C_SRST1) //VIOLATION
  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
endmodule
```

Design with Verilog

3. Lint

* Spyglass Lint_Reset Rule – W395

- Multiple asynchronous resets or sets in a process or always may not be synthesizable
- It reports if more than one asynchronous reset or set signals exist in the same process or always block
- Severity level: Warning

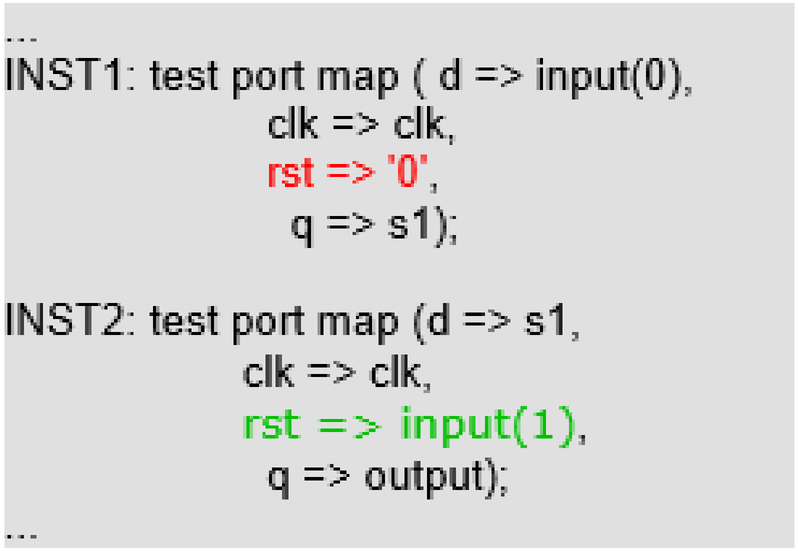
```
always @(posedge clk, posedge rst1, posedge rst2)
begin
if (rst1)
t <= 1'b0;
else if (rst2)
t <= 1'b0;
else if(clk)
t <= in1 & in2;
end
```

Design with Verilog

3. Lint

* **Spyglass Lint_Reset Rule – W501**

- A connection to a reset port should not be a static name
- Always connect a real signal. Tie that signal off if you really want to disable the reset
- Severity level: Warning



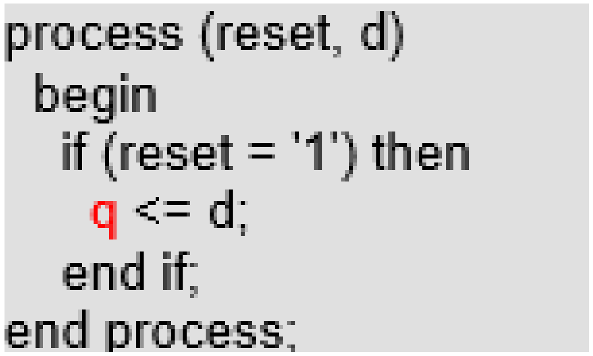
```
...
INST1: test port map ( d => input(0),
                       clk => clk,
                       rst => '0',
                         q => s1);

INST2: test port map (d => s1,
                     clk => clk,
                     rst => input(1),
                       q => output);
...
```

Design with Verilog

3. Lint

* Spyglass Lint_Latch Rule – W18

- Do not infer latches
- 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
- Severity level: Warning



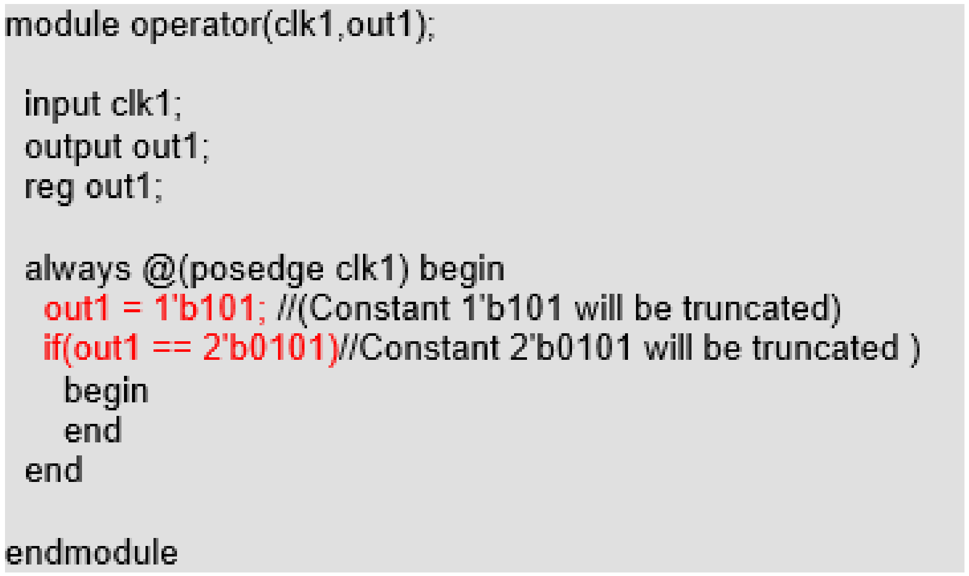
```
process (reset, d)
 begin
  if (reset = '1') then
    q <= d;
  end if;
end process;
```

Design with Verilog

3. Lint

* Spyglass Assign Rule – W19

- Reports the truncation of extra bits
- When constant value is wider than the width of the constant, it results in truncation of extra bits
- To resolve the violation, determine the width specification and the constant value
- Severity level: Warning



```
module operator(clk1,out1);

 input clk1;
 output out1;
 reg out1;

 always @(posedge clk1) begin
   out1 = 1'b101; //(Constant 1'b101 will be truncated)
   if(out1 == 2'b0101)//Constant 2'b0101 will be truncated )
     begin
     end
 end

endmodule
```

Design with Verilog

3. Lint

* Spyglass Assign Rule – W336

- Blocking assignment should not be used in a sequential block
- When a blocking assignment is used in a sequential block, inherent sequence of operation is implied in simulation
- However, the synthesized hardware may behave in a concurrent fashion
- Severity level: Warning

```
module test3(clk, reset, d, q);
input clk, reset, d;
output q;
reg q;

always @(posedge clk or negedge reset)
begin
  if (!reset)
    q = 1'b0;
  else
    q = d;
end

endmodule
```

Design with Verilog

3. Lint

* Spyglass Assign Rule – W414

- Reports nonblocking assignment in a combinational block
- Violation may arise when a nonblocking assignment is used in a combinational block
- Not fixing the violation may result in unexpected code behavior
- Severity level: Warning

Case 1

```
out1 <= in1 & in2;
out2 = in3 & in4;
```

Case 2

```
out1 <= in1 & in2;
out2 < = in3 & in4;
```

Design with Verilog

3. Lint

* Spyglass Case Rules

- W69: case constructs that do not have all possible clauses described and also do not have the default clause
- W71: case constructs that do not contain a default clause
- W187: case constructs where the default clause is not the last clause
- W263: case constructs that do not have all possible clauses described and have a default clause
- W398: Duplicate choices in CASE construct

Design with Verilog

3. Lint

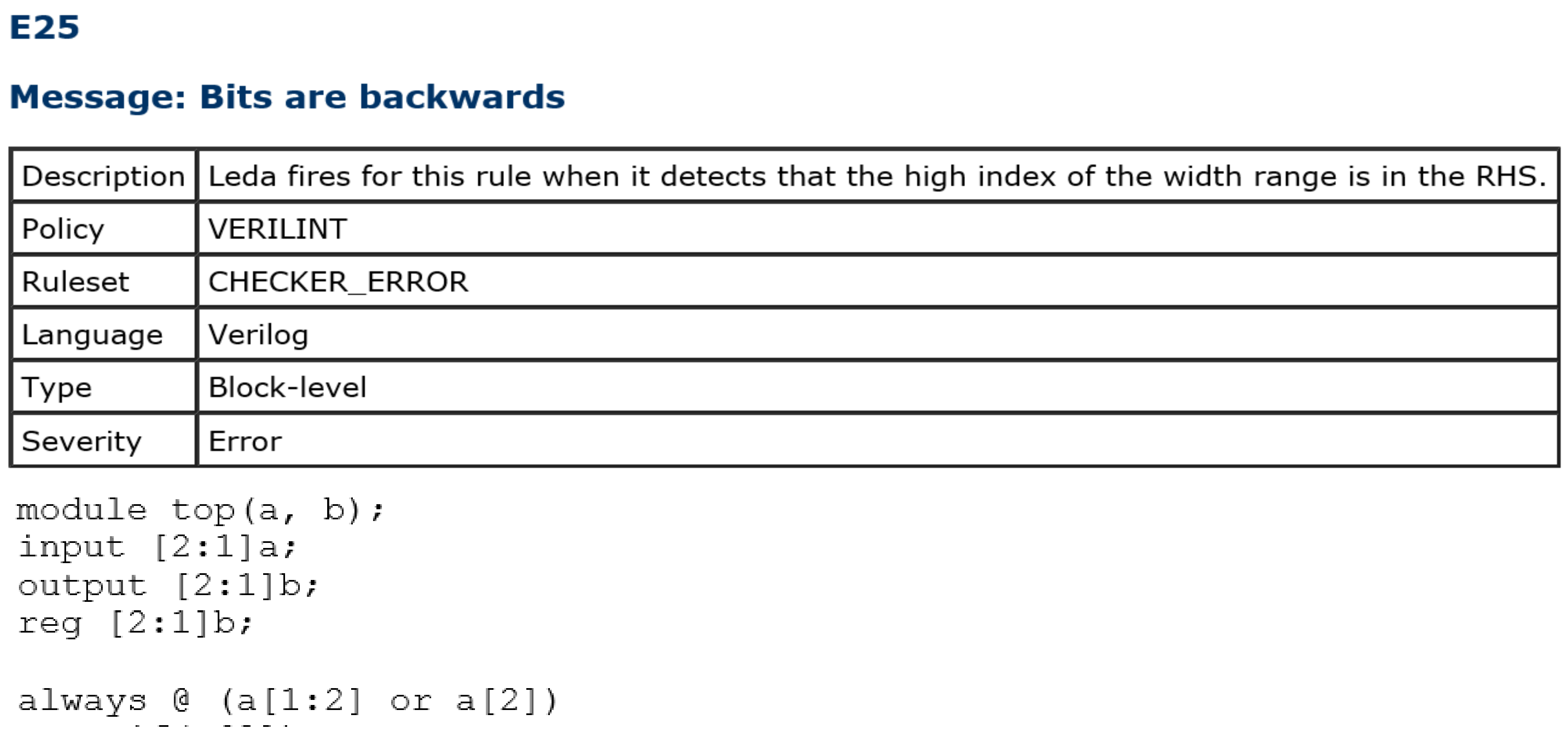
* Spyglass Instance Rules

- W107: Bus connections to primitive gates
- W110: Width mismatch between a module port and the net connected to the port in a module instance
- W287a: Module instances where nets connected to input ports are not driven

Design with Verilog

3. Lint

* **Lint rule examples (Leda Verilint policy)**



E25

Message: Bits are backwards

Description	Leda fires for this rule when it detects that the high index of the width range is in the RHS.
Policy	VERILINT
Ruleset	CHECKER_ERROR
Language	Verilog
Type	Block-level
Severity	Error

```
module top(a, b);
input [2:1]a;
output [2:1]b;
reg [2:1]b;

always @ (a[1:2] or a[2])
```

Design with Verilog

3. Lint

* Lint rule examples (Leda Verilint policy)

E54

Message: Instance name required for module

Description	Leda fires for this rule when there is no instance name for a module. To solve this problem, name the instance.
Policy	VERILINT
Ruleset	CHECKER_ERROR
Language	Verilog
Type	Block-level
Severity	Error

```
module top (clk, reset, d, q);
input clk, reset, d;
output q;

test (clk, reset, d, q); //E54, no instance name.
```

3. Lint

* Lint rule examples (Leda Verilint policy)

E66

Message: Not a constant expression

Description	Leda fires for this rule when it detects variables in parameter or defparam definitions. To solve this problem, use a constant expression.
Policy	VERILINT
Ruleset	CHECKER_ERROR
Language	Verilog
Type	Block-level
Severity	Error

```
module test;

integer i;
parameter k = i;
```

Design with Verilog

3. Lint

* Lint rule examples (Leda Verilint policy)

E267

Message: Range index out of bound

```
module top(clk, reseta, resetb, a, b);
input clk, reseta, resetb;
input [2:1]a;
output [2:1]b;
reg [2:1]b;

always @ (posedge a[1:0])
```

Design with Verilog

3. Lint

* Lint rule examples (Leda Verilint policy)

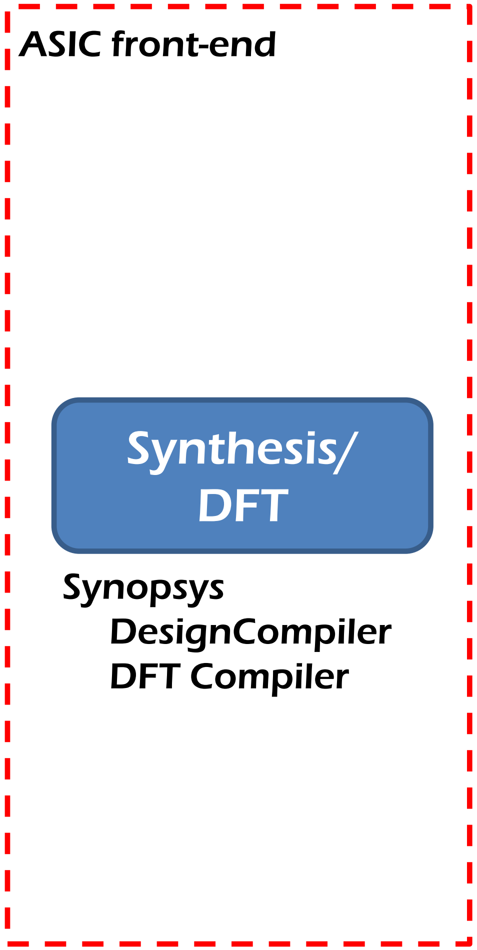
W69

Message: Case statement without default clause but all the cases are covered

Description	Leda fires for this rule when it finds a case statement that has no default clause, but which appears to cover all cases. Even if all cases that have 1's and 0's are covered, there may be others that are not covered. A default clause can cover these additional cases.
Policy	VERILINT
Ruleset	CHECKER_ERROR
Language	Verilog
Type	Block-level
Severity	Warning

ASIC Front-end

1. Now, synthesizable. Is it enough?



* Design constraints

- Timing

  Does the operation speed of the design meet the spec?

- Power

  Does the power consumption of the design meet the spec?

- Area

  Does the design fit for the size of product?

* DFT rule check

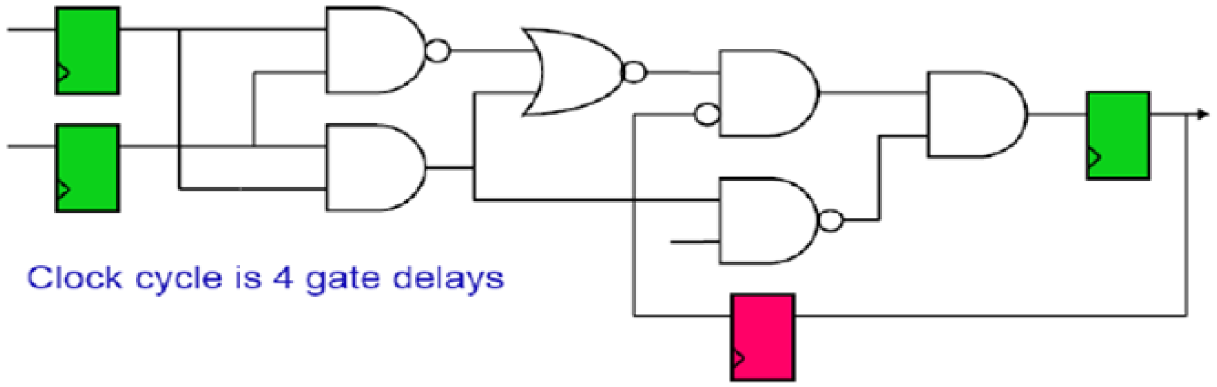
- How to identify the defect caused by manufacturing process

ASIC Front-end

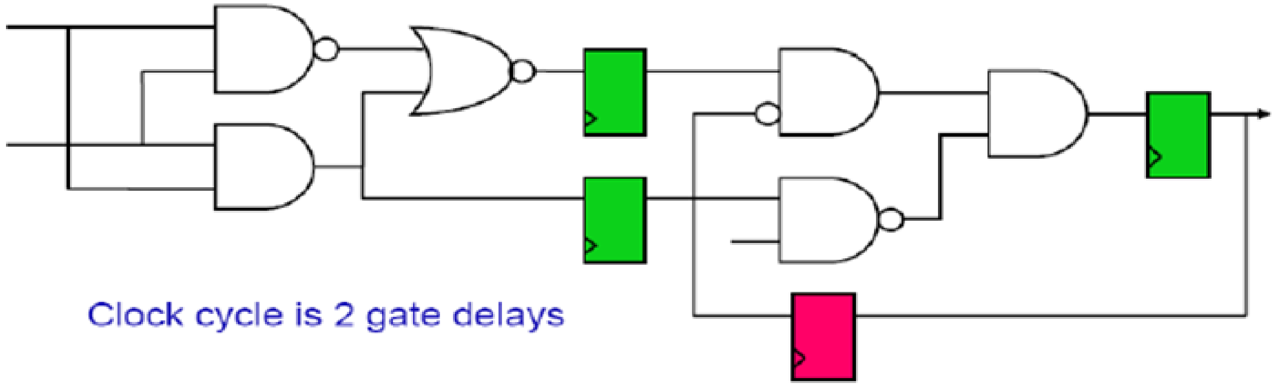
2. Pipelined design

* Pipelined RTL

  - Insert the additional F/Fs into the data path



Clock cycle is 4 gate delays



Clock cycle is 2 gate delays

ASIC Front-end

3. Low power design

* Dynamic power

- Power dissipation in a CMOS transistor depends on the capacitance, supply voltage and the rate at which the data toggles

$$P = f\, C_{load}\, V_{DD}^2$$

- C_{load} is the load capacitance of the CMOS transistor
- V_{DD} is the supply voltage
- f is the frequency at which the data transition happens

- *An* efficient and high quality HDL code can reduce unwanted transitions

ASIC Front-end

3. Low power design

* Minimizing data transitions on bus

- The data on the bus keeps on transitioning from one value to another because there is no default state for assigning a constant value

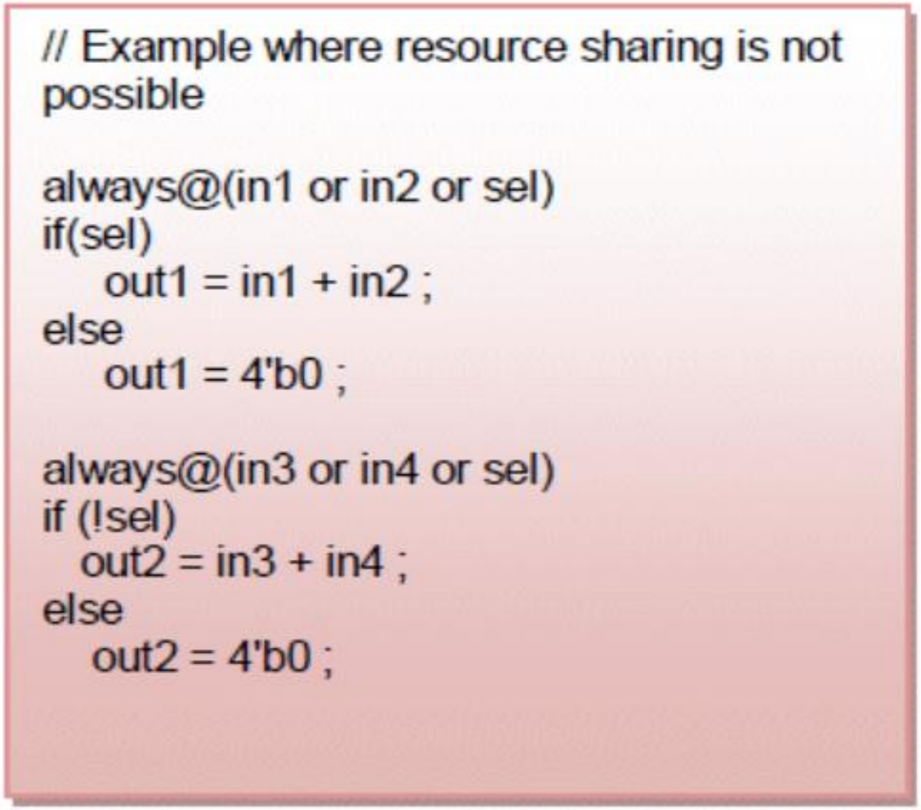
```
// Code that resets the bus to default
status after valid gets de-asserted

always@(posedge clk or negedge reset)

begin
    if(!reset)
    data_bus  <=  16'b0    ;
    else if (data_bus_valid)
    data_bus  <=  data_o ;
    else
    data_bus   <=  16'b0    ;
end
```

```
// Code that holds the bus to its previous
value after valid gets de-asserted

always@(posedge clk or negedge reset)

begin
    if(!reset)
    data_bus  <=  16'b0    ;
    else if (data_bus_valid)
    data_bus  <=  data_o ;
end
```

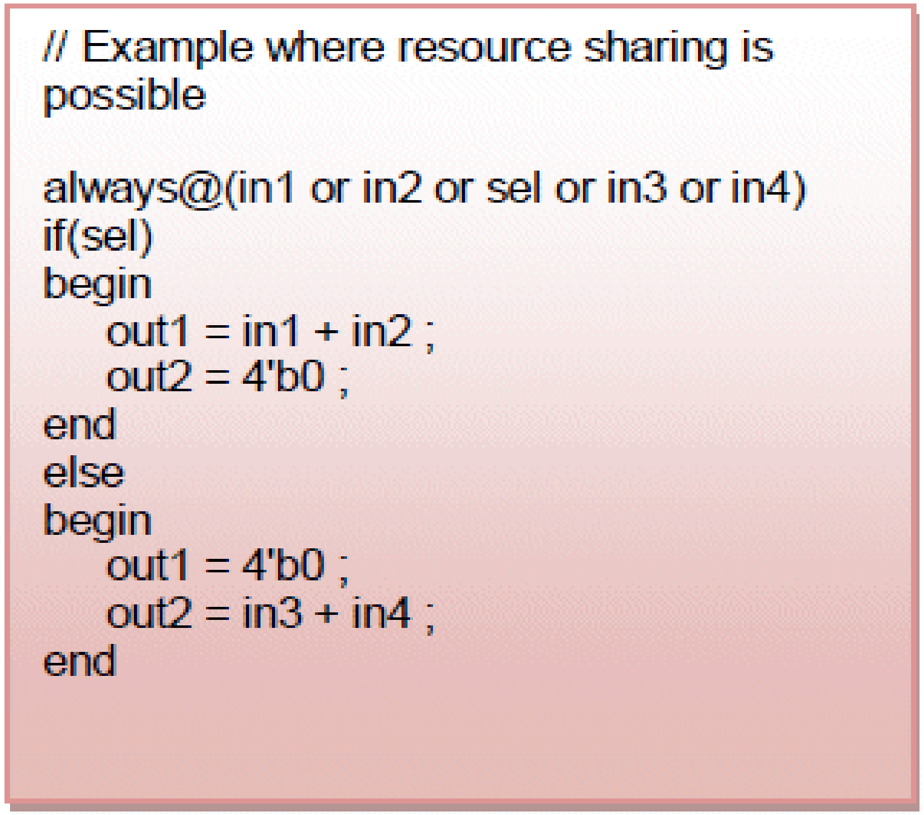
3. Low power design

* Resource sharing

- The RTL coding should be carried out in a manner that there are no unwanted or redundant logic elements



```
// Example where resource sharing is not
possible

always@(in1 or in2 or sel)
if(sel)
     out1 = in1 + in2 ;
else
     out1 = 4'b0 ;

always@(in3 or in4 or sel)
if (!sel)
   out2 = in3 + in4 ;
else
    out2 = 4'b0 ;
```



```
// Example where resource sharing is
possible

always@(in1 or in2 or sel or in3 or in4)
if(sel)
begin
     out1 = in1 + in2 ;
     out2 = 4'b0 ;
end
else
begin
     out1 = 4'b0 ;
     out2 = in3 + in4 ;
end
```

ASIC Front-end

3. Low power design

* Avoiding unnecessary transition of signal

- It is seen in many designs that certain signals transit when they are not required to, but they are not detected in functional verification, as they satisfy the logical requirements.

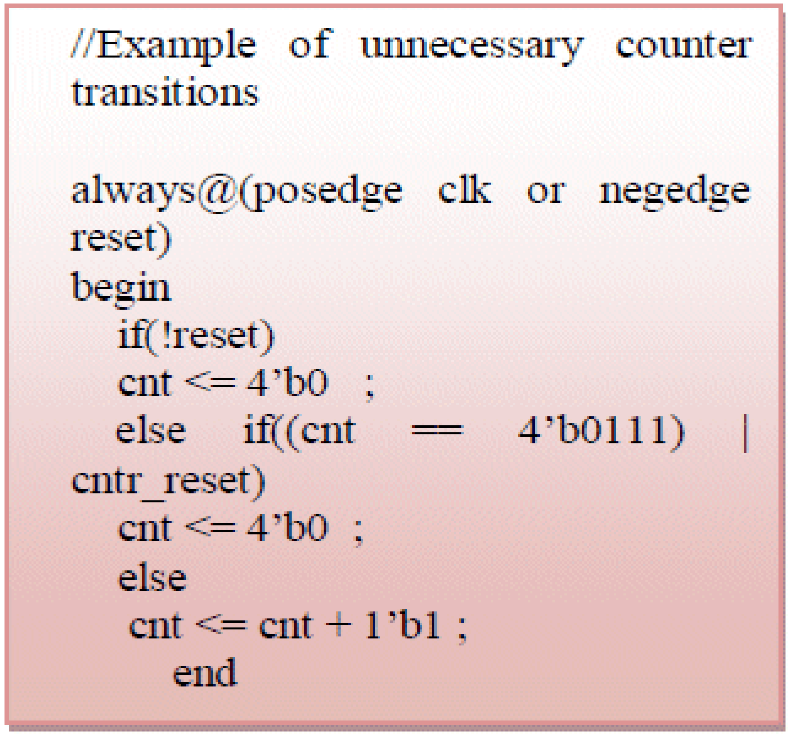
* State Machine Encoding

- It is a well known fact that one-hot and Gray encoding consume lesser power as compared to binary encoding

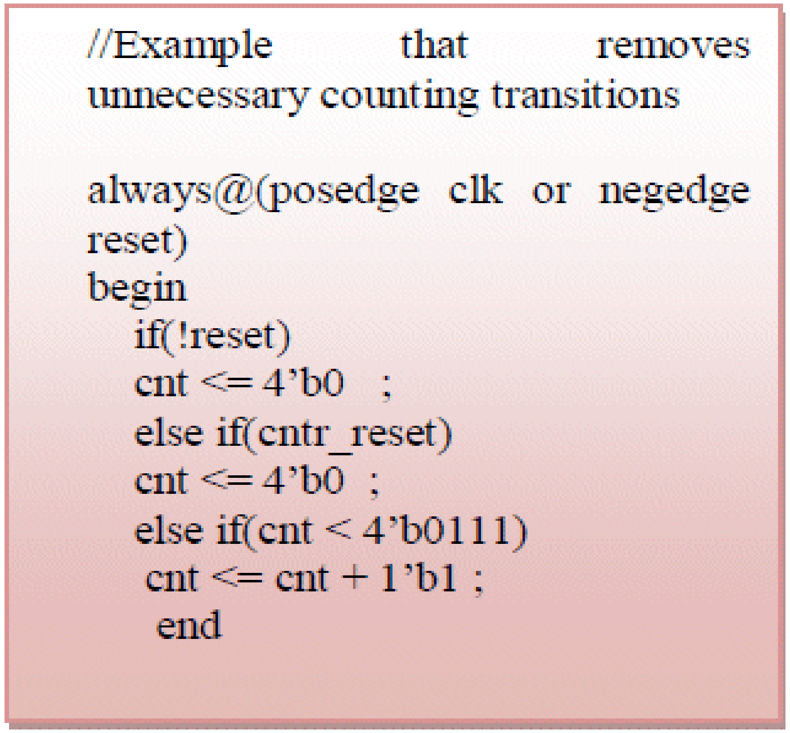
3. Low power design

* Control over counters

- Due to improper coding, all the start and stop conditions are not taken care of and the counter may unnecessarily keep on counting



```
//Example of unnecessary counter
transitions

always@(posedge clk or negedge
reset)
begin
    if(!reset)
    cnt <= 4'b0   ;
    else   if((cnt   ==   4'b0111)   |
cntr_reset)
    cnt <= 4'b0  ;
    else
     cnt <= cnt + 1'b1 ;
        end
```



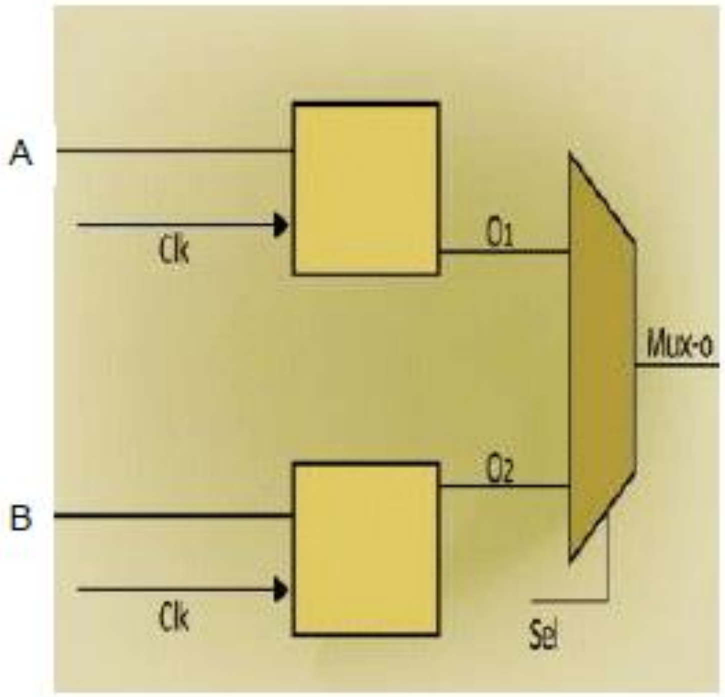
```
//Example that removes
unnecessary counting transitions

always@(posedge clk or negedge
reset)
begin
    if(!reset)
    cnt <= 4'b0   ;
    else if(cntr_reset)
    cnt <= 4'b0  ;
    else if(cnt < 4'b0111)
     cnt <= cnt + 1'b1 ;
      end
```

ASIC Front-end

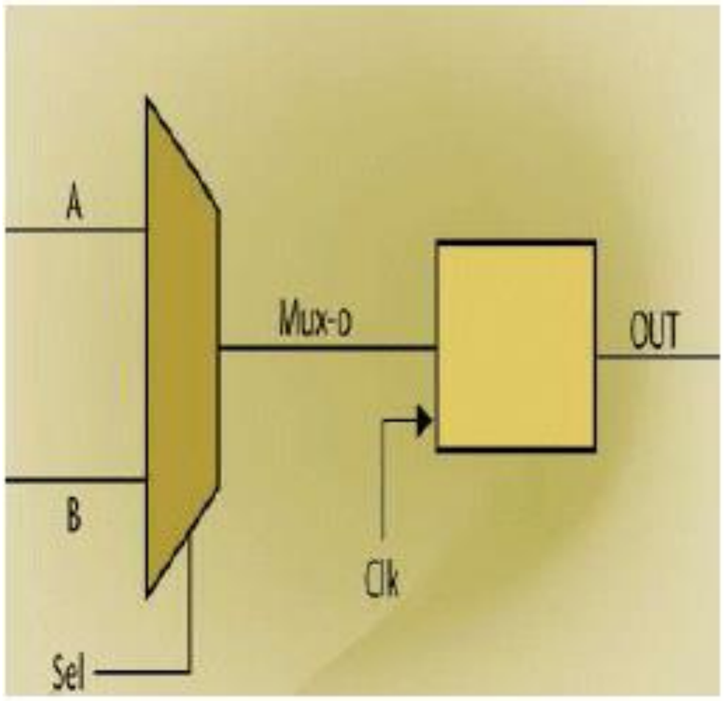
3. Low power design

* Register retiming

- There is a saving of logic and thus can help improve upon power consumption



Without Retiming



With Retiming

ASIC Front-end

3. Low power design

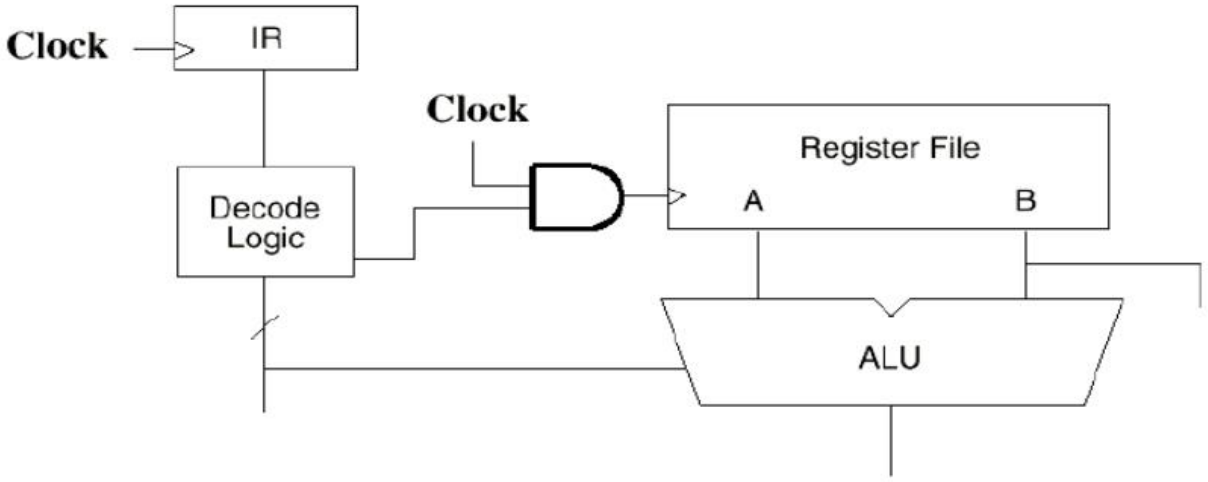
* Clock gating

- Clock

  Highest transition probability

  Long lines and interconnections

  Consumes a significant fraction of power (sometimes more than 40% if not guarded)

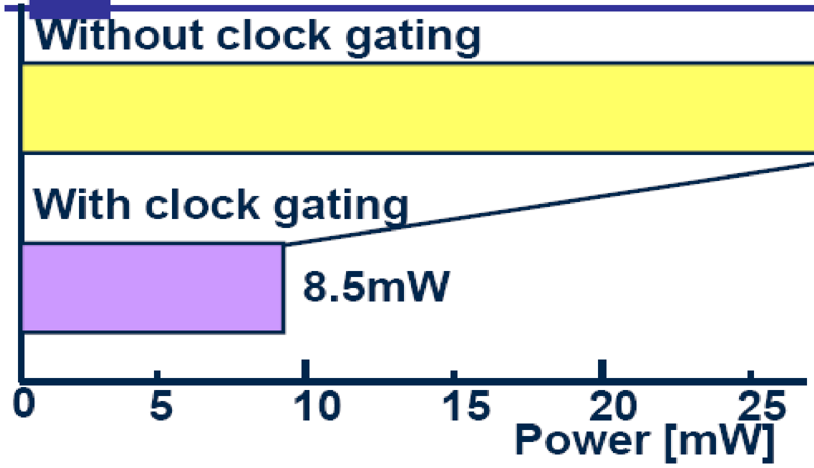


3. Low power design

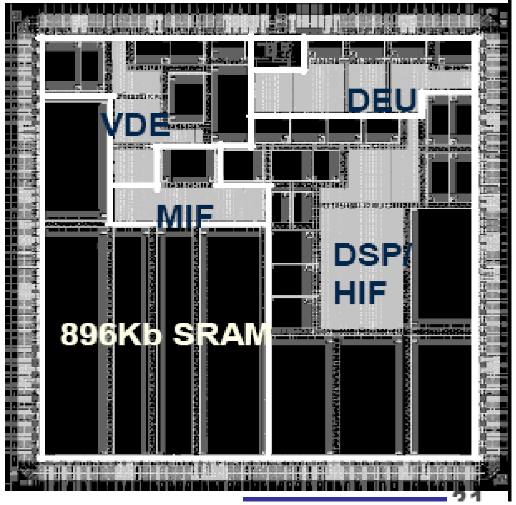
* Clock gating

  - Gate the clock if is not needed

Clock-gating efficiently reduces power



MPEG4 decoder



90% of F/F's were clock-gated.

70% power reduction by clock-gating alone.

Courtesy M. Ohashi, Matsushita, ISSCC 2002

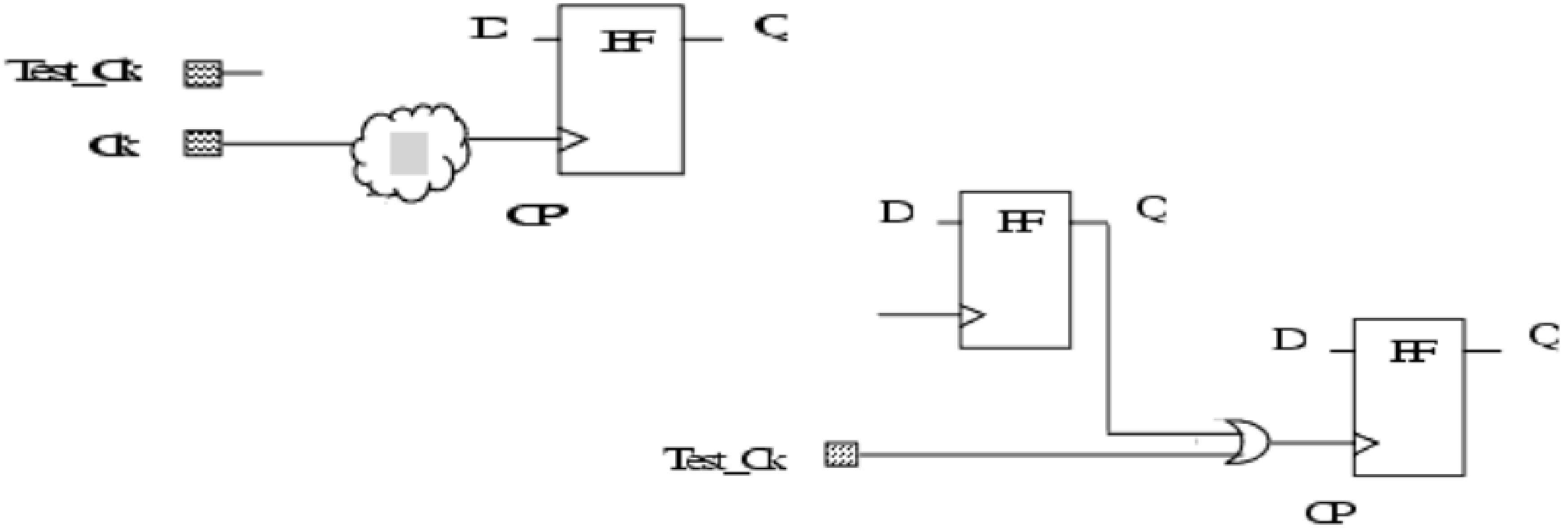
ASIC Front-end

4. DFT

* DFT mandatory rules (Synopsys Tetramax)

- D1 : Clock of F/F cannot be controlled

By inserting multiplexer with scan clock from outside of the design, D1 rule can be fixed



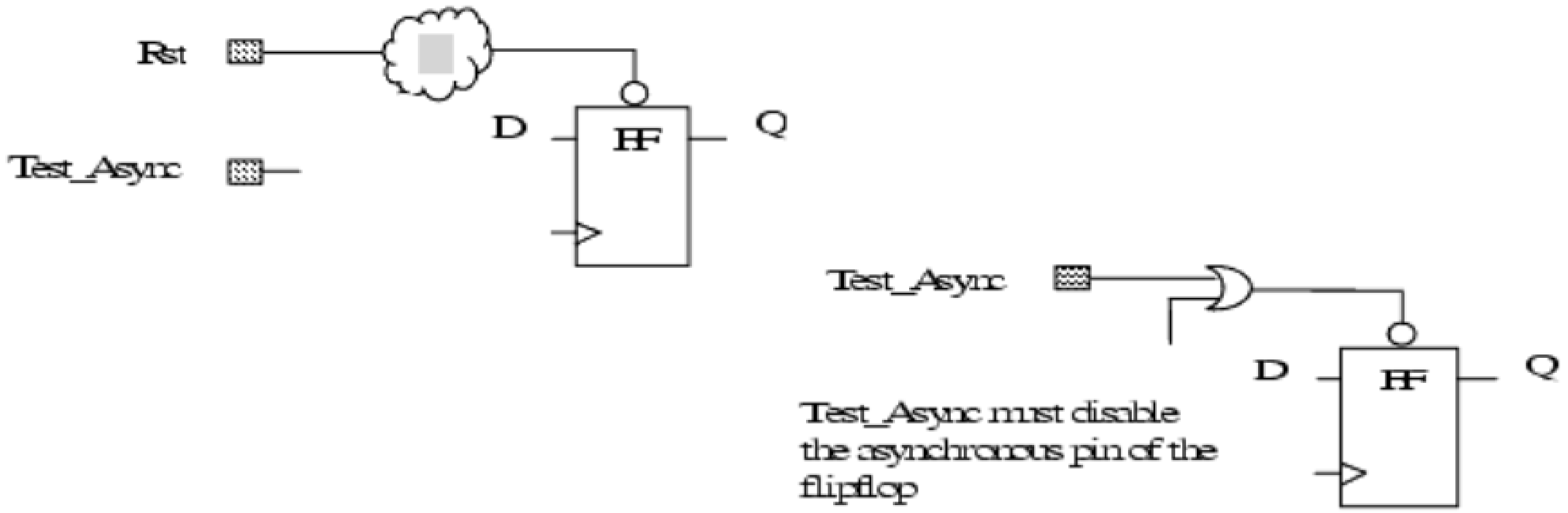
ASIC Front-end

4. DFT

* DFT mandatory rules (Synopsys Tetramax)

- D2/D3 : Reset/Set of F/F cannot be controlled

  By inserting multiplexer with external reset from outside of the design, D2 and D3 can be fixed


Rst
Test_Async
D
FF
Q
Test_Async
D
FF
Q
Test_Async must disable the asynchronous pin of the flipflop

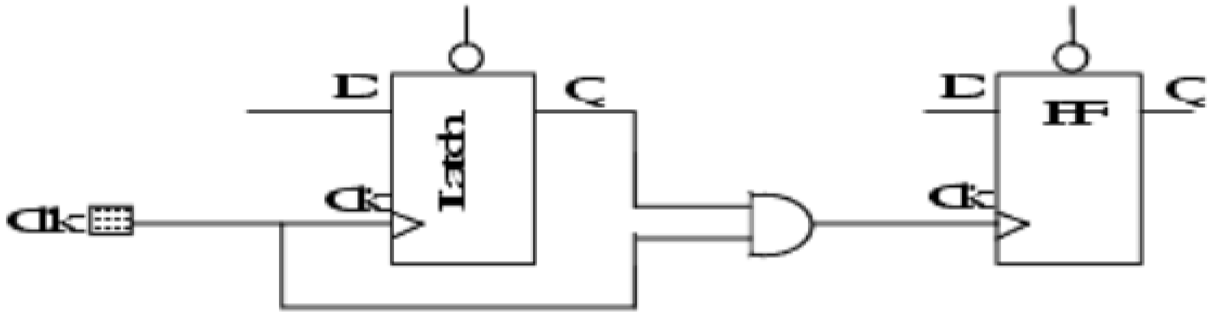
4. DFT

* DFT mandatory rules (Synopsys Tetramax)

- D9 : Clock gating logic is not identified – clock of F/F cannot be controlled

Clock gating identification is only way to fix this violation

Unrecognizable clock gating logic have to be removed



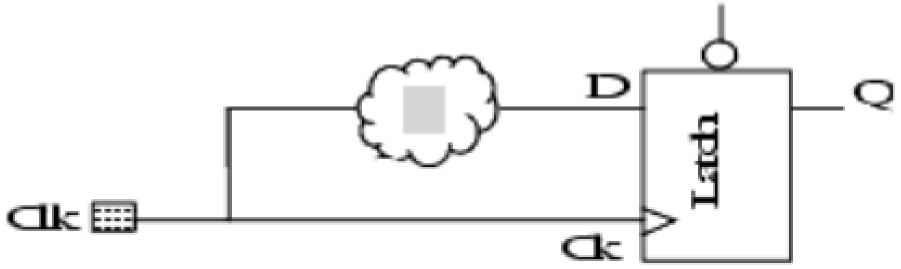
ASIC Front-end

4. DFT

* DFT mandatory rules (Synopsys Tetramax)

- D11 : race condition between clock and data input is occurred

  This violation is caused by F/F which use scan test clock as its data input signal



Wrap-up

For design quality assurance

* Functional verification

- Code coverage
- Lint

* Design constraints verification

- Timing
- Power
- Area

* DFT

ASIC 개발과 VERILOG HDL 파트 2

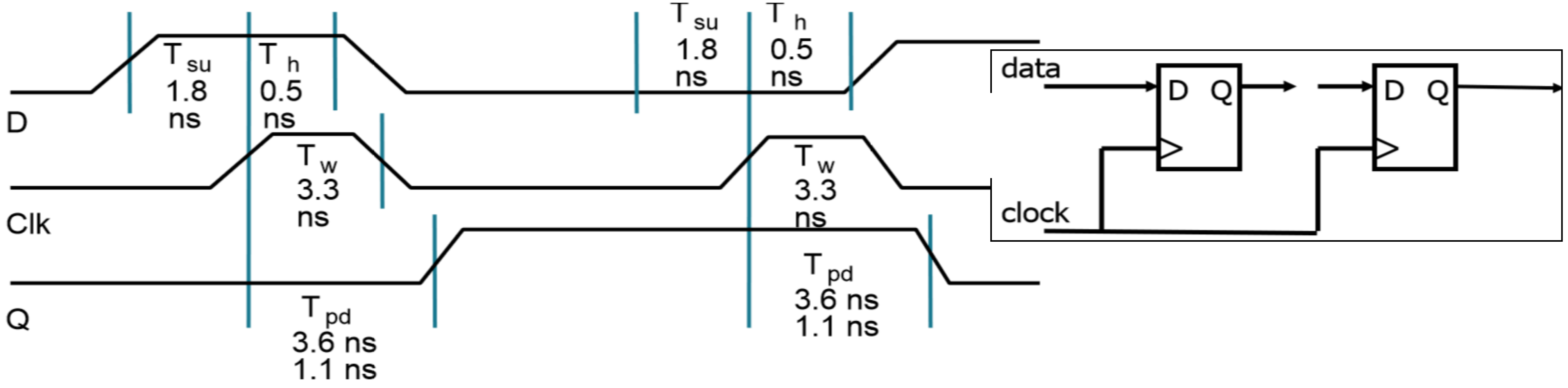
반도체설계교육센터
IC DESIGN EDUCATION CENTER

Synchronization

1. Non-ideal FF behavior

* Timing factors

- Setup time : minimum time before the clocking event by which the input must be stable (T_{su})
- Hold time : minimum time after the clocking event until which the input must remain stable (T_h)
- Propagation delay : amount of time for value to propagate from input to output (T_{pd})

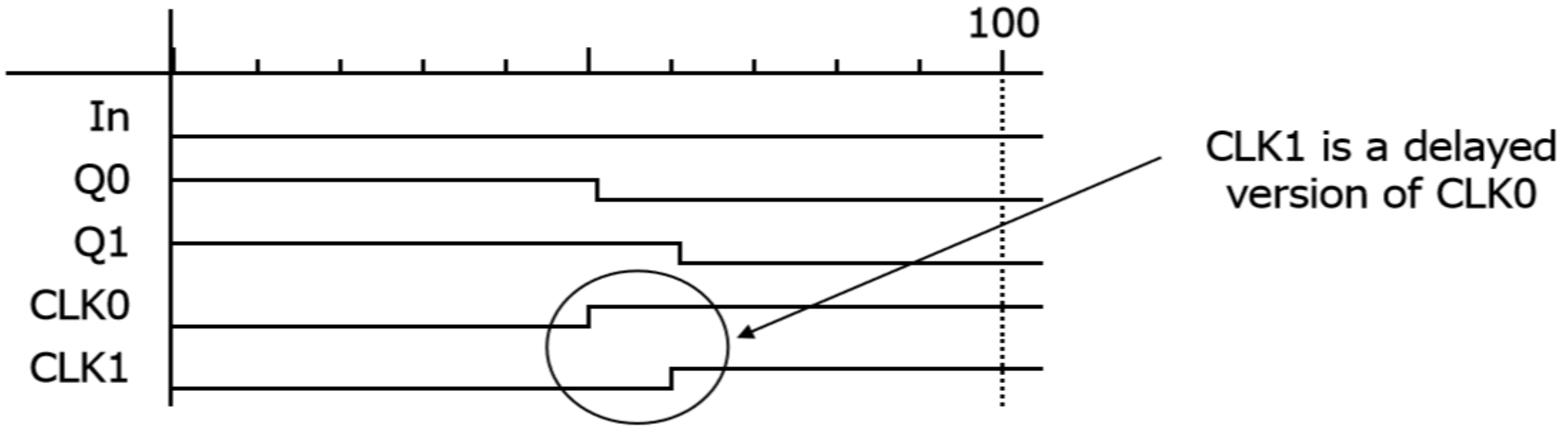


Synchronization

2. Clock skew

* The problem

- Correct behavior assumes next state of all storage elements determined by all storage elements at the same time
- This is difficult in high-performance systems because time for clock to arrive at flip-flop is comparable to delays through logic
- Effect of skew on cascaded flip-flops:



Synchronization

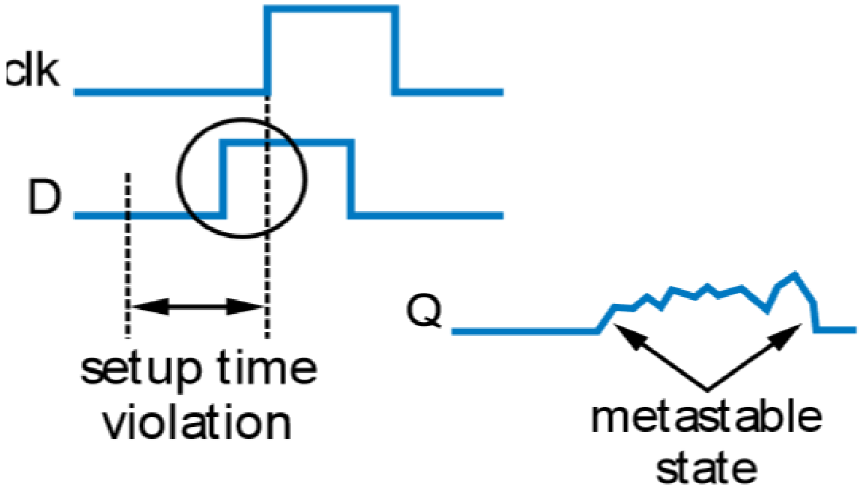
3. Metastability

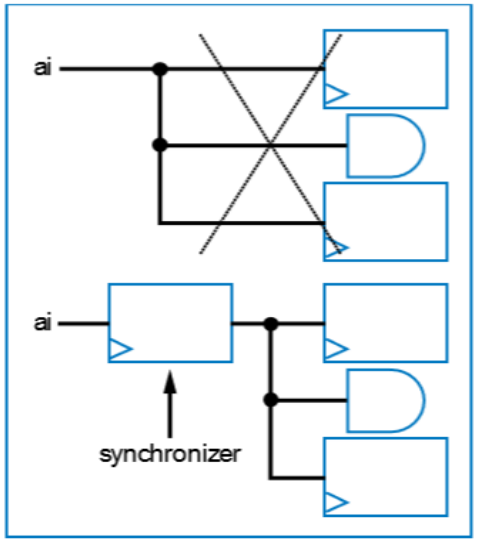
* Definition

- Violating setup/hold time can lead to a metastable state
- Any flip-flop state other than a stable 1 or 0
- Eventually settles to one or other, but we don't know which

* Solution

- Insert synchronizer flip-flop for asynchronous input





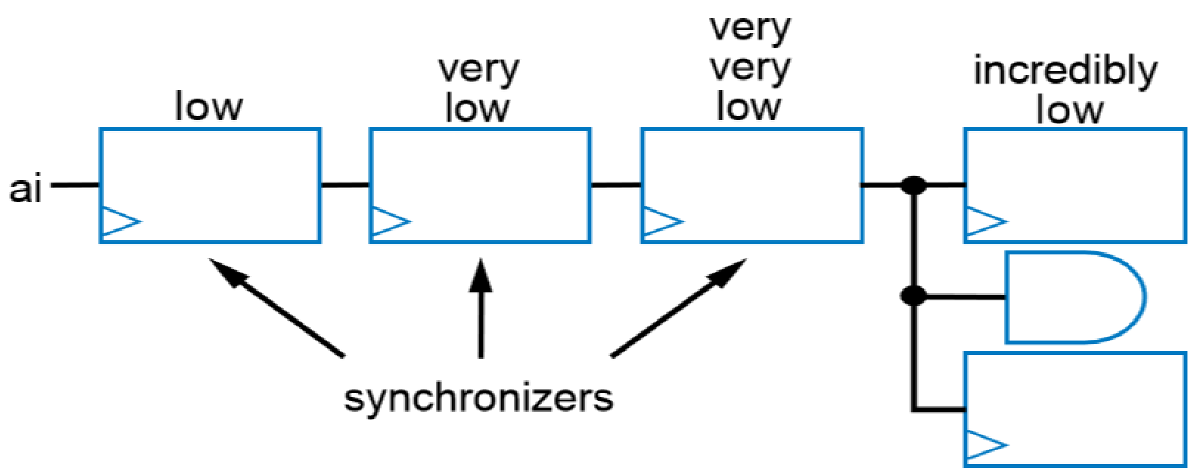
Synchronization

3. Metastability

* More than double flip-flop

- One flip-flop doesn't completely solve problem
- Add more synchronizer flip-flops to decrease the probability of metastability
- Can't solve completely
- just decrease the likelihood of failure

Probability of flip-flop being metastable is…

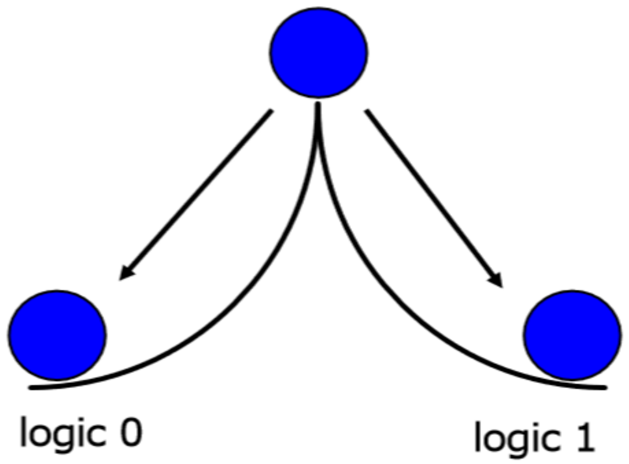


Synchronization

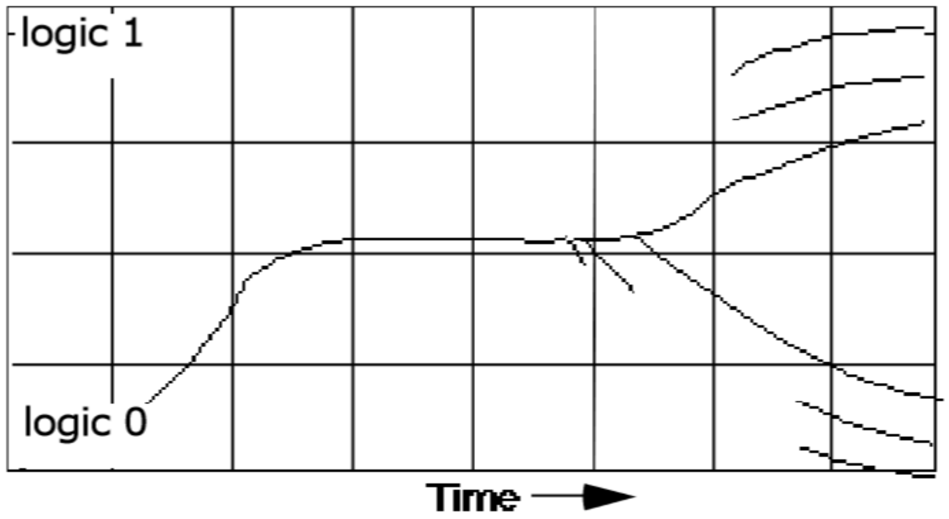
4. Synchronization failure

* Occurs when FF input changes close to clock edge

- The FF may enter a metastable state : Neither a logic 0 nor 1
- It may stay in this state an indefinite amount of time
- This is not likely in practice but has some probability



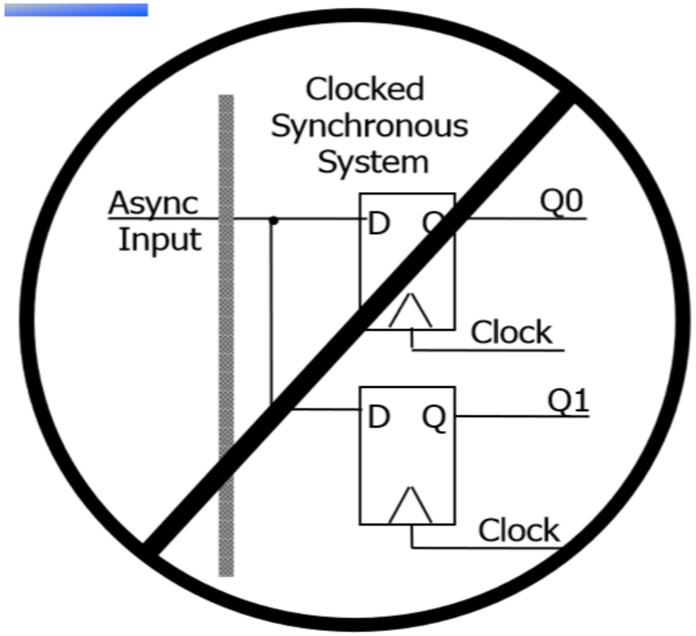
small, but non-zero probability that the FF output will get stuck in an in-between state

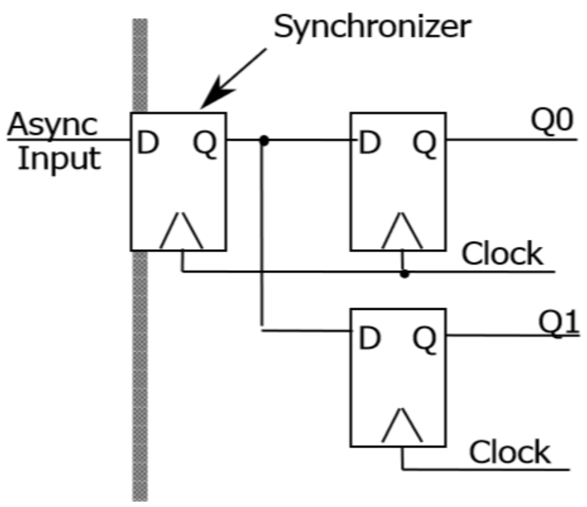


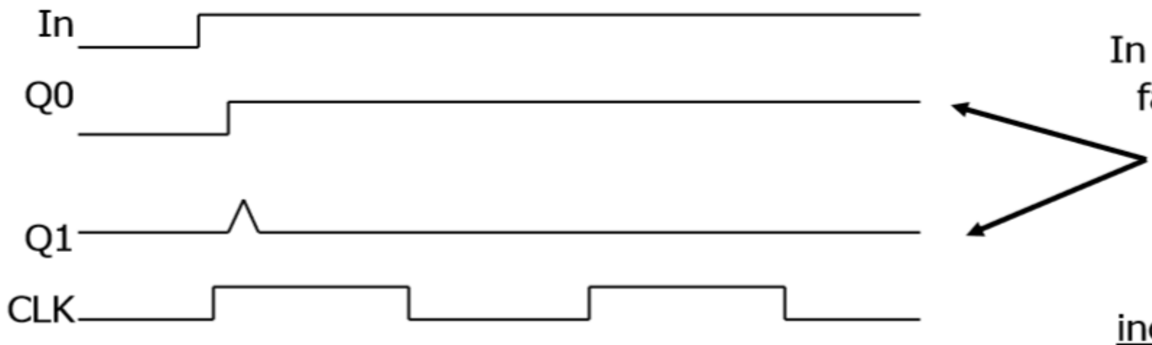
oscilloscope traces demonstrating synchronizer failure and eventual decay to steady state

Synchronization

5. Handling asynchronous inputs







In is asynchronous and fans out to D0 and D1

one FF catches the signal, one does not

<u>inconsistent state may be reached!</u>

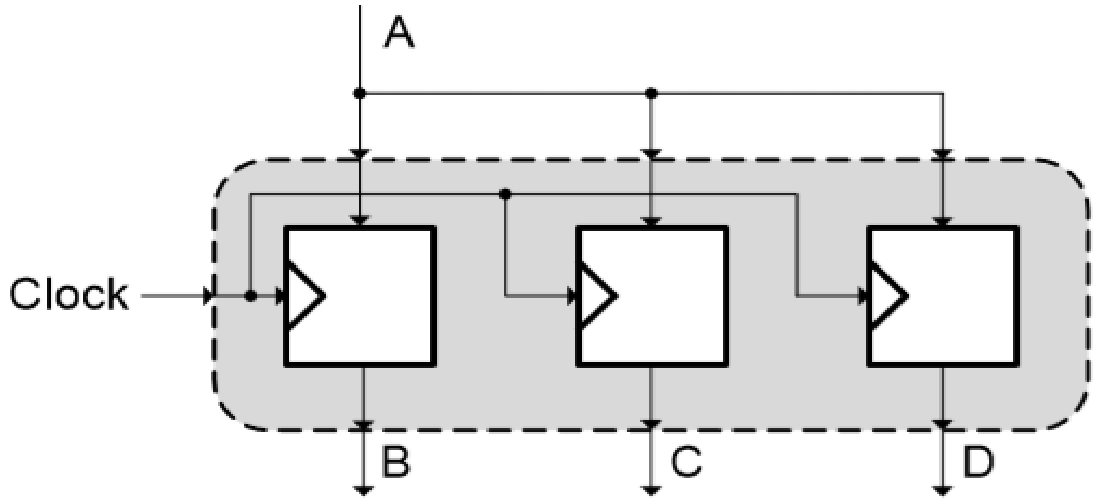
Assignment

1. Blocking vs. Non-blocking

* Blocking (=) assignment

- Blocking assignments happen sequentially
- Blocking assignments are used when specifying combi. logics

```
always @(posedge Clock) begin
    B = A;
    C = B;
    D = C;
end
```



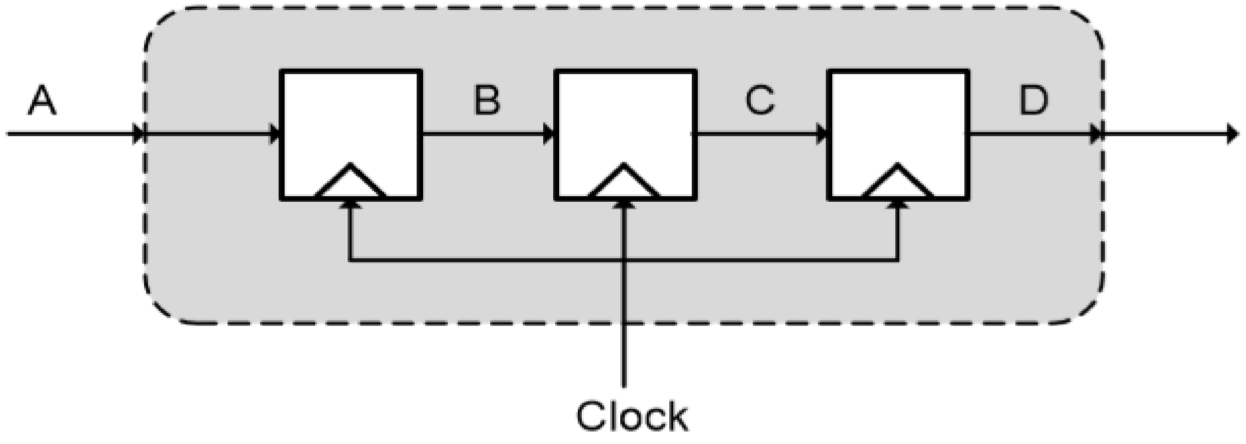
Assignment

1. Blocking vs. Non-blocking

* Non-blocking (<=) assignment

- Non-blocking assignments happen in parallel
- Non-blocking assignments are used when specifying seq. logics

```
always @(posedge Clock) begin
    B <= A;
    C <= B;
    D <= C;
end
```



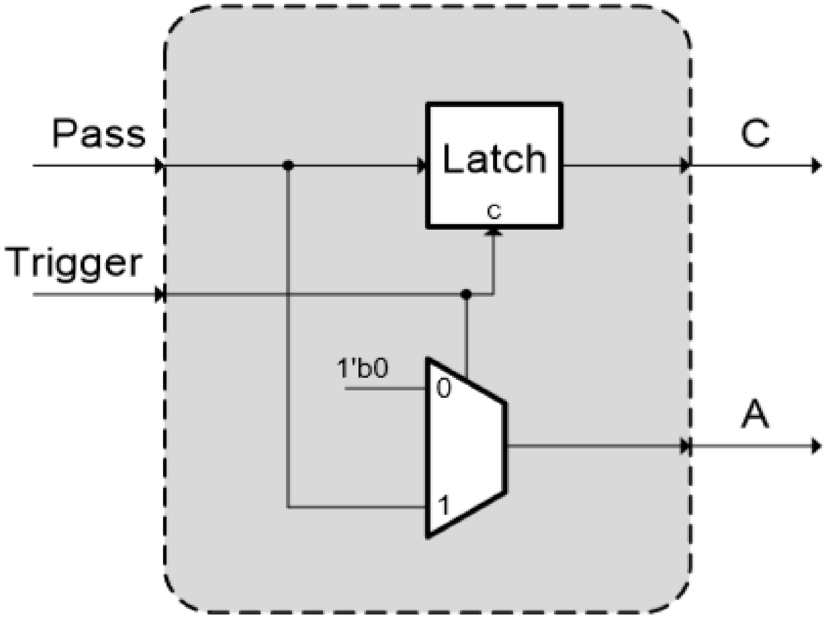
Latch-free Design

1. Latch generation

* Incomplete assignment

- If you don't assign every element that can be assigned inside an always@(*) block every time, a latch will be inferred
- C on the other hand is not always assigned
- As such, a latch is inferred for C

```
wire Trigger, Pass;
reg A, C;

always @( * ) begin
    A = 1'b0;
    if (Trigger) begin
        A = Pass;
        C = Pass;
    end
end
```



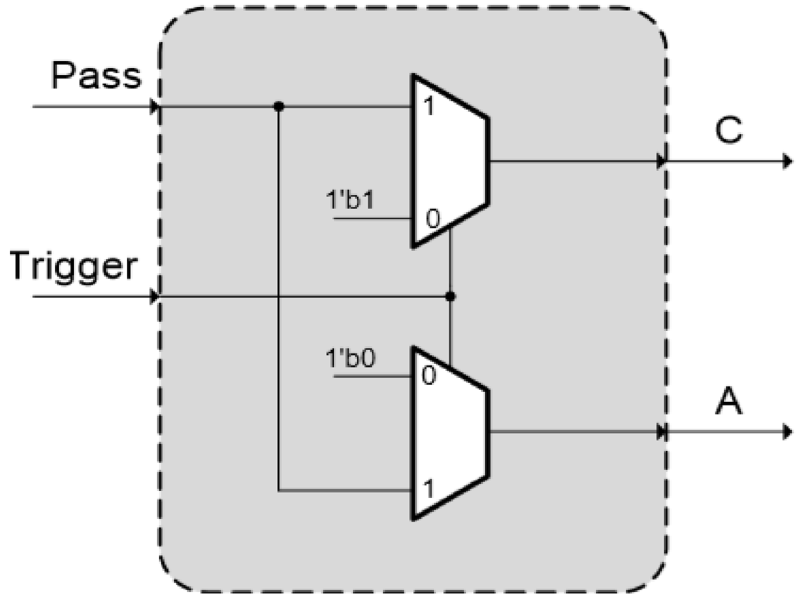
Latch-free Design

2. Latch-free

* Complete assignment

- Default values are an easy way to avoid latch generation, however, will sometimes break the logic in a design
- Typically, they involve proper use of the Verilog else statement, and other flow constructs.

```
wire Trigger, Pass;
reg A, C;

always @( * ) begin
    A = 1'b0;
    C = 1'b1;
    if (Trigger) begin
        A = Pass;
        C = Pass;
    end
end
```



Wrap-up

Advanced design issues

* Synchronization

- Clock skew
- Metastability
- Asynchronous input handling

* Assignment

- Blocking assignment
- Non-blocking assignment

* Latch-free design

ASIC Design Flow 실습 - 합성

반도체설계교육센터
IC DESIGN EDUCATION CENTER

Synthesis – Design Compiler

1. Target Library 설정

* Target library

- tc6a_cbacore.db: SYNOPSYS에서 제공하는 라이브러리
- target_library: 합성 시에 사용할 라이브러리
- link_library: RTL 코드 상에서 직접 instantiation하는 라이브러리 cell에 대해 link 과정에서 인식할 수 있도록 하는 라이브러리

* 실습

```
dc_shell> set target_library “tc6a_cbacore.db“
dc_shell> set link_library “tc6a_cbacore.db“
```

Synthesis – Design Compiler

2. Design 준비

* Working space

  - Design 분석 과정에서 발생하는 출력 파일이 저장될 working space 설정

  - 기존 작업 내용과 충돌 방지하기 위해 디렉토리 삭제 후 재생성

* 실습

```
dc_shell> sh rm –rf .template
dc_shell> sh mkdir .template
dc_shell> define_design_lib WORK –path .template
```

Synthesis – Design Compiler

2. Design 준비

* Design 파일 읽기

  - Design 파일 읽기: *analyze* command 사용, 읽어오는 순서는 상관없음

* 실습

```
dc_shell> analyze –format verilog
                                        ../01_RTL/src/rtl/verilog/stack_top.v

==========================================

dc_shell> source
          /home/student40/100_EXPR/02_SYN/import_design.src
```

Synthesis – Design Compiler

2. Design 준비

* Design Elaboration & Link

- Design hierarchy를 구축하는 elaboration 및 link 수행
- 두 번 이상 instantiation 되는 module에 대해 별도의 module로 선언하는 uniquify 작업도 수행됨

* 실습

```
dc_shell> elaborate RISC_CORE
dc_shell> current_design RISC_CORE
dc_shell> link
```

Synthesis – Design Compiler

3. Design Constraints 준비

* Timing constraints 정의

- Clock을 정의: create_clock, create_generated_clock

* 실습

```
dc_shell> create_clock –name “Clk” –period 7 –waveform {0 3.5} [get_ports “Clk”]
dc_shell> set_don’t_touch_network [get_clocks “Clk”]
dc_shell> set_input_delay 4.4 –max –rise –clock “Clk” [get_ports “Instrn[*]”]
dc_shell> set_input_delay 4.5 –max –fall –clock “Clk” [get_ports “Instrn[*]”]
dc_shell> set_input_delay 4.4 –max –rise –clock “Clk” [get_ports “Reset”]
dc_shell> set_input_delay 4.3 –max –fall –clock “Clk” [get_ports “Reset”]
dc_shell> set_output_delay 4.3 –max –fall –clock “Clk” [all_outputs]
dc_shell> set_clock_uncertainty 1.0 –setup [get_clocks “Clk”]
dc_shell> set_clock_uncertainty 0.055 –hold [get_clocks “Clk”]
dc_shell> set_false_path –to [get_cells “I_ALU/Neg_Flag_reg”]
dc_shell> set_false_path –to [get_cells “I_ALU/Zro_Flag_reg”]
dc_shell> set_wire_load_mode “enclosed”
```

Synthesis – Design Compiler

4. Design 합성

* Design 합성 수행

- compile_ultra
- Design을 Target library로 mapping, timing constraints, area constraints, power constraints를 최적화하는 작업을 반복함

* 실습

```
dc_shell> compile_ultra –scan –no_autoungroup
                    –no_seq_output_inversion –no_boundary_optimization
```

Synthesis – Design Compiler

5. Design 합성 결과 확인하기

* Design constraints 사항 확인

- Timing violation: report_timing
- Area overhead: report_area
- Power overhead: report_power

* 실습

```
dc_shell> report_timing –nworst 2 –delay_type max
dc_shell> report_timing –nworst 2 –delay_type min
dc_shell> report_area –hier
dc_shell> report_power –hier -verbose
```

Synthesis – Design Compiler

6. Design 합성 결과 파일 생성하기

* 합성 결과 파일 생성

- Gate-level netlist, DDC, SDF, SDC 파일 등 생성

* 실습

```
dc_shell> write –f verilog –hier –output RISC_CORE_gate.v
dc_shell> write_sdf –v 2.1 –context verilog RISC_CORE_gate.sdf
```

ASIC Design Flow 실습 - LEC

반도체설계교육센터
IC DESIGN EDUCATION CENTER

Logic Equivalent Check – Formality

1. 환경 설정

* LEC 수행 환경 조건 설정

- Design elaboration error waive하기 위한 setting
- Design elaboration을 위한 library 지정

* 실습

```
fm_shell> set hdlin_warn_on_mismatch_message “FMR_ELAB-146
                                                  FMR_ELAB-147 FMR_ELAB-149”
fm_shell> read_db
        /tools/synopsys/design_compiler/2014_09/libraries/tc6a_cbacore.db
```

Logic Equivalent Check – Formality

2. Design 설정

* RTL Design 설정

- Golden. Formality에서는 reference라 명함

* 실습

```
fm_shell> read_verilog –libname WORK –c r
              /home/student40/100_EXPR/01_RTL/src/rtl/verilog/stack_top.v
.....
fm_shell> set_top r:/WORK/RISC_CORE
```

Logic Equivalent Check – Formality

2. Design 설정

* Gate-level Design 설정

- Revised. Formality에서는 implementation이라 명함

* 실습

```
fm_shell> read_verilog –libname WORK –c i ./RISC_CORE_gate.v
fm_shell> set_top i:/WORK/RISC_CORE
```

Logic Equivalent Check – Formality

3. LEC 수행

* Verification 수행 및 결과확인

- Reference design과 implementation design 설정 후 검증 수행
- 수행 결과가 non-equivalent인 경우 원인 파악 및 해결 과정 필요

* 실습

```
fm_shell> set_reference_design r:/WORK/RISC_CORE
fm_shell> set_reference_design i:/WORK/RISC_CORE
fm_shell> verify r:/WORK/RISC_CORE  i:/WORK/RISC_CORE
```

Logic Equivalent Check – Formality

4. Non-equivalent 원인 해결 과정

* GUI를 통한 원인 파악

- Register merge, constant register deletion 등에 의해 unmapped point 발생하는 경우가 많음
- 합성 제약 조건 부여 또는 LEC 상에서 design에 대한 정보 반영하여 해결 가능

* 실습

fm_shell> set_svf [glob ./top.svf] => Design compiler에서 생성

or

fm_shell> guide

fm_shell> guide_reg_merging –design INSTRAN_LAT –from {Crnt_Instrn_1_reg[0] Crnt_Instrn_2_reg[0]} –to {Crnt_Instrn_2_reg[0]}

fm_shell> setup

ASIC Design Flow 실습 - DFT

반도체설계교육센터
IC DESIGN EDUCATION CENTER

DFT – Design Compiler

1. Target Library 설정

* Target library

- tc6a_cbacore.db: SYNOPSYS에서 제공하는 라이브러리
- target_library: 합성 시에 사용할 라이브러리
- link_library: RTL 코드 상에서 직접 instantiation하는 라이브러리 cell에 대해 link 과정에서 인식할 수 있도록 하는 라이브러리

* 실습

```
dc_shell> set target_library “tc6a_cbacore.db“
dc_shell> set link_library “tc6a_cbacore.db“
```

DFT – Design Compiler

2. Design 준비

* Working space

- DFT 과정에서 발생하는 출력 파일을 저장할 reporting space 설정
- 기존 작업 내용과 충돌 방지하기 위해 디렉토리 삭제 후 재생성

* 실습

```
dc_shell> sh rm –rf report
dc_shell> sh mkdir report
```

DFT – Design Compiler

2. Design 준비

* Design 파일 읽기

  - Gate-level Design Reading: read_verilog -netlist

* 실습

```
dc_shell> read_verilog –netlist ./RISC_CORE_gate.v
dc_shell> current_design RISC_CORE
dc_shell> link
```

DFT – Design Compiler

3. DFT constraints 정의

* DFT configuration 정의

- Scan test configuration

* 실습

```
dc_shell> set_scan_configuration –chain_count 4
dc_shell> set_scan_configuration –add_lockup true
dc_shell> set_scan_configuration –lockup_type latch
dc_shell> set_scan_configuration –clock_mixing mix_clocks
dc_shell> set_scan_configuration –create_dedicated_scan_out_ports true
dc_shell> set_scan_configuration –preserve_multibit_segment true
dc_shell> set_scan_configuration –style multiplexed_flip_flop
dc_shell> set_scan_configuration –replace true
```

DFT – Design Compiler

3. DFT constraints 정의

* DFT configuration 정의

- DFT signal 정의

* 실습

```
dc_shell> create_port scan_mode
dc_shell> set_dft_signal –view spec –port scan_mode –type TestMode
                                                          –active 1
dc_shell> set_dft_signal –view exist –port Reset –active 1
dc_shell> set_dft_signal –view exist –port ScanClock –timing [list 45 55]
```

DFT – Design Compiler

4. Pre-DFTDRC 진행

* DFT rule checking

- Pre-DFT DRC는 design내의 모든 flip-flop에 대해 scan chain에 삽입하는 것이 가능한지 검사

* 실습

```
dc_shell> create_test_protocol –capture_procedure single_clock
dc_shell> dft_drc –verbose –pre_dft > report/drc_pre_insert.dft
```

DFT – Design Compiler

5. DFT insertion 진행

* DFT insertion

  - Pre-DFT DRC에 대한 검사 및 해결 후 design 상에 scan chain을 삽입하는 단계

  - DFT 삽입 후 post-DFT DRC 진행

* 실습

```
dc_shell> preview_dft –test_points all –show all > report/preview_dft.log
dc_shell> insert_dft
dc_shell> dft_drc –verbose > report/insert_drc.dft
```

DFT – Design Compiler

6. DFT 결과 파일 생성

* DFT 결과 파일 reporting
  - DFT 삽입된 후의 gate-level netlist
  - Scan chain 정보를 갖는 def 파일
  - ATPG를 위한 테스트 procedure 파일

* 실습

```
dc_shell> write –f verilog –hier –out report/RISC_CORE_after_scan.v
dc_shell> write_scan_def –output report/RISC_CORE_after_scan.def
dc_shell> write_test_protocol –o report/RISC_CORE_after_scan.spf
```

Q & A

Thank you