EECS 2070 02 Fall 2021

Improper Coding Styles

黃稚存

Chih-Tsun Huang

cthuang@cs.nthu.edu.tw

國立清華大學
資訊工程學系

Lecture 09

聲明

- 本課程之內容 (包括但不限於教材、影片、圖片、檔案資料等)，僅供修課學生個人合理使用，非經授課教師同意，不得以任何形式轉載、重製、散布、公開播送、出版或發行本影片內容(例如將課程內容放置公開平台上，如 Facebook, Instagram, YouTube, Twitter, Google Drive, Dropbox 等等)。如有侵權行為，需自負法律責任。

Inferring Latch

- Perhaps the most common mistakes
- Latch is a sequential element
  - In modern synchronous digital designs, latch is not recommended
  - Flip-flop is preferred
- Improper coding style will introduce latches into combinational blocks

Inferred Latches in Combinational Circuits (1/2)

- Incomplete sensitivity list in combinational circuits

```
always @(a or b) begin
  sum = a + b + cin;
end
```

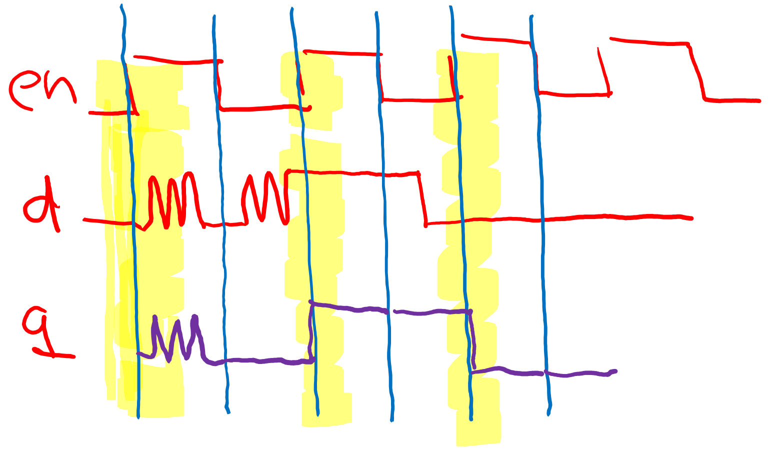
```
always @(b, c) begin
  if (select)  a = b;
  else  a = c;
end
```

- Use `always @*` or `always @(*)` instead

Inferred Latches in Combinational Circuits (2/2)

- Incomplete if-else statement ➔ inferring **latch!**

X
```
always @*
  if (en)  q = d + 1;
```



- Combinational block must have
  - Complete if-else statement

```
always @*
  if (en)  q = d + 1;
  else  q = 0;
```

  - Or always specify a default value to a variable

```
always @* begin
  q = 0;
  if (en)  q = d + 1;
end
```

Incomplete Assignment in Combinational Block

X

```
always @* begin
  if (a > b) begin
    x = c;
    y = 0;
  end else begin
    y = 1;
  end
end
```

```
always @* begin
  if (a > b) begin
    x = c;
    y = 0;
  end else begin
    x = 0;
    y = 1;
  end
end
```

```
always @* begin
  x = 0;
  y = 1;
  if (a > b) begin
    x = c;
    y = 0;
  end else begin
    x = 0;
    y = 1;
  end
end
```

Incomplete **if** Statement

- The priority encoder fails to check for the possibility of no true inputs

```
module priority (
  input [3:0] a,
  output reg [3:0] y);
  always @*

    if      (a[3]) y = 4'b1000;
    else if (a[2]) y = 4'b0100;
    else if (a[1]) y = 4'b0010;
    else if (a[0]) y = 4'b0001;
    // else        y = 4'b0000;
endmodule
```

Warning: Inferred memory devices in process in routine priority line 4 in file ‘priority_decoder.v’.

Another Incomplete **if** Statement

```
always @* begin
  if (enable == 1'b1) begin
    reg_a = A;
    reg_b = B;
  end
  if (swap == 1'b1) begin
    data_a = reg_b;
    data_b = reg_a;
  end else begin
    data_a = reg_a;
    data_b = reg_b;
  end
end
```

Warning: <location>: **inferring latch(es)** for variable "*<name>*", which holds its previous value in one or more paths through the always construct

Incomplete **case** Statement

```
module divideby3FSM(
  input clk,
  input reset,
  output out);
  reg [1:0] state, nextstate;
  parameter S0 = 2'b00;
  parameter S1 = 2'b01;
  parameter S2 = 2'b10;

  // State register
  always @(posedge clk, posedge reset)
    if (reset) state <= S0;
    else state <= nextstate;
```

```
  always @(state) // Next state logic

    case (state)
      S0: nextstate = S1;
      S1: nextstate = S2;
      S2: nextstate = S0;
      // default: nextstate = S0;
    endcase

  // Output logic
  assign out = (state == S2);
endmodule
```

Warning: <location>: **inferring latch(es)** for variable "<*name*>", which holds its previous value in one or more paths through the always construct

Combinational Multi-Way Branch

- Nested if-else statements

```
if (alu_control == 0)
  y = x + z;

else if (alu_control == 1)
  y = x - z;

else if (alu_control == 2)
  y = x * z;

else
  y = 0;
```

- case statements

```
reg [1:0] alu_control;

case (alu_control)
  2'd0: y = x + z;
  2'd1: y = x - z;
  2'd2: y = x * z;
  default: y = 0;
endcase
```

- Can replace nested if-else-if statements
- Can be nested

Complete Assignment for Combinational Circuits

Style A

```
always @* begin
  if (en==1'b1) begin
    if (out == 4'd9) begin
      out_next = 4'b0;
      co = 1'b1;
    end else begin
      out_next = out + 1'b1;
      co = 1'b0;
    end
  end else begin
    out_next = out;
    co = 1'b0;
  end
end
```

Style B

```
always @* begin
  out_next = out;
  co = 1'b0;
  if (en==1'b1) begin
    if (out == 4'd9) begin
      out_next = 4'b0;
      co = 1'b1;
    end else begin
      out_next = out + 1'b1;
    end
  end
end
```

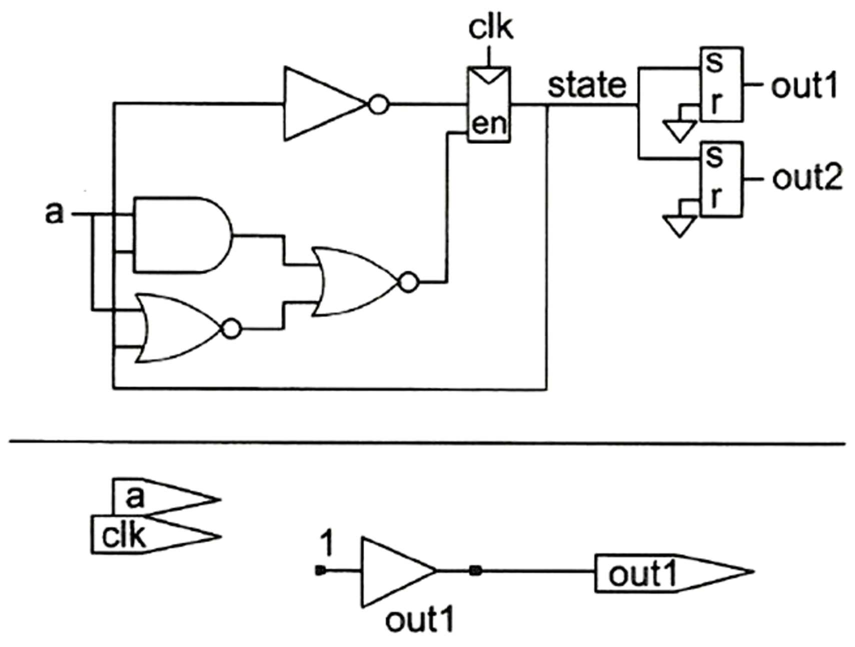
Ex: Ambiguous Output Definition

- out1 is intended to be 1 when state is 0;
  out2 to be 1 when state is 1
  - However, the code never sets the outputs low.

```
reg state, out1, out2;
always @(posedge clk)
  if (state == 0) begin
    if (a) state <= 1;
  end else begin
    if (~a) state <= 0;
  end

always @*
  if (state == 0) out1 = 1;
  else            out2 = 1;
```

Synthesis Results

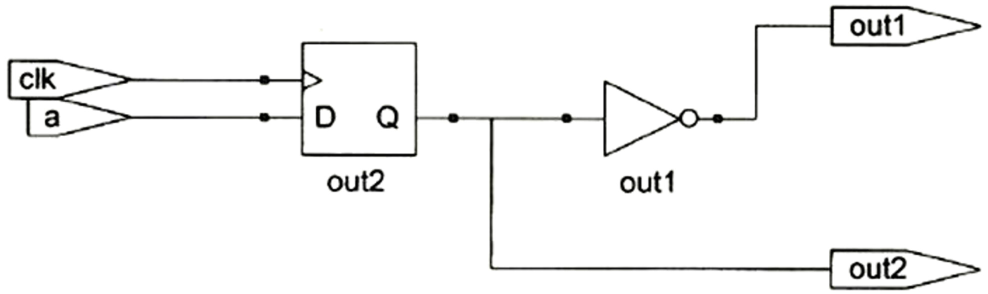


Ex: Ambiguous Output Definition (Correction)

```
reg state, out1, out2;
always @(posedge clk)
  if (state == 0) begin
    if (a) state <= 1;
  end else begin
    if (~a) state <= 0;
  end

always @*
  if (state == 0) begin
    out1 = 1;
    out2 = 0;
  end else begin
    out1 = 0;
    out2 = 1;
  end
endmodule
```

Synthesis Results


clk
a
D
Q
out2
out1
out1
out2