



Phil Moorby and Prabhu Goel – inventors of Verilog Programming Language

Contact Information

Anupam Chattopadhyay

Email: anupam@ntu.edu.sg

Office: N4-02c-105



Plan for the 2nd half of the semester

■ Full-Time Course

Week	Pre-Recorded Lectures	Monday (LT19A) 830-920	Thursday (Zoom) 1630-1720	Tutorial	Lab
7	L13-L14				
Recess Week					
8	L15-L16	L13-L14 Summary	Online Consultation (Zoom)	Tutorial 6	+ quiz 3
9	L17-L18	L15-L16 Summary		Tutorial 7	Experiment 4 + quiz 4
10	L19-L20	L17-L18 Summary		Tutorial 8	
11	L21-L22	L19-L20 Summary (Zoom)		Tutorial 9	Experiment 5 + quiz 5
12	L23	L21-L22 Summary		Tutorial 10	
13		Public holiday			

Plan for the 2nd half of the semester (*contd.*)



■ Part-Time Course

Week	Pre-Recorded Lectures	Tuesday (LT11) 1830-2130		Lab
7	L13-L14			
Recess Week				
8	L15-L16	L13-L14 Summary	Tutorial 6	
9	L17-L18	L15-L16 Summary	Tutorial 7, 8	
10	L19-L20	L17-L19 Summary	Tutorial 9	
11	L21-L22			Experiment 4 + quiz 4
12	L23	L20-L22 Summary	Tutorial 10	
13				Experiment 5 + quiz 5

Plan for the 2nd half of the semester *(contd.)*



- Online Tasks for L13 to L22
 - Will not be graded

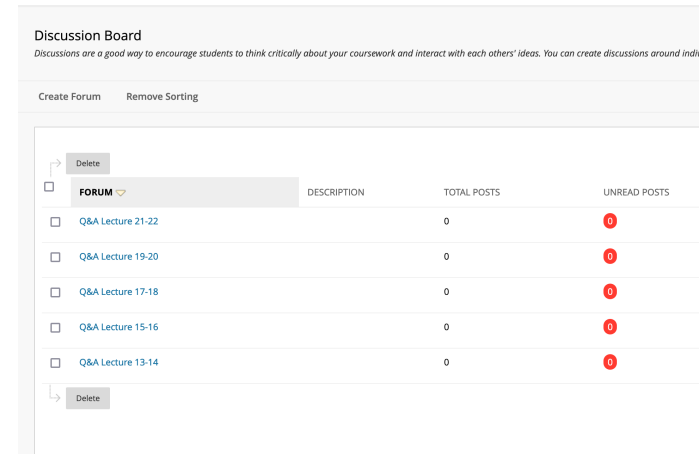
- Discussion lectures (Monday, 8:30-9:20 AM, LT19A)
 - You are required to view the pre-recorded lectures
 - Recap and discussion (slides to be uploaded afterwards)
 - Additional examples and exercises
 - Polls through **Wooclap** (QR code in respective slides)

Plan for the 2nd half of the semester (contd.)



■ Participation in Course

- Use NTULearn Discussion Forum to ask follow-up questions



■ Consultation Slots on (Thursday, 4:30-5:20 PM, Zoom)

- Limit yourself to 3 questions
- Avoid the clarification on tutorial
- <https://ntu-sg.zoom.us/j/81954891824>
Meeting ID: 819 5489 1824
Passcode: 364003



SC1005

Digital Logic

Recap and Discussion

Lecture 15

Introduction to Verilog

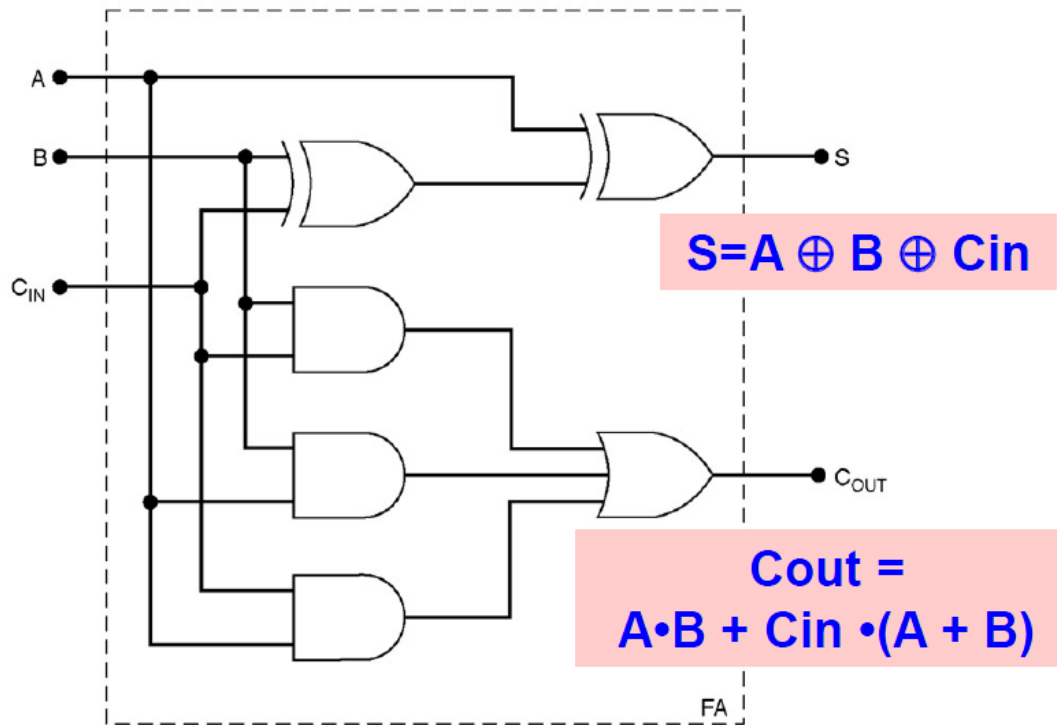
Summary of Lecture 15

- Introduction to Verilog HDL
 - Verilog Assignments
 - Vectors in Verilog
 - Number Literals and Parameters

VERILOG ASSIGNMENTS

Continuous Assignment Example

- Full Adder (using assign statements)



Operator	Name
~	Bitwise NOT
&	Bitwise AND
	Bitwise OR
^	Bitwise XOR
~&	Bitwise NAND
~	Bitwise NOR

Remember that all assign statements happen concurrently (at the same time)

```

module full_adder (input A, B, Cin,
                  output S, Cout);

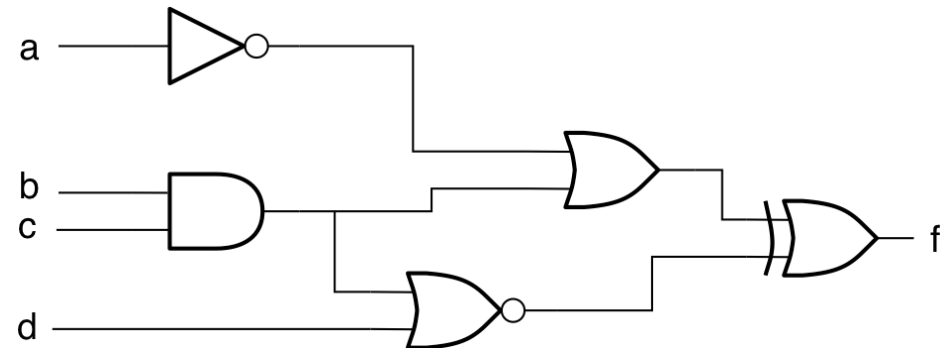
    assign S      = A ^ B ^ Cin;
    assign Cout = (A & B) | (B & Cin) | (A & Cin);

endmodule
    
```

Recap: Verilog Assignments

- Continuous Assignment

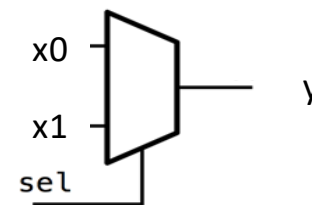
```
assign f = (~a | (b&c)) ^ ((b&c) ~| d);
```



- Conditional Assignment

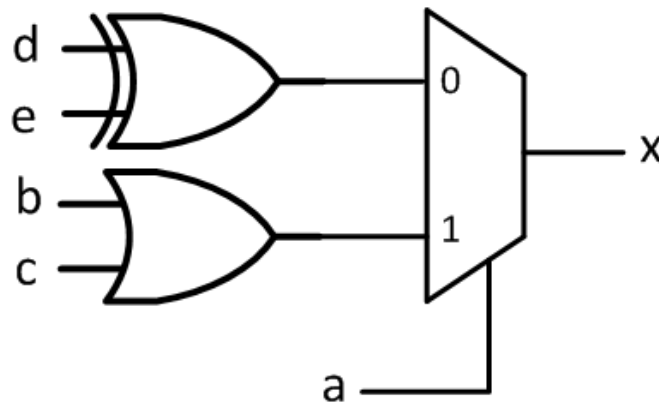
```
assign y = sel ? x1 : x0; // a multiplexer
```

1-bit 2x1 mux



Exercise 1

- Which of the following is the correct continuous assignment for the circuit below?



1


Go to wooclap.com

2

Enter the event code
in the top banner

Event code

DMYHZJ

- A. `assign x = a ? (b^c) : (d|e);`
- B. `assign x = a ? (d^e) : (b|c);`
-  C. `assign x = a ? (b|c) : (d^e);`
- D. `assign x = a ? (d|e) : (b^c);`

VECTORS IN VERILOG

Vectors

- Verilog has a special construct for handling multi-bit signals (wires). **Formed by specifying a range:**
- By convention, we label the *most significant bit* (MSB) using the higher number, and the *least significant bit* (LSB) using zero

```
wire [31:0] databus;
```

- Also for specifying multi-bit module ports

```
module add16 (input [15:0] a, b,
              output [15:0] sum,
              output cout);

    // add module internals here

endmodule
```

~~32~~ databus

is short
for
↓

— databus[31]

— databus[30]

—

— databus[0]

Recap: Vectors and Arithmetic Operations

- Arithmetic: +, -, *
- Comparisons: <, <=, >, >=, ==, !=

```
module adder (  
    input Cin,  
    input [31:0] A, B,  
    output Cout,  
    output [31:0] Sum);  
  
    wire [32:0] w1;  
  
    assign w1    = A + B + Cin;  
    assign Sum   = w1[31:0];  
    assign Cout  = w1[32];  
  
endmodule
```

- We must declare multi-bit signals

Recap: Vectors in Verilog

- We can select *individual bits* of the vector:

```
assign y = some[3]; //assign 4th bit of  
                  //some to y
```

- We can select a *range* of the vector:

```
assign z = some[4:3]; //assign 5th/4th bit of some  
                  //to two-bit signal z
```

- We can *assign* to individual bits or a range:

```
assign x[0] = y[1];  
assign x[2:1] = y[4:3];
```

- Check the Synthesis warnings for width mismatch
 - **WARNING:**HDLCompiler:189 "C:\path\myDesign.v" Line 25: Size mismatch in connection of port <sum>. Formal size is 16-bit, actual signal size is 1-bit.

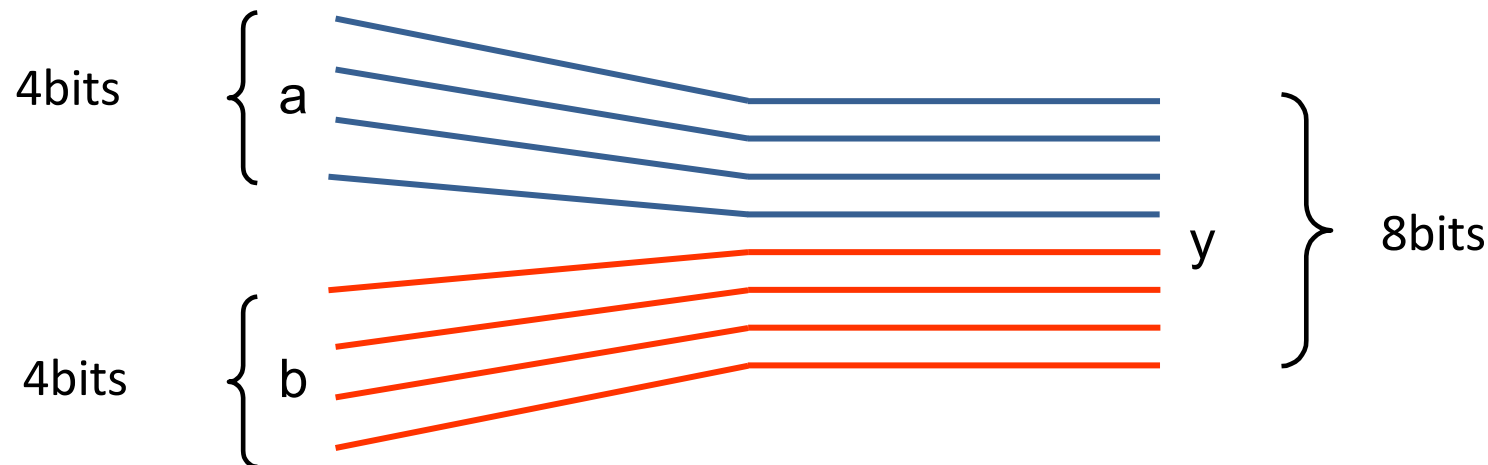
NUMBER LITERALS AND PARAMETERS

Number Literals

- Verilog allows us to use number *literals*:
 - `<size>'<radix><value>`
 - `<size>` is the width **in bits**
 - `<radix>`: b for binary, o for octal, h for hex, d for decimal
 - `<value>`: the number you want, with as many optional underscores as needed (for readability)
- Examples:
 - `4'b0000` (4 binary bits “0 0 0 0”)
 - `8'h4F` (= `8'b01001111`)
 - `8'b0100_1111` (Same as above. Note the use of the underscore)
 - `1'b1` (a single “1”)

Concatenation

- It is sometime very useful to be able to concatenate a number of signals into a single signal.
 - Concatenation is signified by curly brackets enclosing a list



```
wire [3:0] a, b;  
wire [7:0] y;  
...  
assign y = {a, b};
```

Concatenation: Example

```
// b = 2'b11, c = 2'b00, d = 3'b110
```

```
wire [3:0] x;
```

```
wire [7:0] y;
```

```
assign x = {b, c}; // x = 4'b1100
```

```
assign y = {b, d, 3'b011}; // y = 8'b11110011
```

Exercise 2

- What is the resulting signal value of z after the following assign statement?

```
// a = 5'b01011, b = 2'b10, c = 2'b00, d = 3'b100
```

```
wire [7:0] z;
```

```
assign z = {a[3:0], b[1], c, d[2]};
```

- A. z = 8'b0101_0000
- ✓ B. z = 8'b1011_1001
- C. z = 8'b0101_1001
- D. z = 8'b1011_0000



1 Go to wooclap.com

2 Enter the event code
in the top banner

Event code
DMYHZJ

Replication: Example

- Replication uses braces with a preceding integer or variable representing an integer.

```
// a = 1'b1, c = 4'b1010
```

```
wire [3:0] x;
```

```
wire [7:0] z;
```

```
assign x = { 4{a} }; // x = 4'b1111
```

```
assign z = {{4{c[3]}}}, c[3:0]};
```

```
// z = 8'b1111_1010
```


Exercise 3

- What is the resulting signal value of y after the following assign statement?

```
// a = 1'b1, b = 2'b10  
wire [7:0] y;  
  
assign y = { 4{a}, 2{b} };
```

- A. y = 8'b1111_0000
- B. y = 8'b1111_0010
- C. y = 8'b1111_1000
- ✓ D. y = 8'b1111_1010





- 1 Go to wooclap.com
- 2 Enter the event code in the top banner

Event code
DMYHZJ

Recap: Adder Example

- **Parameter** is a constant that is local to a module
- Use **arithmetic addition operator (+)** to generate the
- sum
- **Concatenation** operator handles the 33-bit result produced by $A+B+C_{in}$

```
module adder #(parameter SIZE=32)(  
    input Cin,  
    input [SIZE-1:0] A, B,  
    output Cout,  
    output [SIZE-1:0] Sum);  
  
    assign {Cout, Sum} = A+B+Cin;  
  
endmodule
```

Recap: Parameters

- It is also possible to redefine a **parameter**

```
module submod #(parameter SIZE=8) (  
    input    [SIZE-1:0] X, Y, output [SIZE-1:0] Z);  
    // some statements in here  
endmodule
```

```
module top_mod #(parameter Const=16) (  
    input    [Const-1:0] a, b, c, output [Const-1:0] D, E);  
  
    submod #(.SIZE(Const)) U1 (.X(a), .Y(b), .Z(D));  
    submod #(.SIZE(Const)) U2 (.X(c), .Y(b), .Z(E));  
  
endmodule
```


SC1005

Digital Logic

Recap and Discussion

Lecture 16

Verilog for Combinational Circuits

Summary of Lecture 16

- Verilog for Combinational Circuits
 - Behavioral Modelling
 - If and Case Statements
 - Important Considerations

So, Where are We?

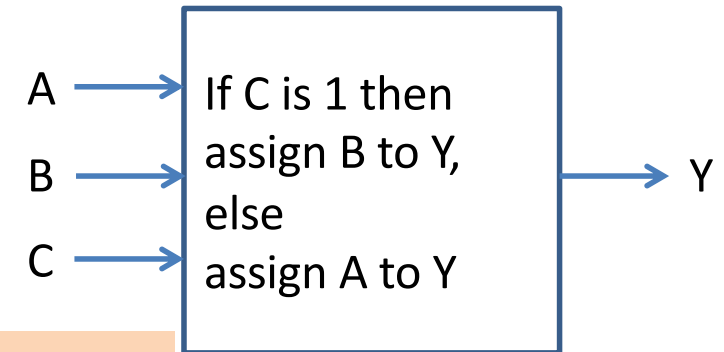
- We have covered: **modules, ports, wires, module instantiation, gates** and **assign statements**

```
module myModule (input a, b, data_i,  
                 output flag, data_o, addr);  
  
    wire or1out;  
  
    get_Data IC1 (.din(data_i), .add(addr), .do(data_o));  
    or g1 (or1out, a, b);  
    assign flag = or1out & data_i;  
  
endmodule
```

- So far we have only examined **Structural Verilog**.

Recap: Behavioral Modeling

- A more powerful level: we describe **how** the circuit **behaves**, not how it is *constructed*
- We use a **combinational always block** to describe the behavior of the desired hardware



```

always @ (a, b)
begin
    x = a & b;
    y = a | b;
    z = a & c;
end
  
```

This will not generate the expected output as it will not be triggered when input c changes

```

always @ *
begin
    x = a & b;
    y = a | b;
    z = a & c;
end
  
```

This is OK. Whenever any signal changes (* means all input signals) a change in the output is forced.

The “*” means all input signals inside the block.

Recap: Behavioral Verilog, Combinational Circuits



- An **always** block contains procedural statements that describe the behavior of the desired hardware
- The *sensitivity list* **must** contain the names of any signals that affect the output of the block.
 - Easier to use **always @***
- If you assign to a signal from inside an **always** block, you must **never** assign to it from **anywhere else**
- Order of statements **matter** within a combinational always block
 - Order of multiple **always** blocks in a module doesn't matter

```
always @ (a, b)
begin
    x = a & b;
    y = a | b;
end
```

Sensitivity List

Procedural statements

Recap: Reg

- **wire** is simply a connection, while **reg** can hold a state
- **wire** is used as the left-hand-value for **assign** statement
- **reg** is used as the left-hand-value for **=** and **<=**
- Signals in left-hand-value within an **always** block
 - Must be declared as being of type: **reg**

```
module temp (input a, b,  
             output out);  
  
    wire w1;  
  
    assign w1 = a & b;  
    assign out = w1;  
  
endmodule
```

```
module temp (input a, b,  
             output out);  
  
    reg w1;  
  
    always @ *  
        w1 = a & b;  
  
    assign out = w1;  
endmodule
```


Exercise 4:

State **all** signals in the following Verilog code segment that must be declared as **reg**.

```
module temp (...);  
    ...  
    assign w1 = a ^ b;  
    always @ *  
    begin  
        r1 = a & b;  
        out_reg = a | w1;  
        out_bus = 5'd25;  
    end  
    assign out = r1;  
endmodule
```

- A. r1, out_reg, out_bus, a, b, w1
- B. r1, out_reg, out_bus, w1
- C. out_reg, out_bus, out
- D. r1, out_reg, out_bus, w1, out
- ✓ E. None of the above





- 1 Go to wooclap.com
- 2 Enter the event code in the top banner

Event code
DMYHZJ

Recap: If Statement

- You can have more than one statement in each branch. If so, you use **begin** and **end**

```
always @ *  
begin  
    if (x < 6)  
        alarm = 1'b0;  
    else  
        alarm = 1'b1;  
end
```

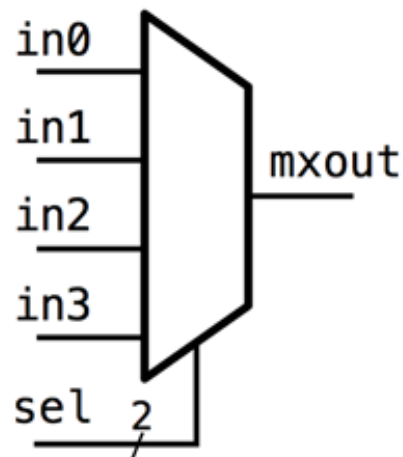
```
always @ *  
begin  
    if (alarm == 1'b1)  
        begin  
            if (after_hours)  
                siren = 1'b0;  
            else  
                siren = 1'b1;  
                light = 1'b1;  
            end  
        else  
            begin  
                siren = 1'b0;  
                light = 1'b0;  
            end  
        end  
end
```


Recap: Case Statement

- We can use **case** statements:

```
always @ *
case (sel)
    2'b00    : y = a;
    2'b01    : y = b;
    2'b10    : y = c;
    default  : y = 4'b1010;
endcase
```

1-bit 4x1 mux

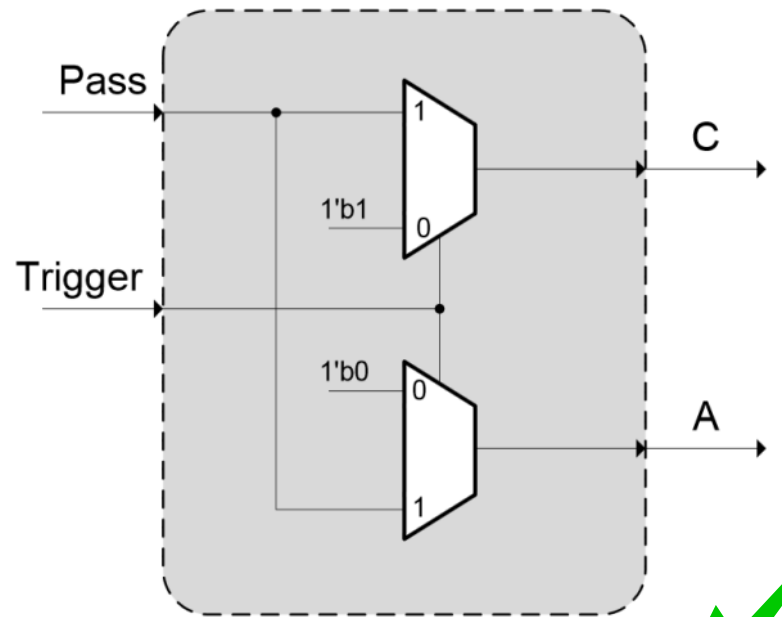


```
module mux4 (output reg q,
             input [3:0] d,
             input [1:0] sel);

    always @ * begin
        case (sel)
            2'b00 : q = d[0];
            2'b01 : q = d[1];
            2'b10 : q = d[2];
            2'b11 : q = d[3];
        endcase
    end
endmodule
```

Exercise 5

- Which of the following Verilog code will generate the circuit shown?



Circuit X

```

wire Trigger, Pass;
reg A, C;

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

```

Circuit Y

```

wire Trigger, Pass;
reg A, C;

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

```

Circuit Z 

```

wire Trigger, Pass;
reg A, C;

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

```



1

Go to wooclap.com

2

Enter the event code
in the top banner


Event code
DMYHZJ


IMPORTANT CONSIDERATIONS IN BEHAVIORAL VERILOG

Recap: Avoiding Latches

- What happens to *y* in the **else** branch?

```
always @*  
begin  
    if(valid) begin  
        x = a | b;  
        y = c;  
    end  
    else  
        x = a;           //y?  
end
```



```
always @*  
begin  
    y = x;   
    if(valid) begin  
        c = a | b;  
        y = z;  
    end  
    else  
        c = a;  
end
```

- One way to fix this is to use a default assignment at the top of the **always** block

Exercise 6

- Which of these Verilog description will result in latches?

X


```
wire A, B;  
reg Y;  
  
always @(A or B)  
begin  
    Y = A|B;  
end
```

Y

```
wire[1:0] x;  
reg [1:0] q;  
  
always @(x)  
begin  
    case (x)  
        2'b00: q = 2'b01;  
        2'b10: q = 2'b10;  
    endcase  
end
```

Z

```
Wire s;  
reg q,r;  
  
always @(s)  
begin  
    case (s)  
        1'b0: q = 1'b1;  
        1'b1: r = 1'b1;  
    endcase  
end
```

- A. X
- B. Y
- C. Z
- D. X and Y
-  E. Y and Z



- 1 Go to wooclap.com
- 2 Enter the event code in the top banner

Event code
DMYHZJ

END OF L15,L16 SUMMARY