# Contact Information

Anupam Chattopadhyay

Email: anupam@ntu.edu.sg

Office: N4-02c-105

# Plan for the 2nd half of the semester

NANYANG TECHNOLOGICAL UNIVERSITY

- 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.)*

NANYANG TECHNOLOGICAL UNIVERSITY

- 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.)*

NANYANG TECHNOLOGICAL UNIVERSITY

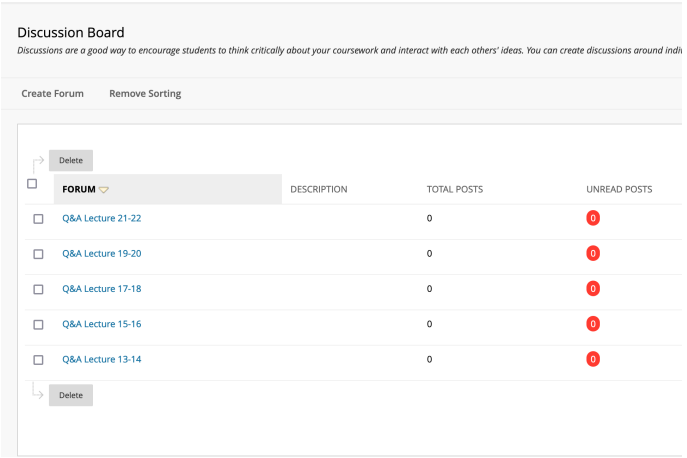
- Online Tasks for L13 to L22
  - Will not be graded

- Discussion lectures (Monday, 8:30-9:30 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.)*

NANYANG TECHNOLOGICAL UNIVERSITY

- 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

NANYANG TECHNOLOGICAL UNIVERSITY

# SC1005
# Digital Logic

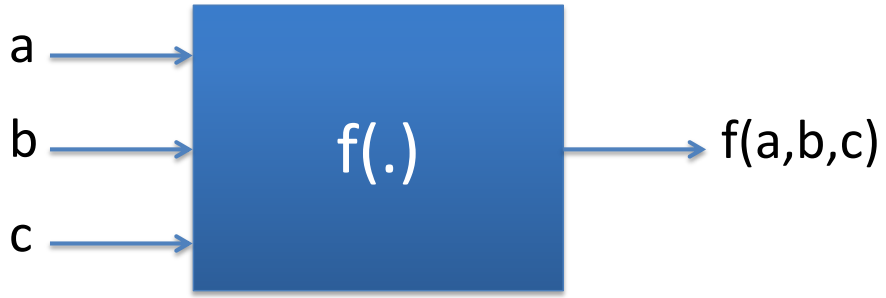
## Recap and Discussion

Lecture 13

Combinational Circuits

# Summary of Lecture 13

- Combinational circuits are functions:
  - *Outputs depend solely on the **present combination** of input values*

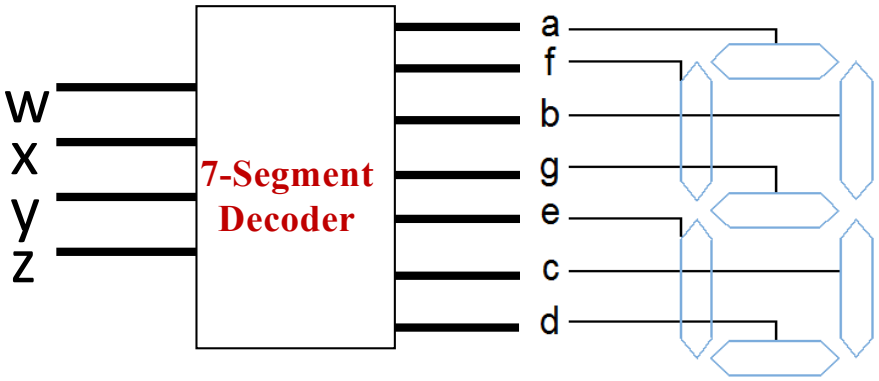


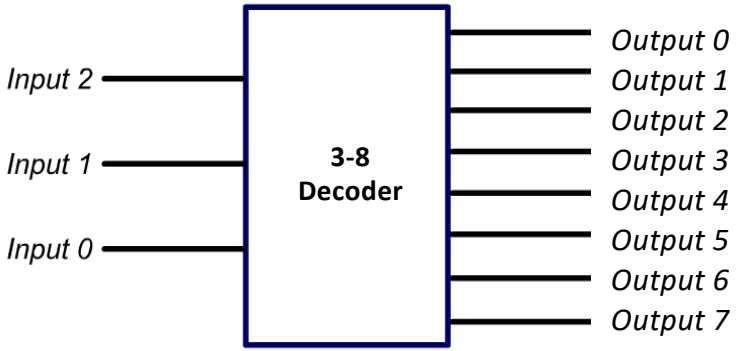

- Combinational Circuits
  - Seven-Segment Decoder
  - Decoder (One-Hot)
  - Multiplexers

- Timing Diagrams

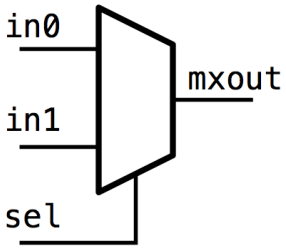
# Combinational Circuits

- Examples:
  - Seven-Segment Decoder
  - Decoder (One-hot)
  - Multiplexer

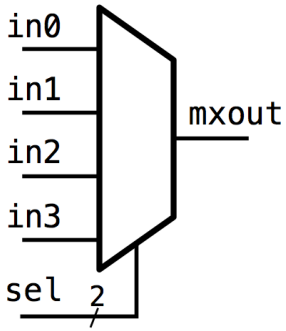


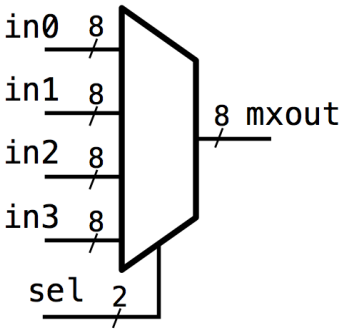

1-bit 2x1 mux




1-bit 4x1 mux




8-bit 4x1 mux



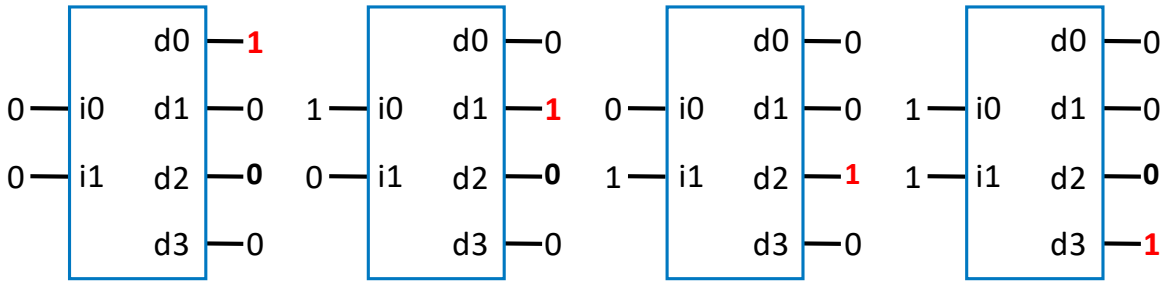
# DECODER

# Decoder

- Decoders are an important basic circuit, similar to the 7-Segment Decoder
- Take a binary input number, and output a corresponding one-hot output
  - Only one bit of the output is high, its position corresponds to the input value
  - Output width is always 2 to the power of input width
    - $N$-input Decoder: $2^N$ outputs

| Input | | Output |
|---|---|---|
| 00 | → | 0001 |
| 01 | → | 0010 |
| 10 | → | 0100 |
| 11 | → | 1000 |

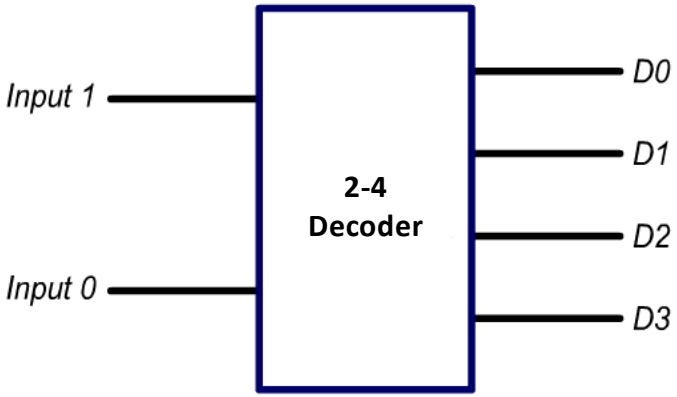
*Source: Frank Vahid, "Digital Design", Wiley, 2nd Edition, pg. 84*

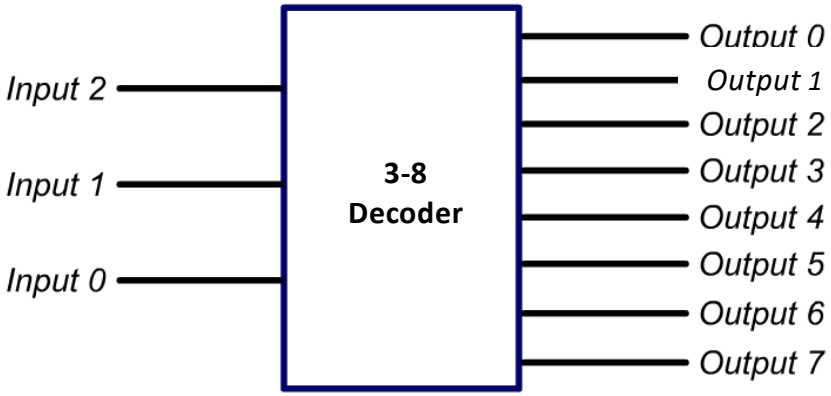



Example: A two-input decoder will have four outputs (2-4 Decoder)

# Decoder

- In general, decoders can be referred to n-m decoders




| Input | | Output | | | |
|---|---|---|---|---|---|
| 1 | 0 | D0 | D1 | D2 | D3 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |

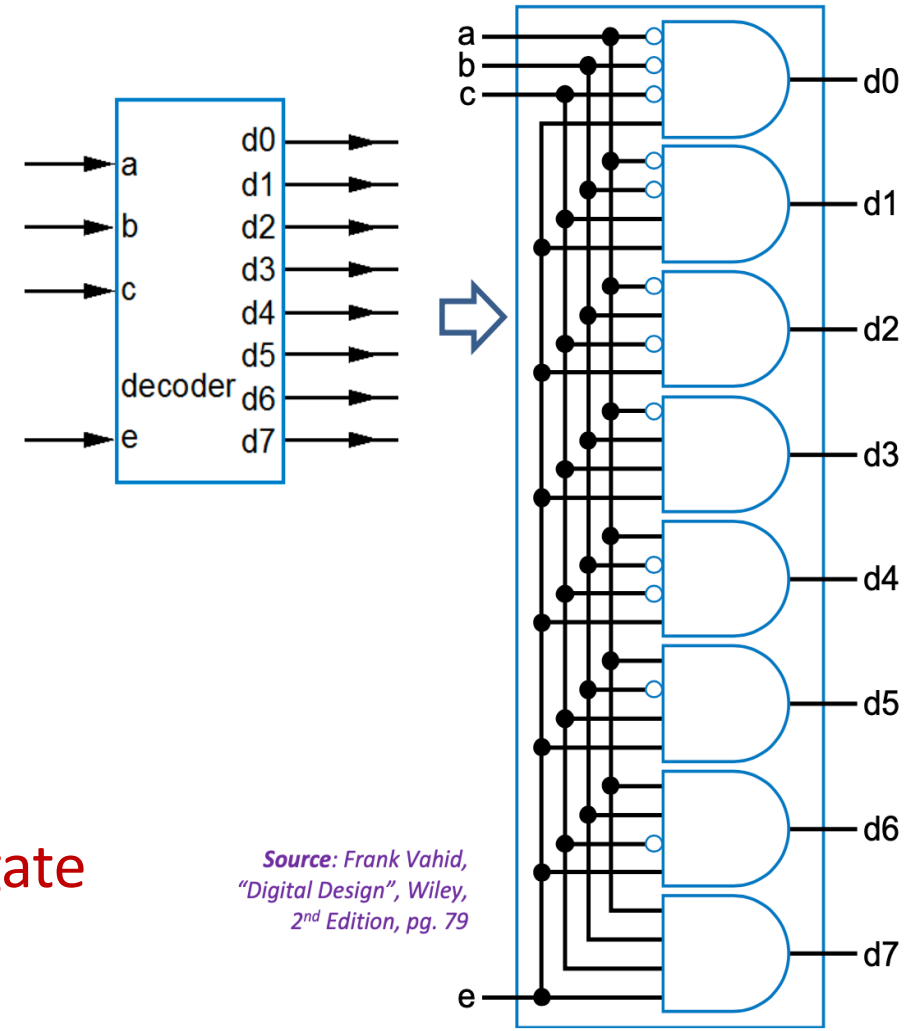
| Input | | | Output | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 1 | 0 | D0 | D1 | D2 | D3 | D4 | D5 | D6 | D7 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

***Source**: https://filebox.ece.vt.edu/~jgtront/introcomp/decoder.swf*

# Decoder: Example

- Step 1: Capture function
  - d0 = a'b'c'e = 0001
  - d1 = a'b'ce = 0011
  - d2 = a'bc'e = 0101
  - d3 = a'bce = 0111
  - d4 = ab'c'e = 1001
  - d5 = ab'ce = 1011
  - d6 = abc'e = 1101
  - d7 = abce = 1111
- Step 2: Implement Circuit
  - Each term is a 4-input AND gate

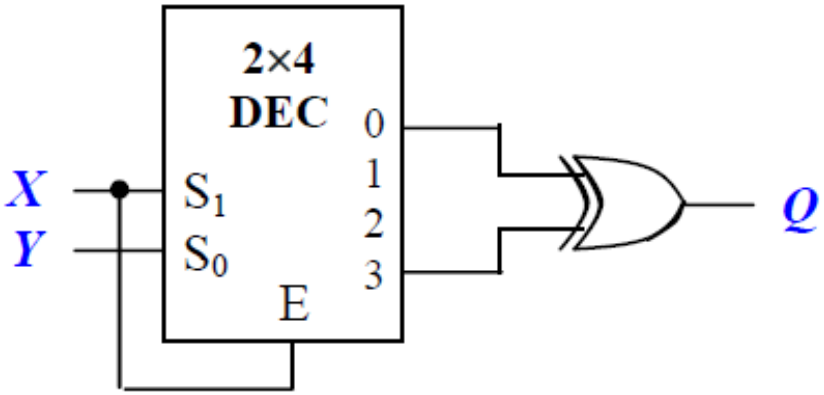
**This is a 3-8 Decoder.**




*Source: Frank Vahid, "Digital Design", Wiley, 2nd Edition, pg. 79*

# Exercise 1

- What is the Boolean expression of Q?




A. 1
B. X
C. X.Y
D. X' + Y'
E. X'Y + X.Y'

$$
\begin{aligned}
Q &= X'Y'E \oplus XYE \\
&= X'Y'X \oplus XYX \quad \text{(since } X = E\text{)} \\
&= XY
\end{aligned}
$$

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

Event code
ZLXOFA

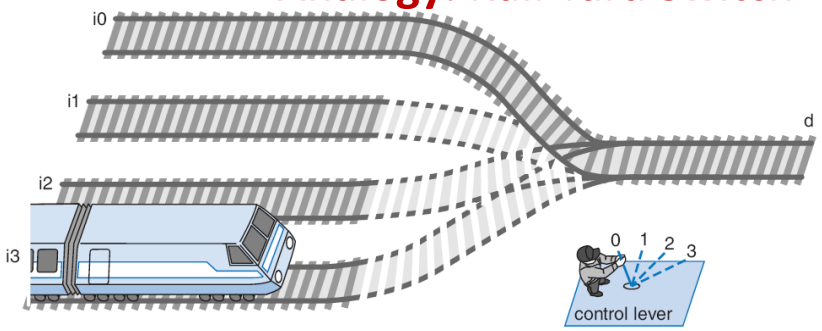
NANYANG TECHNOLOGICAL UNIVERSITY

# MULTIPLEXER

# Multiplexer (Mux)

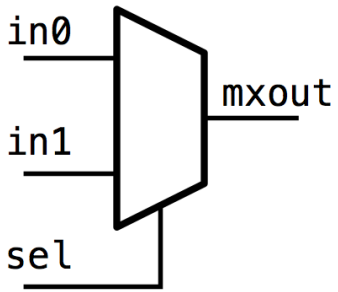
- Consists of multiple inputs, and a single output
- A select input determines which input should be connected to the output
- 2-input MUX needs
  - 1-bit select input
- $N$-input MUX needs
  - $log_2(N)$ bit select inputs

**Analogy: Rail Yard Switch**

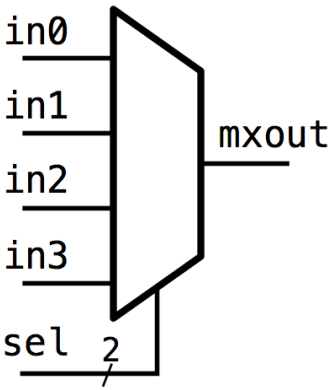



***Source**: Frank Vahid, "Digital Design", Wiley, 2nd Edition, pg. 87*

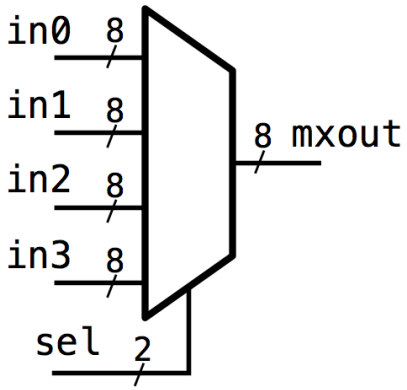
1-bit 2x1 mux




1-bit 4x1 mux

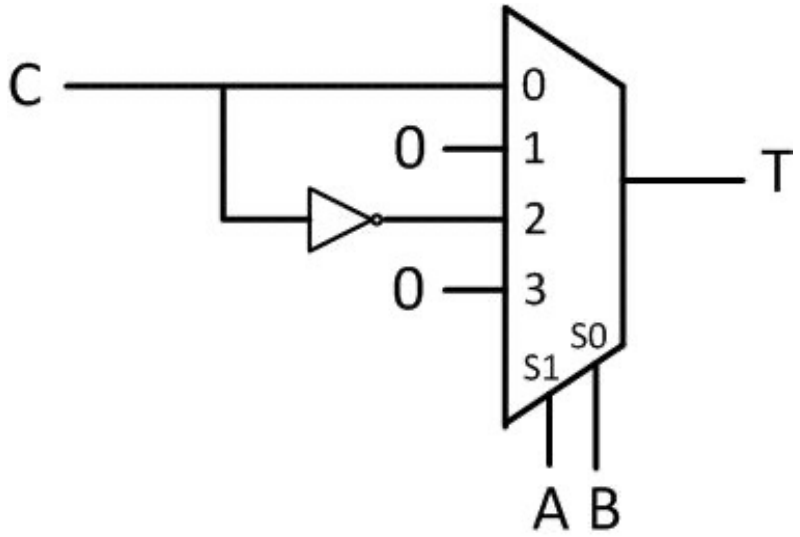



8-bit 4x1 mux



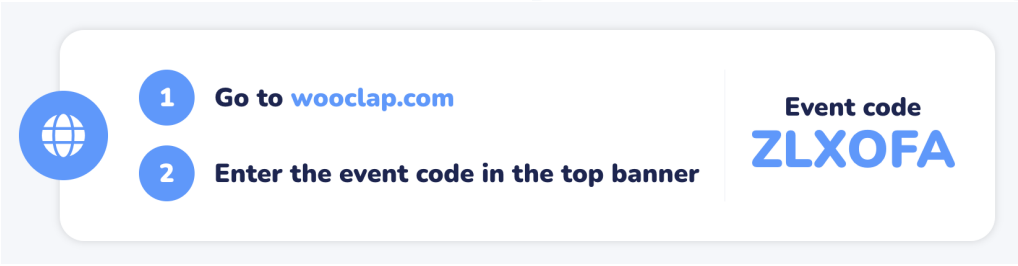
# Exercise 2

- Find the SOP expression for function T(A,B,C)




A, B are the select signals of the 4x1 multiplexer.
T = C when AB = 00 → A'B'C
T = C' when AB = 10 → AB'C'

A. A'.B'.C + A.B'.C'
B. A'.B'.C' + A.B'.C'
C. A'.B'.C + A.B'.C
D. A'.B'.C' + A.B'.C
E. None of the above



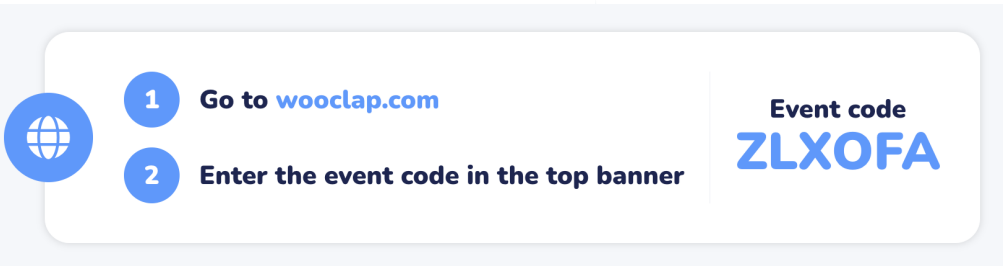
# Exercise 3

- An $Nx1$ multiplexer has $N$ inputs

TRUE

✓ FALSE

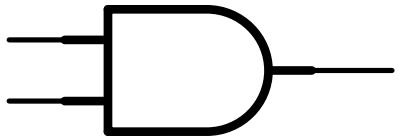
$N$-input MUX needs $log_2(N)$ bit select inputs
Hence, total [N + $log_2(N)$] inputs



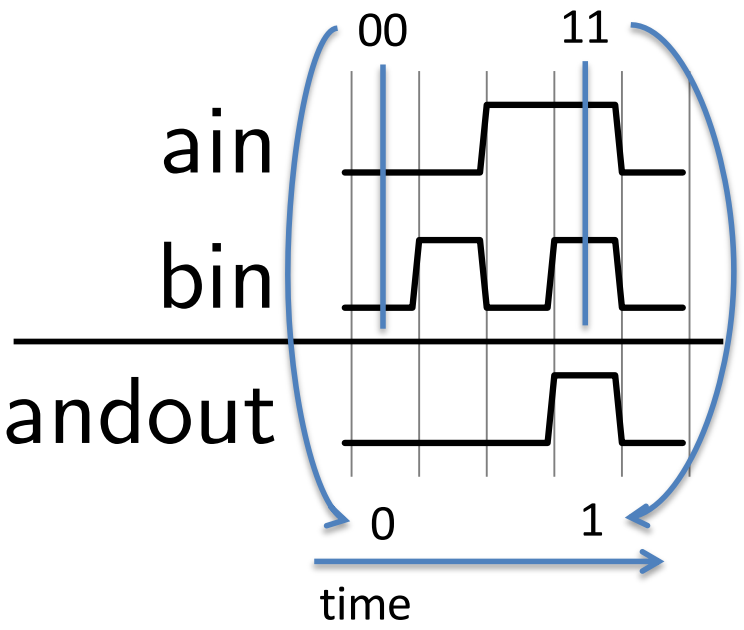
# TIMING DIAGRAMS

# Timing Diagrams

- Timing diagrams show the behavior of a circuit with progression of time
- Input values are changed and the resultant outputs shown
- Consider an **AND** gate:

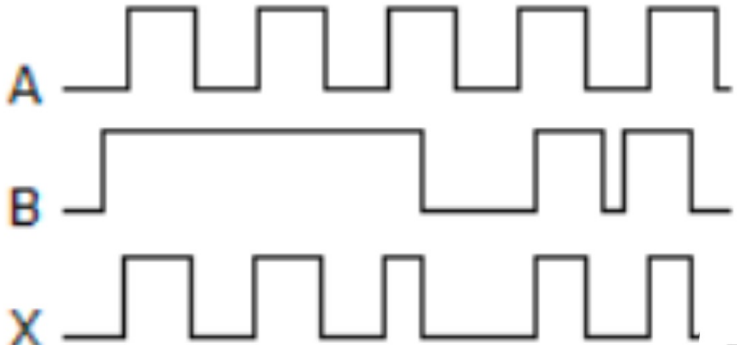


- A timing diagram can show any combination or order of input values
- The output at any point is calculated by looking at the input values at that instance

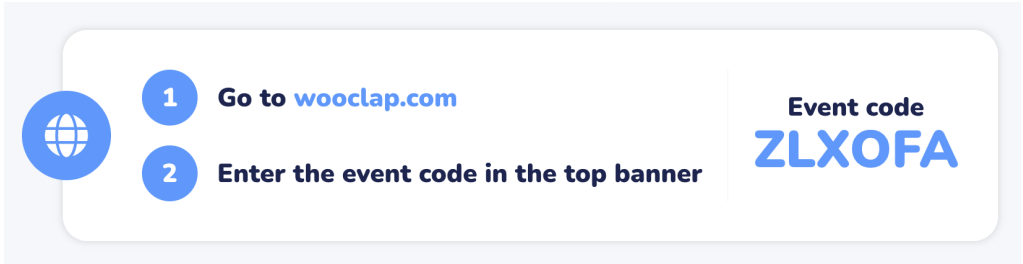


# Exercise 4

- The following waveform for inputs A, B and output X depicts




A. 2-input AND gate
B. 2-input OR gate
C. 2-input XOR gate
D. None of the above



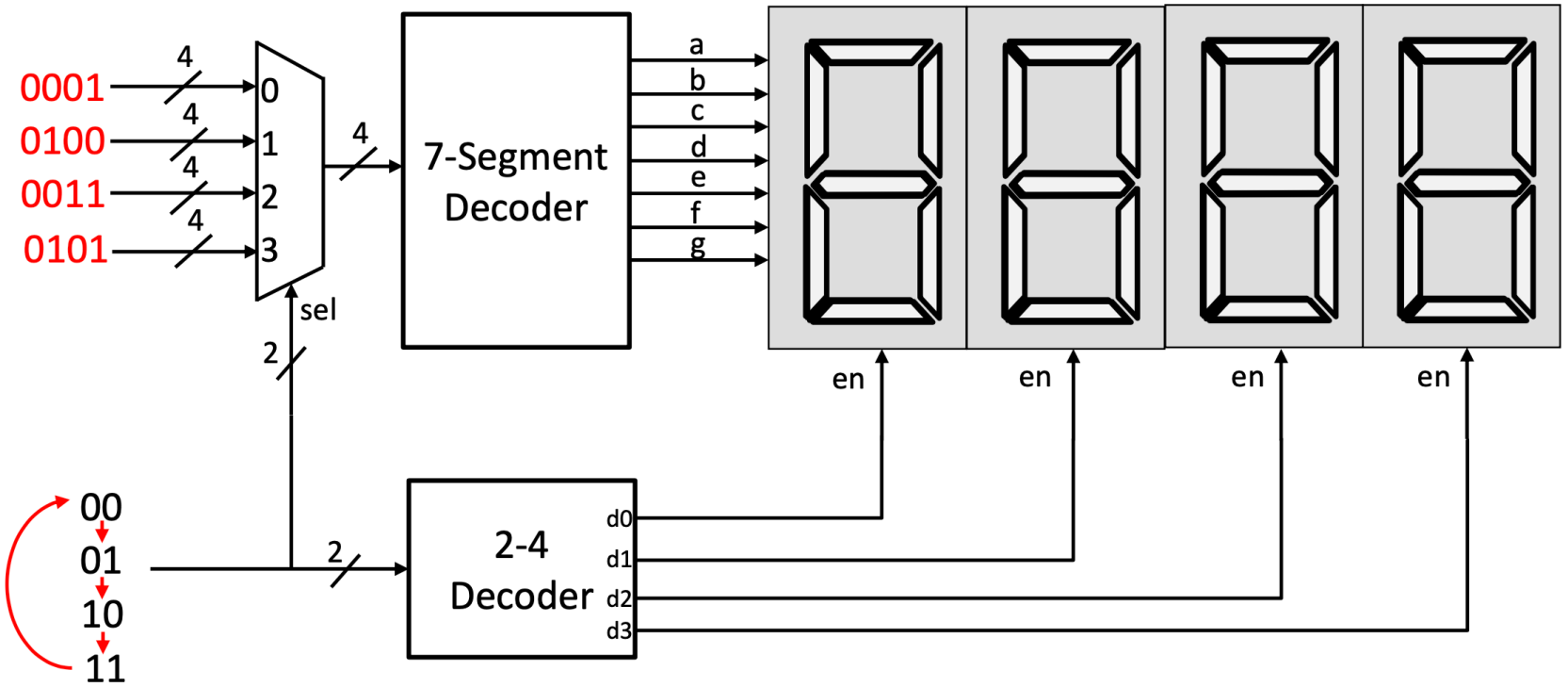
NANYANG TECHNOLOGICAL UNIVERSITY


Combinational Circuits

# DESIGNING A DIGITAL CLOCK DISPLAY

# Digital Clock Display

- Four seven segment displays
- One seven segment decoder
- One 4-bit 4x1 Multiplexer
- One 2-4 Decoder



NANYANG TECHNOLOGICAL UNIVERSITY

# SC1005
# Digital Logic

## Recap and Discussion

Lecture 14

Introduction to Verilog

# Summary of Lecture 14

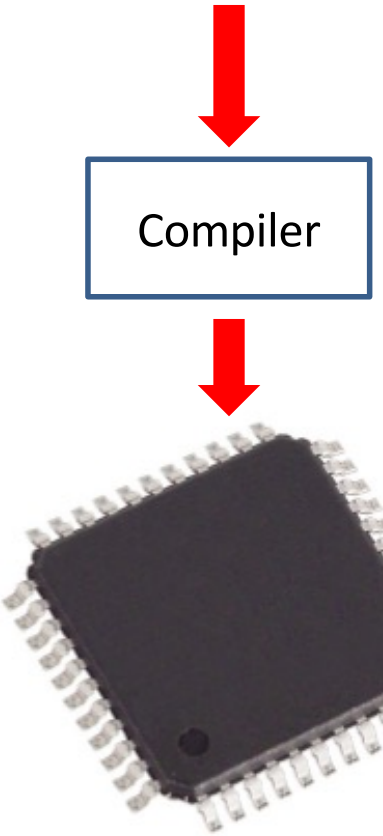
NANYANG TECHNOLOGICAL UNIVERSITY

- Introduction to Verilog HDL
  - Module Declaration
  - Instantiating Gate-Level Primitives
  - Module Instantiation

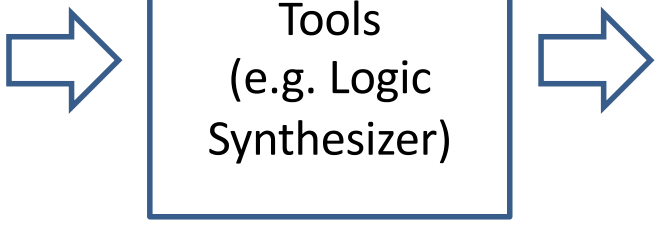
# Hardware Description Languages (HDLs)

- HDLs are **programming-like** languages that are used to describe hardware
- HDLs are synthesized (and optimized) to hardware primitives
- Sophisticated tools can then ensure the hardware generated from the description is efficient

```
void main()
{
   int i, nc;
   nc = 0;
   i = getchar();
   while (i != EOF) {
        nc = nc + 1;
        i = getchar();
   }
   printf("Chars = %d\n", nc);
}
```



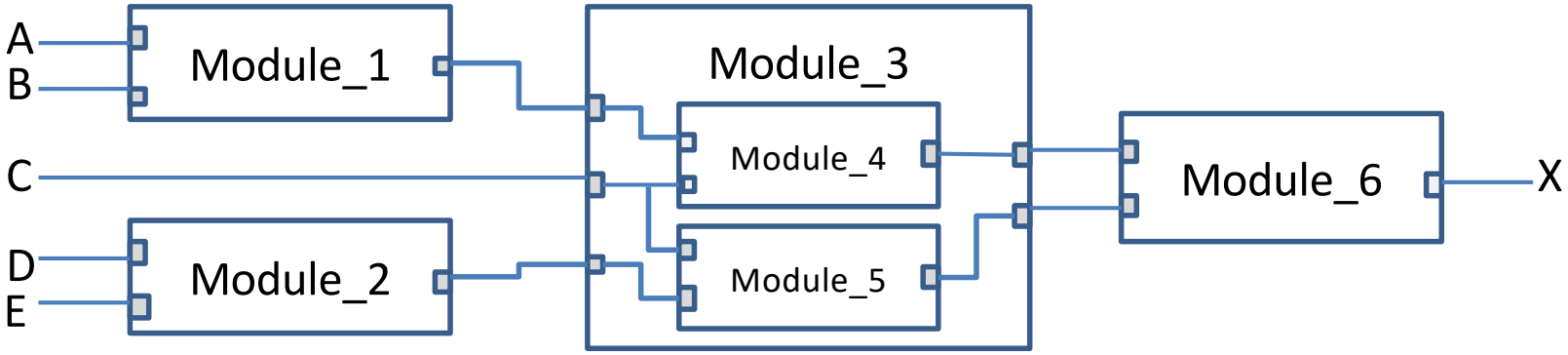

```
module counter (clk,
reset, enable, count);
  input clk, reset,
enable;
  output [3:0] count;
  reg [3:0] count;

  always @ (posedge clk)
  if (reset == 1'b1)
     count <= 0;
else if ( enable == 1'b1)
     count <= count + 1;

endmodule
```

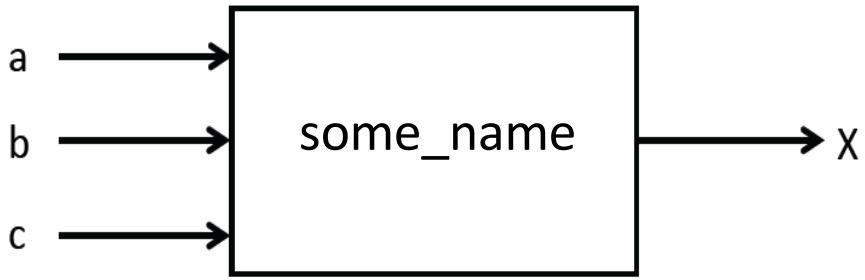


# MODULES

# Module

- In Verilog, designs are broken down into ***modules***
- At each level in the hierarchy, a module instance is treated as a "black-box" – the internals are unknown



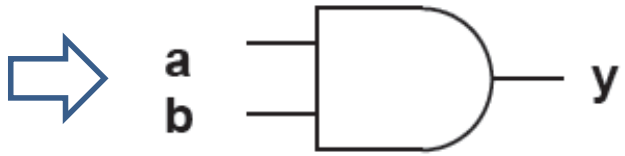

```
module some_name (
      input a, b, c,
      output X);

  // Describe your circuit here

endmodule
```

# INSTANTIATING GATE-LEVEL PRIMITIVES

# Gate-Level Primitives

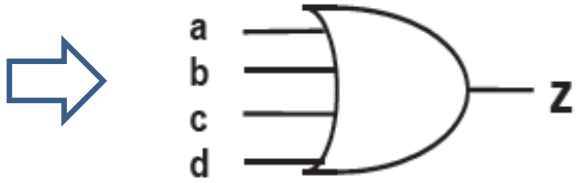
- Verilog provides us with basic ***primitives*** to model Boolean gates: and, nand, or, nor, not, xor, xnor

```
and (y, a, b);
```




- Represents an ***and*** gate with inputs connected to wires a and b, and output connected to wire y
- Gate primitives allow more than two inputs:

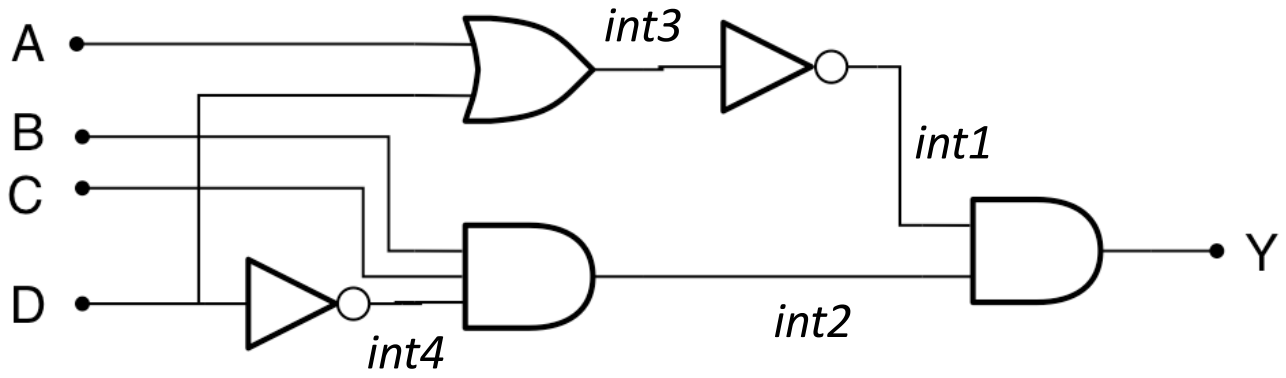
```
or (z, a, b, c, d);
```




- This represents an ***or*** gate with inputs a, b, and c, d and output z

# Exercise

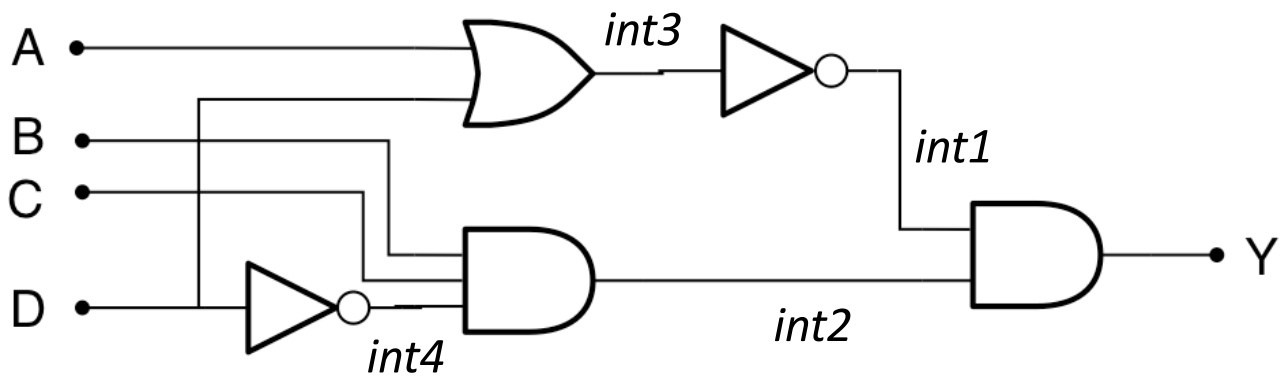
- Implement a Verilog module using structural gate-level primitives for the following circuit:




```
module simple_circ (input A, B, C, D, output Y);
   wire int1, int2, int3, int4;

   and (Y, int1, int2);
   not (int1, int3);
   or  (int3, A, D);
   and (int2, B, C, int4);
   not (int4, D);
endmodule
```

# Exercise

- Implement a Verilog module using structural gate-level primitives for the following circuit:




```
module simple_circ (input A, B, C, D,
                    output Y);

   wire int1, int2, int3, int4;

   and (Y, int1, int2);
   not (int1, int3);
   or  (int3, A, D);
   and (int2, B, C, int4);
   not (int4, D);
endmodule
```

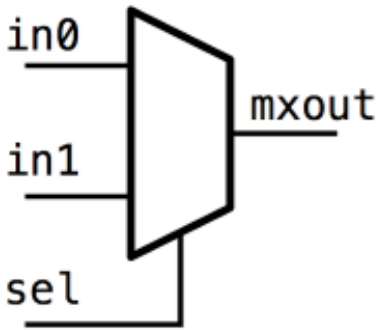
```
module simple_circ (input A, B, C, D,
                    output Y);

   wire int1, int2, int3, int4;

   not (int1, int3);
   not (int4, D);
   and (int2, B, C, int4);
   and (Y, int1, int2);
   or  (int3, A, D);
endmodule
```

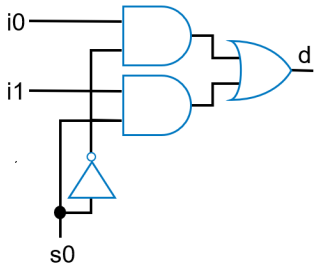
These two implementations will produce the same circuit

# Exercise 5

- Which of the following is correct gate-level instantiation of a 2x1 multiplexer in Verilog:

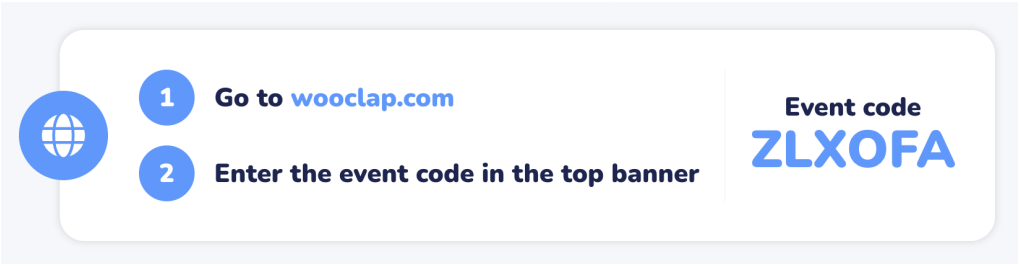
1-bit 2x1 mux




```
module mux (input in0, in1, sel,
            output mxout);

   wire w1, w2, w3;

   or  (muxout, w1, w2);
   and (w1, in0, w3);
   not (w3, sel);
   and (w2, in1, sel);
endmodule
```

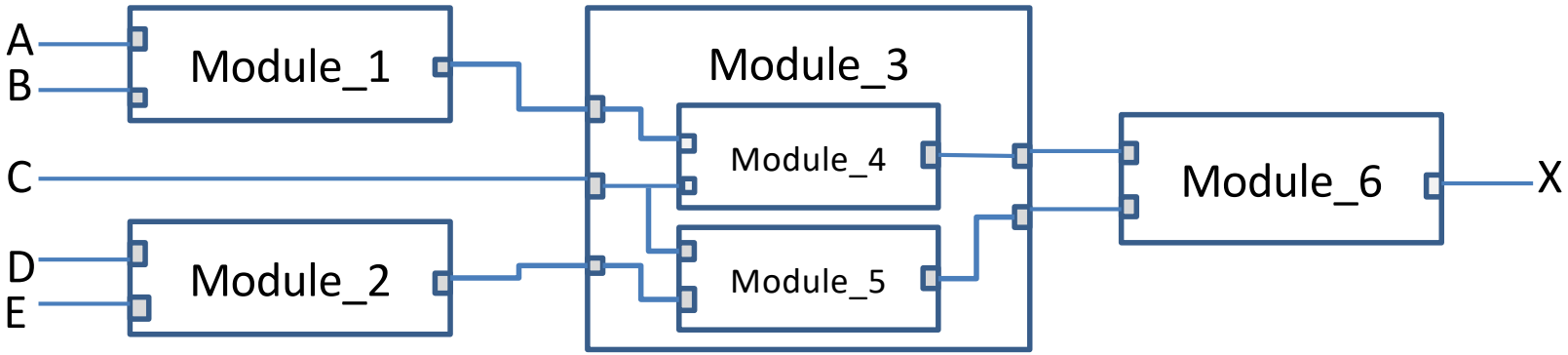
A. mux (mxout, in0, in1, sel);
B. mux (in0, in1, sel, mxout);
C. mux (mxout, sel, in0, in1);
D. None of the above



# MODULE INSTANTIATION

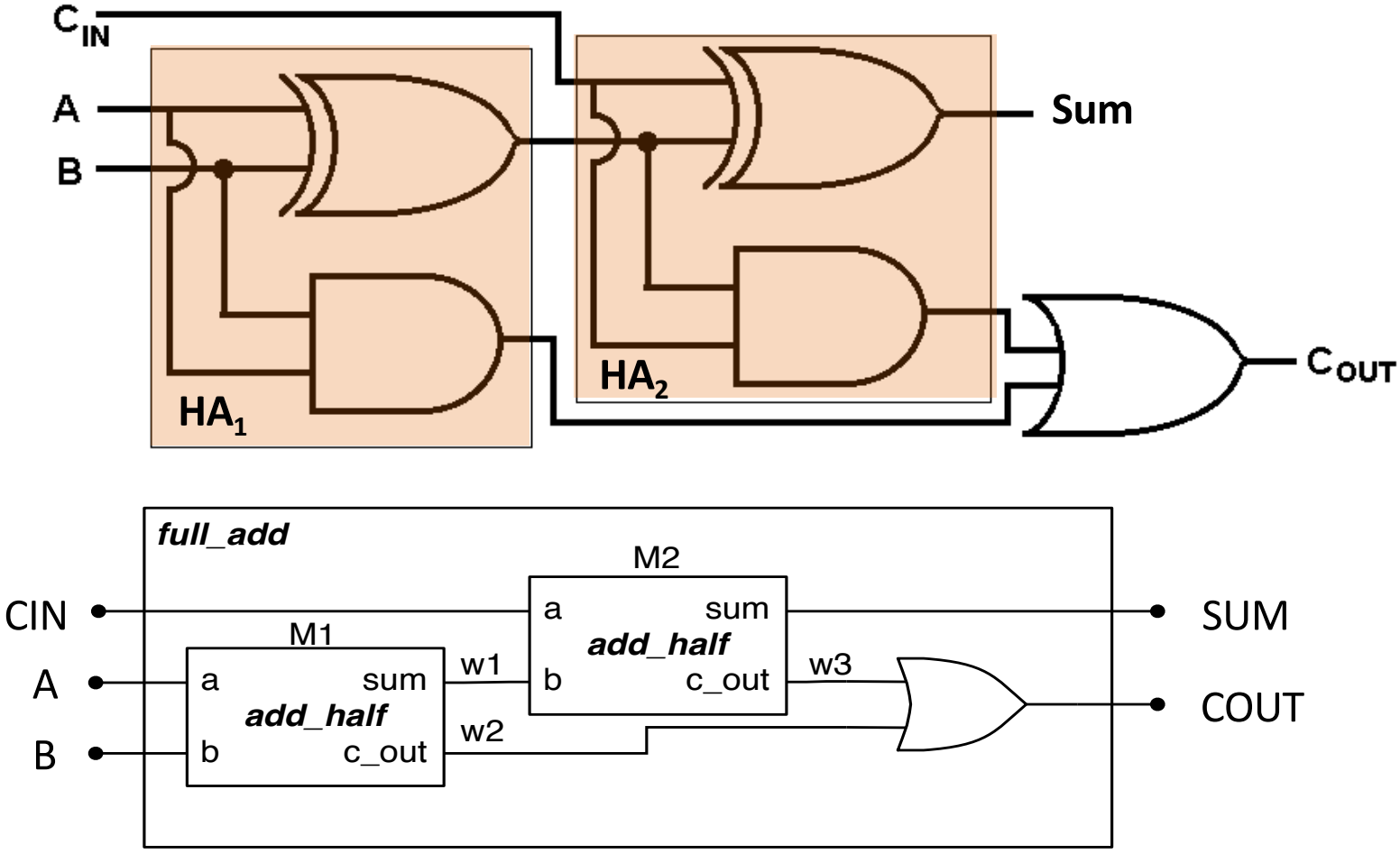
# Module

- In Verilog, designs are broken down into ***modules***
- Good designs consist of sufficient (but not excessive) ***levels of hierarchy***, with modules containing instances of other modules
- At each level in the hierarchy, a module instance is treated as a "black-box" – the internals are unknown



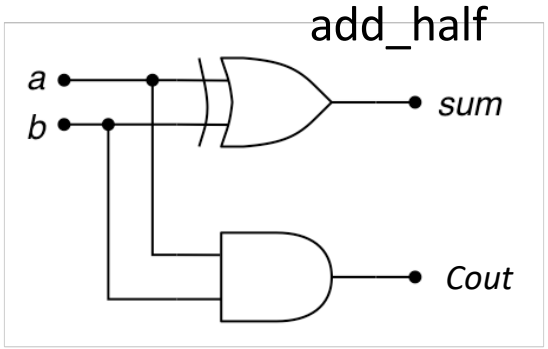
# Instantiation in Verilog

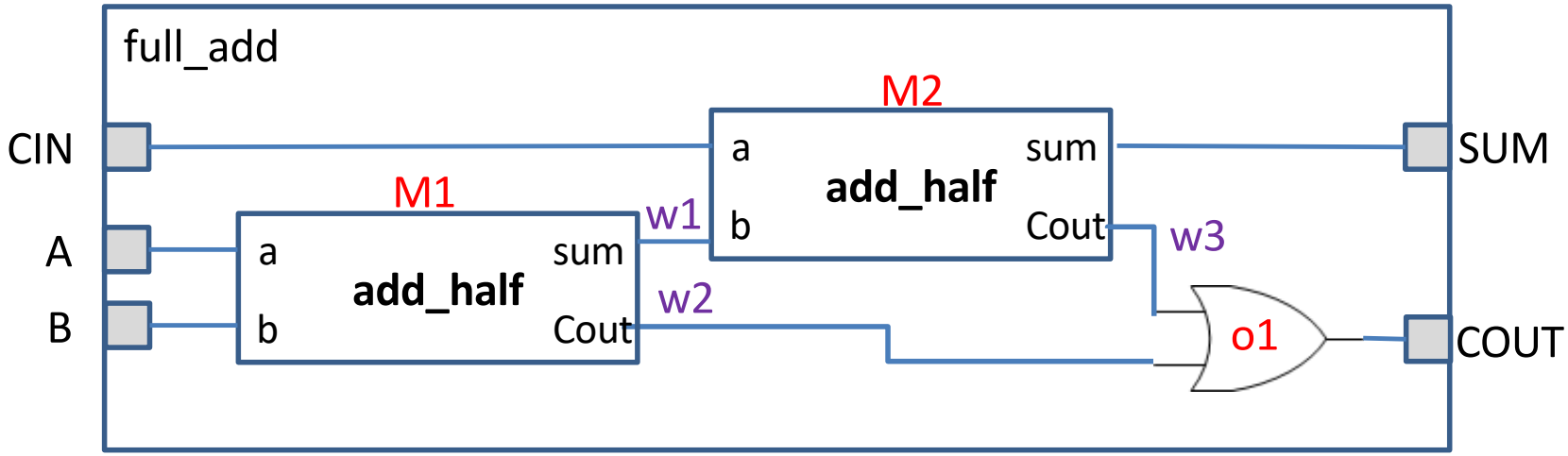
- We can ***instantiate*** the half adder module twice to design a full adder



# Instantiation in Verilog (Full Adder)

```
module add_half (input a, b,
                 output sum, Cout);
  xor g1 (sum, a, b);
  and g2 (Cout, a, b);
endmodule
```



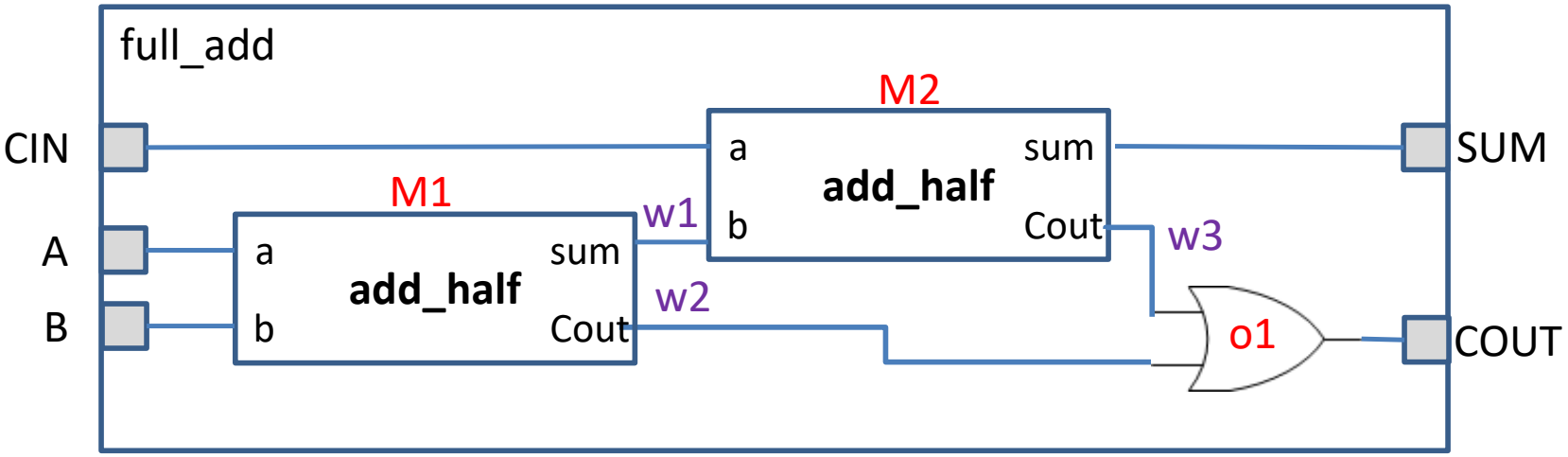

```
module full_add (input A, B, CIN,
                 output SUM, COUT);
    wire  w1, w2, w3;

    add_half M1 (A, B, w1, w2);
    add_half M2 (CIN, w1, SUM, w3);
    or o1 (COUT, w3, w2);
endmodule
```

# Instantiation in Verilog

- We ***instantiate*** a module by invoking its name and giving an ***instance*** a name:

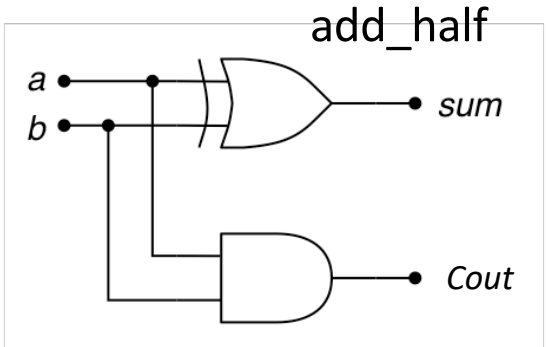
```
add_half M1 (A, B, w1, w2);
```

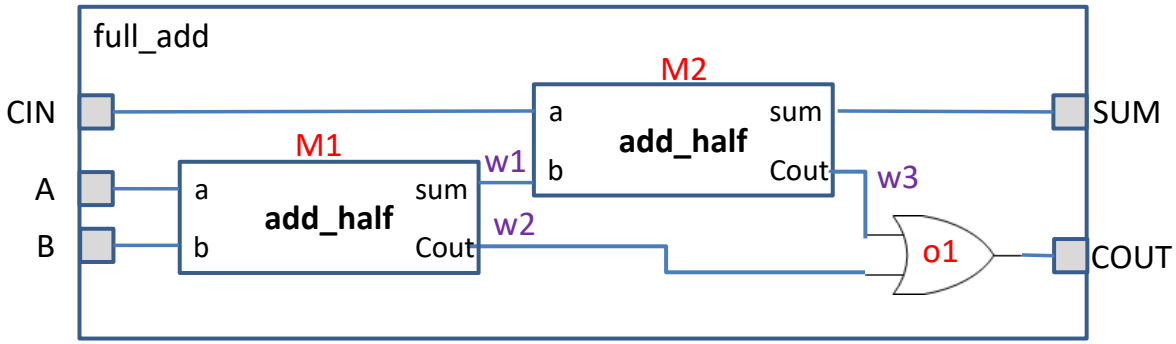
- Creates an ***instance*** called **M1** of module **add_half**



# *Ordered* Instantiation

- The **order** determines connections
- Looking at the original **add_half** module declaration:
- Its first port is its ***a*** input, the second its ***b*** input, the third its ***sum*** output and the fourth its ***Cout*** output
- Connects a wire called ***A*** in the outer module with the ***a*** port, ***B*** with the ***b*** port, the ***w1*** wire to its ***sum*** output and the ***w2*** wire to its ***Cout*** port




```
module add_half (input a, b,
                 output sum, Cout);
  xor g1 (sum, a, b);
  and g2 (Cout, a, b);
endmodule
```

```
add_half M1 (A, B, w1, w2);
```

```
add_half M2 (CIN, w1, SUM, w3);
```

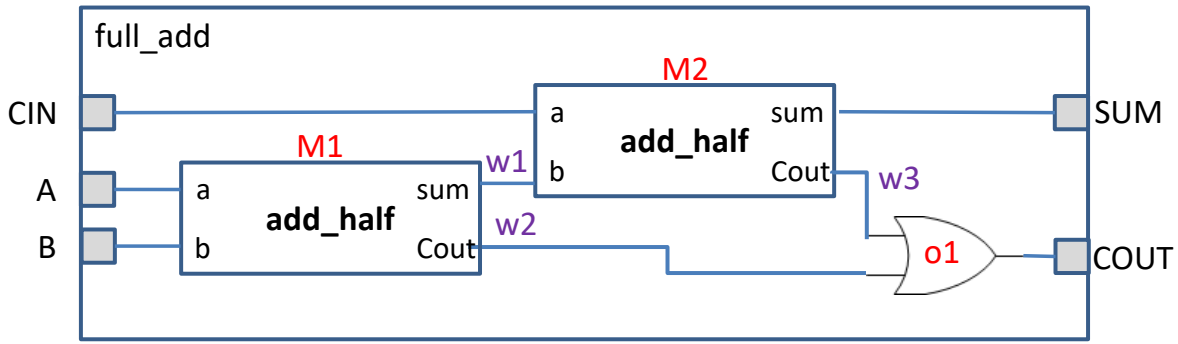
# *Named* Instantiation

- Connecting instantiated modules to wires and ports in the manner just described can be **error-prone**:
- Hence, we generally use a ***named connection***:

```
add_half M1 (.a(A),
             .b(B),
             .sum(w1),
             .Cout(w2));
```

```
add_half M2 (.a(CIN),
             .b(w1),
             .sum(SUM),
             .Cout(w3));
```

**The port name is preceded by a dot, and the connected signal is placed in the brackets**



# END OF L13,L14 SUMMARY