Welcome to The Hardware Lab!

Fall 2018

Verilog Data Flow and Data Types

Prof. Chun-Yi Lee

Department of Computer Science

National Tsing Hua University

Agenda

- **Further Discussion on Gate-Level Design**
- Basic Concepts of CPU Execution
- Verilog Data Types
- Verilog Data Flow
- 7-Segment Display

Today's class will help you:

1. Introduce the concepts of universal logic gates, CPU, and memory hierarchy
2. Understand various Verilog data types
3. Understand the difference among gate-level, data flow, and behavioral modeling techniques
4. Understand the basic concepts of 7-segment display

Issues from Lab 1 (1/2)

- Different port mapping styles

Your module

```
module My_Module (out, a, b, c);
output out;
input a,b,sel;

endmodule
```

Your top module

M1, M2, and M3 are the same

```
module My_Top_Module (X1, X2, X3, X4, Out);
input X1, X2, X3, X4;
output Out;

My_Module M1(Out, X1, X2, X3);                    // connect by ordered lists
My_Module M2(.out(Out), .a(X1), .b(X2), .c(X3)); //connect by port names
My_Module M3(.b(X2), .a(X1), .c(X3), .out(Out)); //connect by port names
// You can reorder the parameter list as in M3
endmodule
```

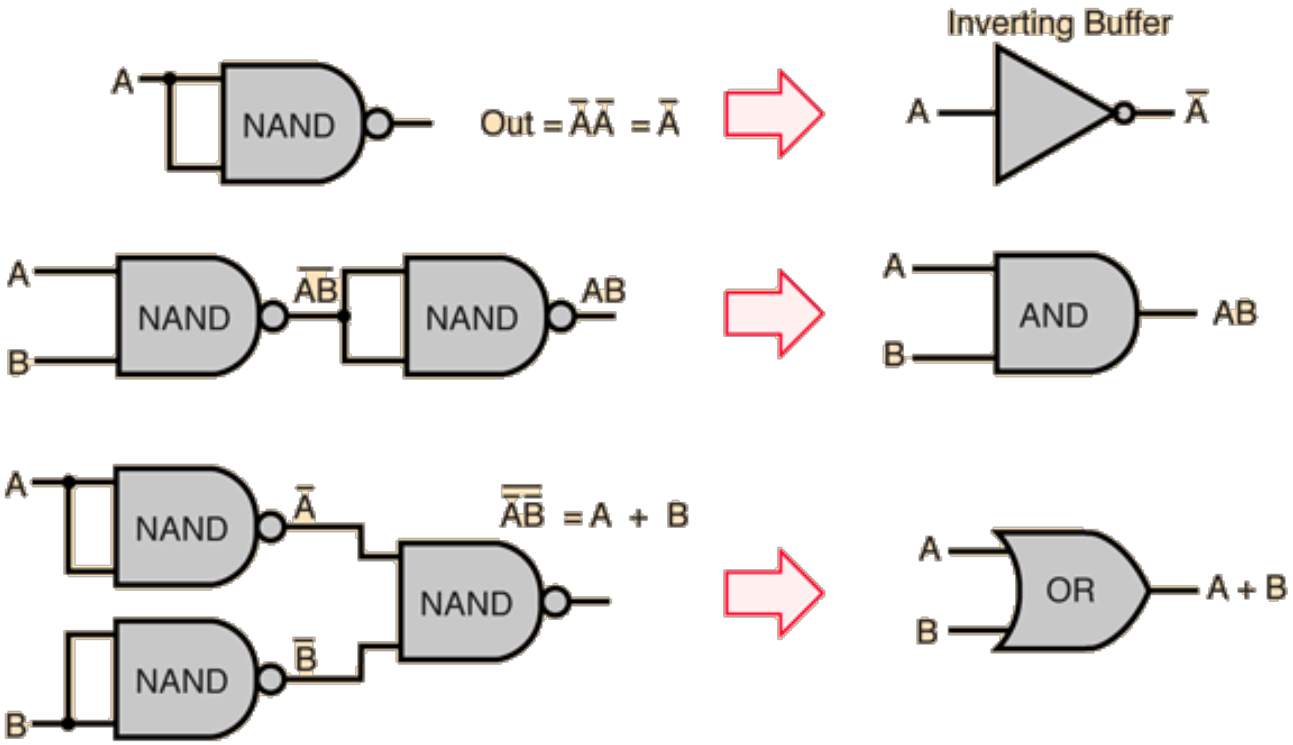
Issues from Lab 1 (2/2)

- For primitive logic gates, you **NEED** to declare gate names
  - Compiler dependent
  - Ensure the execution of your design

```
wire        OUT, IN, IN1, IN2;

and         a1(OUT, IN1, IN2);
nand        na1(OUT, IN1, IN2);
or          or1(OUT, IN1, IN2);
nor         nor1(OUT, IN1, IN2);
xor         x1(OUT, IN1, IN2);
xnor        nx1(OUT, IN1, IN2);
not         n1(OUT, IN);
```

Universal Logic Gates

- A universal logic gate is a logic gate that can be used to construct all other logic gates
- **NAND** and **NOR** are two examples of universal logic gates



Agenda

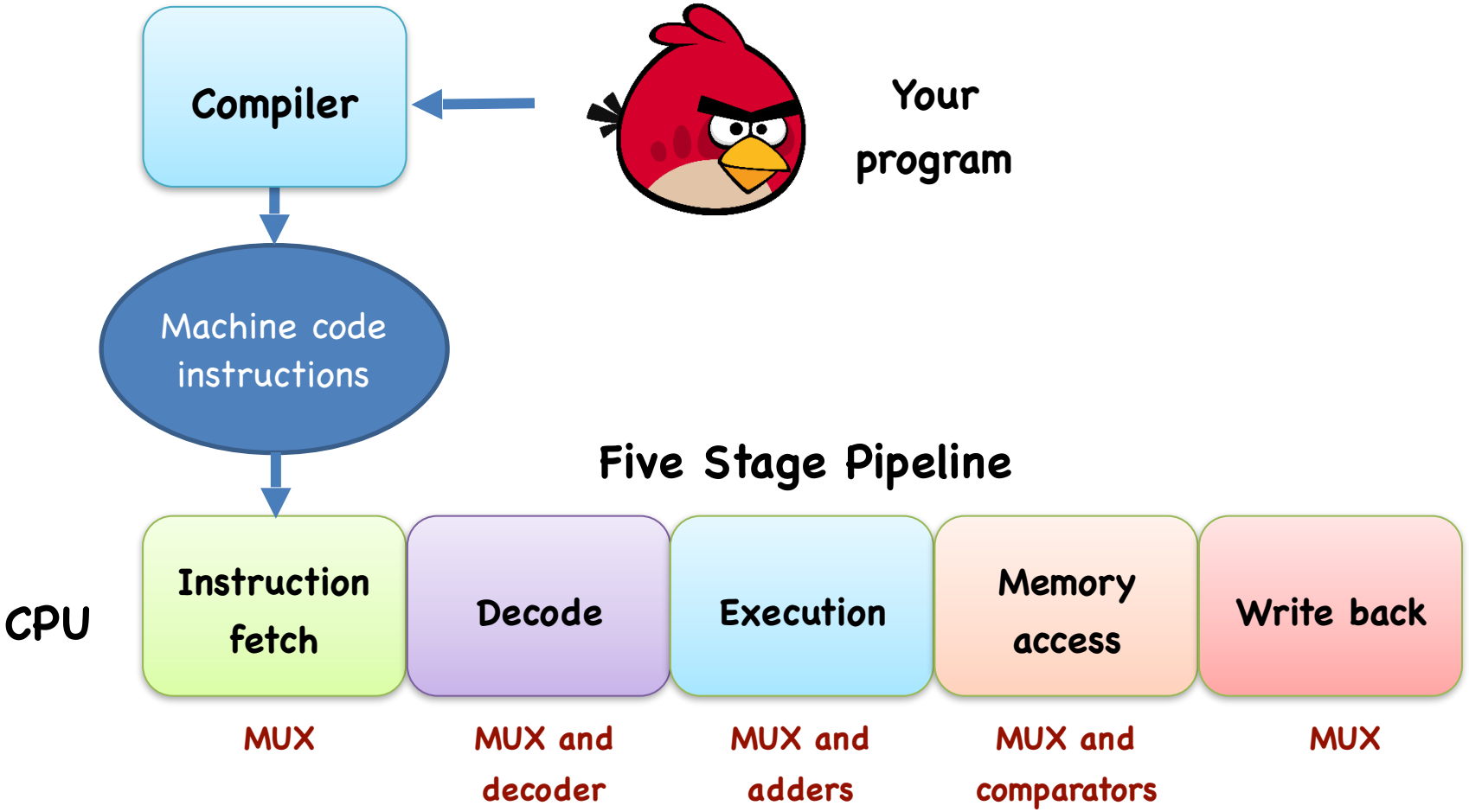
- Further Discussion on Gate-Level Design
- **Basic Concepts of CPU Execution**
- Verilog Data Types
- Verilog Data Flow
- 7-Segment Display

Today's class will help you:

1. Introduce the concepts of universal logic gates, CPU, and memory hierarchy
2. Understand various Verilog data types
3. Understand the difference among gate-level, data flow, and behavioral modeling techniques
4. Understand the basic concepts of 7-segment display

Basic Concepts of CPU Execution

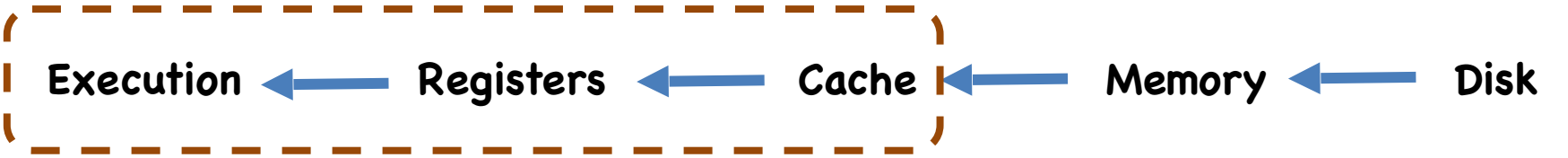
- In a simple baseline CPU, there are five pipeline stages



*Pictures cited from www.rovio.com for education purpose only

Memory System Hierarchy

- Memory system hierarchy is similar to **a library system**

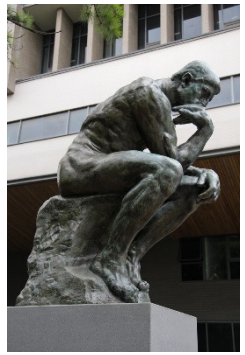


You study hard

You take notes

You have a pile of books on the desk

You go to the bookshelves



You go to other libraries

*Pictures cited from www.cbs.com, blogs.independent.co.uk, hopkins.typepad.com, commons.wikipedia.org, and www.nthu.edu.tw for education purpose only

Agenda

- Further Discussion on Gate-Level Design
- Basic Concepts of CPU Execution
- **Verilog Data Types**
- Verilog Data Flow
- 7-Segment Display

Today's class will help you:

1. Introduce the concepts of universal logic gates, CPU, and memory hierarchy
2. Understand various Verilog data types
3. Understand the difference among gate-level, data flow, and behavioral modeling techniques
4. Understand the basic concepts of 7-segment display

Verilog Data Types

- Two data types in Verilog
  - Net data type
  - Register data type
- Perform logic operations using variables of these data types
- Draw your circuits FIRST before deciding your data types

Value Set

- The net types in Verilog can be one of these four states

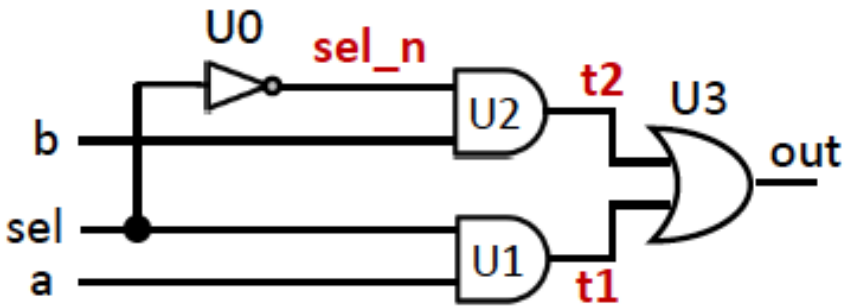
Net	Reg	Representation
0	0	Logic zero, false condition
1	1	Logic one, true condition
x		Unknown logic value
z		High impedance, floating state

- Please only set the value of a reg to be either **0** or **1**
  - Registers of real circuits do not have values of either **x** or **z**

- It is a better habit to CLEARLY specify the values in Verilog

Net Data Type

- Net type is basically the wires
  - Declare by the keyword **wire**
- Can be used to connect different modules
- No storage functionality
- Default value is **z**




```
module MUX(out, a, b, sel);

output out;
input a,b,sel;

wire sel_n,t1,t2;

not U0(sel_n,sel);
and U1(t1,a,sel);
and U2(t2,b,sel_n);
or U3(out,t1,t2);

endmodule
```

Register Data Type

- Register data type represents a data storage element **or a net**
  - Depends on how they are used in the **always** block in modules
  - Does not represent anything in a testbench
  - Declare by the keyword **reg**
- Can have storage functionality
  - If it is a data storage element
- Default value is **x**
- **reg** assignment (**=**) only occurs in either an **initial** or an **always** block

Register Data Type in Testbench

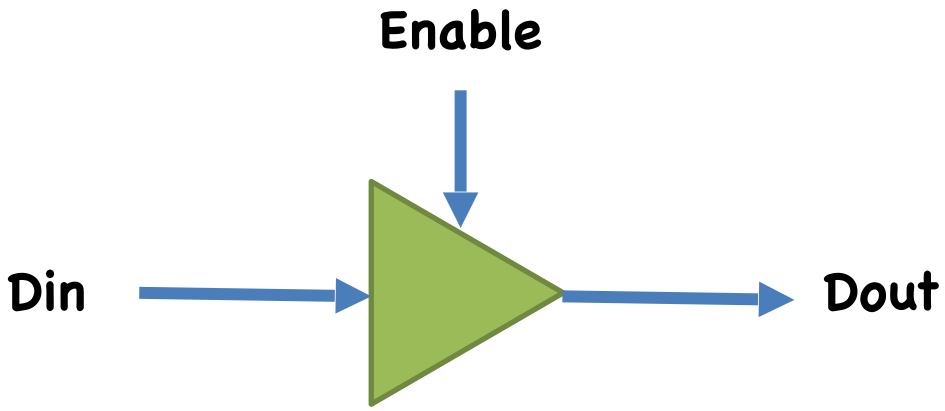
- Variables of reg data type in testbenches are not synthesized
- They are simply variables to be fed into your modules

```
`timescale 1ns / 1ps                    // Simulation Unit / Accuracy
module      test_Nand_Latch_1;          // Testbench module
reg         preset, clear;              // Inputs should be declared as reg
wire        q, qbar;                    // Outputs should be declared as wire

 Nand_Latch_1 M1 (q, qbar, preset, clear);   // Instantiate YOUR DESIGN module

 always
 begin                                  // always condition: The description always happens
   #20       clear = !clear;            // The value of clear inverts every 20 ns
 end

 initial                                // Initial conditions
 begin
           preset = 1'b0;   clear = 1'b1;
   #10                                  // Units of "Simulation Units".  In this case, 10ns
           preset = 1'b1;
 end

endmodule
```


*Pictures cited from Prof. Ting-Chi Wang's 2013 slides for education purpose only

Tri State Buffer

- A tri state buffer can be one of the three states: **1**, **0**, and **z**
- When enable equals 1, Din can pass its value to Dout
- Otherwise, Dout is floating (**z** state)




```
assign Dout = (Enable == 1'b1) ? Din : 1'bz;
```

Vectors

- Both data types can be declared as vectors
  - Similar to what we have done for inputs and outputs
  - Examples
    - reg [7:0] reg_vector1, reg_vector2;
    - wire [5:0] data_bus;
- You can assign only part of a vector to another

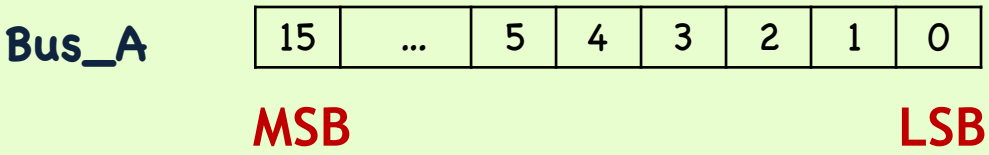
```
wire    [7:0] a, b, y;

assign y[7:4] = a[3:0];
assign y[3:0] = b[7:4];
```

MSB and LSB

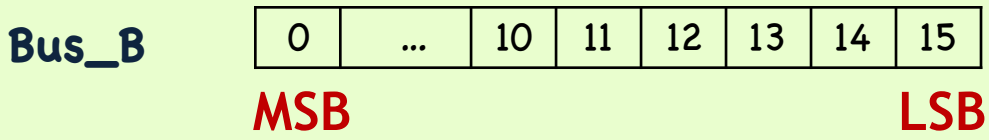
- Vector definition can determine its Most Significant Bit (**MSB**) and Least Significant Bit (**LSB**)

- Important for arithmetic operations

  - E.g., **wire** Bus_A[15:0];



  - E.g., **wire** Bus_B[0:15];



- There is a difference between **Bus_A+1'b1** and **Bus_B+1'b1**

Agenda

- Further Discussion on Gate-Level Design
- Basic Concepts of CPU Execution
- Verilog Data Types
- **Verilog Data Flow**
- 7-Segment Display

Today's class will help you:

1. Introduce the concepts of universal logic gates, CPU, and memory hierarchy
2. Understand various Verilog data types
3. Understand the difference among gate-level, data flow, and behavioral modeling techniques
4. Understand the basic concepts of 7-segment display

Verilog Modeling

- Gate level modeling

```
and     a1(OUT, IN1, IN2);
nand    na1(OUT, IN1, IN2);
or      or1(OUT, IN1, IN2);
```

- Data flow modeling

E.g., **assign** a[7:0] = b[7:0] & c[7:0]; (bitwise AND)

- Behavior modeling **(in always blocks)**

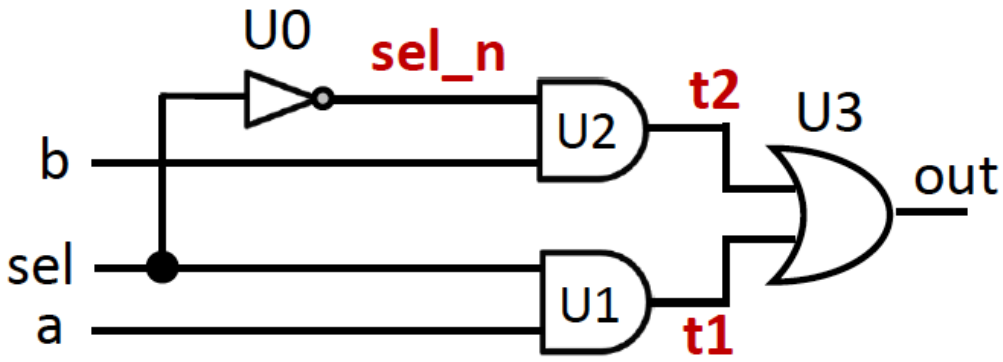
E.g.,

```
if (sel == 1'b1)    begin
                S = A;
end
else                begin
                S = B;
end
```

Data Flow Modeling

- For combinational circuits only
- Similar to logic expressions
- Easier for development
- Need to be synthesized to the gate level
- Usage
  - Continuous assignment
  - Conditional assignment
  - Operators

Comparison of Data Flow & Gate Level Modeling



Gate level modeling

```
module MUX(out, a, b, sel);
output out;
input a,b,sel;
wire sel_n,t1,t2;

not         U0(sel_n,sel);
and         U1(t1,a,sel);
and         U2(t2,b,sel_n);
or          U3(out,t1,t2);

endmodule
```

```
module MUX2(out, a, b, sel);

output out;
input a,b,sel;

assign out = (a&sel) | (b& ~sel);
// assign out = sel? a: b;

endmodule
```

Data flow modeling

*Pictures cited from Prof. Shih-Chieh Chang's 2010 slides for education purpose only

Continuous Assignment

- Syntax:
  - **assign** [the name of **lvalue**] = **rvalue**;
- Assignment can be done for a single wire or a bus
  - **E.g.,** **assign** a[7:0] = b[7:0];
- **lvalue** must be of type **wire** or **output** (but not **reg** type)
- **rvalue** can be an **expression**
  - **E.g.,** **assign** a[7:0] = b[7:0] & c[7:0]; (bitwise AND)
- Cascade is allowed
  - **E.g.,** **assign** {c_out, sum[3:0]} = a[3:0] + b[3:0] + c_in;

Multiple Assignments

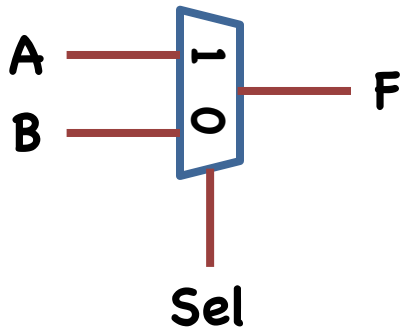
- You can do multiple assignments in a single line
  - **assign** S1 = A1 & B1, S2 = A2 | B2, S3 = A3 + B3;
  - However, it is a better habit to separate different assignment to different lines

Conditional Operator

- Evaluation depending on the value of condition
- Syntax:
  - (Condition) ? (Expression 1) : (Expression 2);
    TRUE FALSE

```
module simple_mux (F, A, B, Sel);
  input   A, B, Sel;
  output F;

  assign F = (Sel == 1'b1) ? A : B;

endmodule
```

Verilog



Schematic

Verilog Operators (1/3)

Operator Type	Usage	Description
Arithmetic	m * n	Multiply m by n
	m / n	Divide m by n (**Usually** Not Synthesizable)
	m + n	Add n to m
	m - n	Subtract n from m
	m % n	Modulus of m / n (**Usually** Not Synthesizable)
Logical	!m	Is m not true? (1-bit True/False result)
	m && n	Are both m and n true? (1-bit True/False result)
	m || n	Are either m or n true? (1-bit True/False result)
Relational	m > n	Is m greater than n? (1-bit True/False result)
	m < n	Is m less than n? (1-bit True/False result)
	m >= n	Is m greater than or equal to n? (1-bit True/False result)
	m <= n	Is m less than or equal to n? (1-bit True/False result)
Equality	m == n	Is m equal to n? (1-bit True/False result)
	m != n	Is m not equal to n? (1-bit True/False result)

Verilog Operators (2/3)

Operator Type	Usage	Description
Bitwise	~m	Invert each bit of m
	m & n	AND each bit of m with each bit of n
	m | n	OR each bit of m with each bit of n
	m ^n	Exclusive OR each bit of m with n
	m ~^ n m ^~ n	Exclusive NOR each bit of m with n
Reduction	&m	AND all bits in m together (1-bit result)
	~&m	NAND all bits in m together (1-bit result)
	|m	OR all bits in m together (1-bit result)
	~|m	NOR all bits in m together (1-bit result)
	^m	Exclusive OR all bits in m (1-bit result)
	~^m ^~m	Exclusive NOR all bits in m (1-bit result)

Verilog Operators (3/3)

Operator Type	Usage	Description
Shift	m >> n	Logical shift m right n bits (fill with zero)
	m << n	Logical shift m left n bits (fill with zero)
	m >>> n	Arithmetic shift m right n bits • Keep sign for **signed** variables • The same as >> for **unsigned** variables
	m <<< n	Arithmetic shift m left n bits • Keep sign for **signed** variables • The same as << for **unsigned** variables
Conditional	sel ? m : n	If sel is true, select m: else select n
Concatenation	{m, n}	Concatenate m to n, creating larger vector
Replication	{n{m}}	Replicate m n-times

Verilog Operators Precedence

- It is better to use parentheses for readability

Operator Type	Symbol	Description
Unary	+ - ! ~	Highest precedence
Multiply, Divide, Modulus	* / %	
Add, Subtract	+ -	
Shift	<< >>	
Relational	< <= > >=	
Equality	== !=	
Reduction	& ~& | ~| ^ ~^	
Logical	&& ||	
Conditional	? :	Lowest precedence

Concatenation and Replication

- Assignment can be done by concatenation and replication

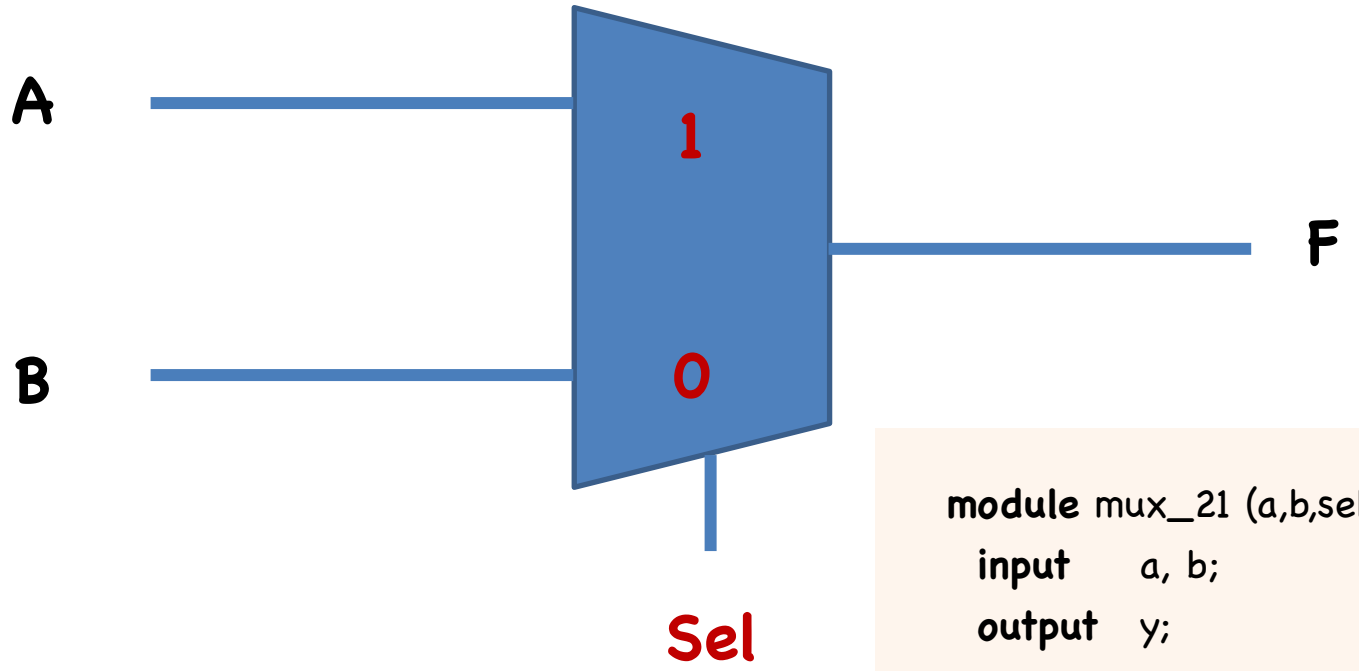
- Concatenation
  - Suppose **A = 1'b1**, **B = 3'b010**
  - assign **Y** = { **A, 4'hc**, **B** }; // ***Result Y is 8'b1_1100_010***
  - assign **Y** = { **A**, **B[0]** }; // ***Result Y is 2'b10***

- Replication
  - assign **Y** = { 4{**A**} }; // ***Result Y is 4'b1111***
  - assign **Y** = { 2{**A**}, 2{**B**}, **A** }; // ***Result Y is 9'b1_1_010_010_1***

***** Please note that **Y** should be declared with sufficient bits to accommodate **rvalue**

Lab 1 in Data Flow Modeling (1/3)

- 1-bit **2-to-1 MUX**




```
module mux_21 (a,b,sel,y);
  input    a, b;
  output   y;
  input    sel;

  assign   y = (sel==1'b1) ? a : b;
endmodule
```

Lab 1 in Data Flow Modeling (2/3)

- 4 x 16 decoder


```
assign Dout = (Din == 4'b0000 ) ? 16'b1000_0000_0000_0000 :
              (Din == 4'b0001 ) ? 16'b0100_0000_0000_0000 :
              ……
              (Din == 4'b1111 ) ? 16'b0000_0000_0000_0001 :
              16'b0000_0000_0000_0000;
```

- The design will be slow if you cascade too many conditional operators

Lab 1 in Data Flow Modeling (3/3)

- 1-bit **Full Adder**
- The logic expression is as follows
  - Sum = A ⊕ B ⊕ C_in
  - C_out = A·B + A·C_in + B·C_in
- A full adder can be described in the following expression
  - **assign** {C_out, Sum} = A + B + C_in;

Agenda

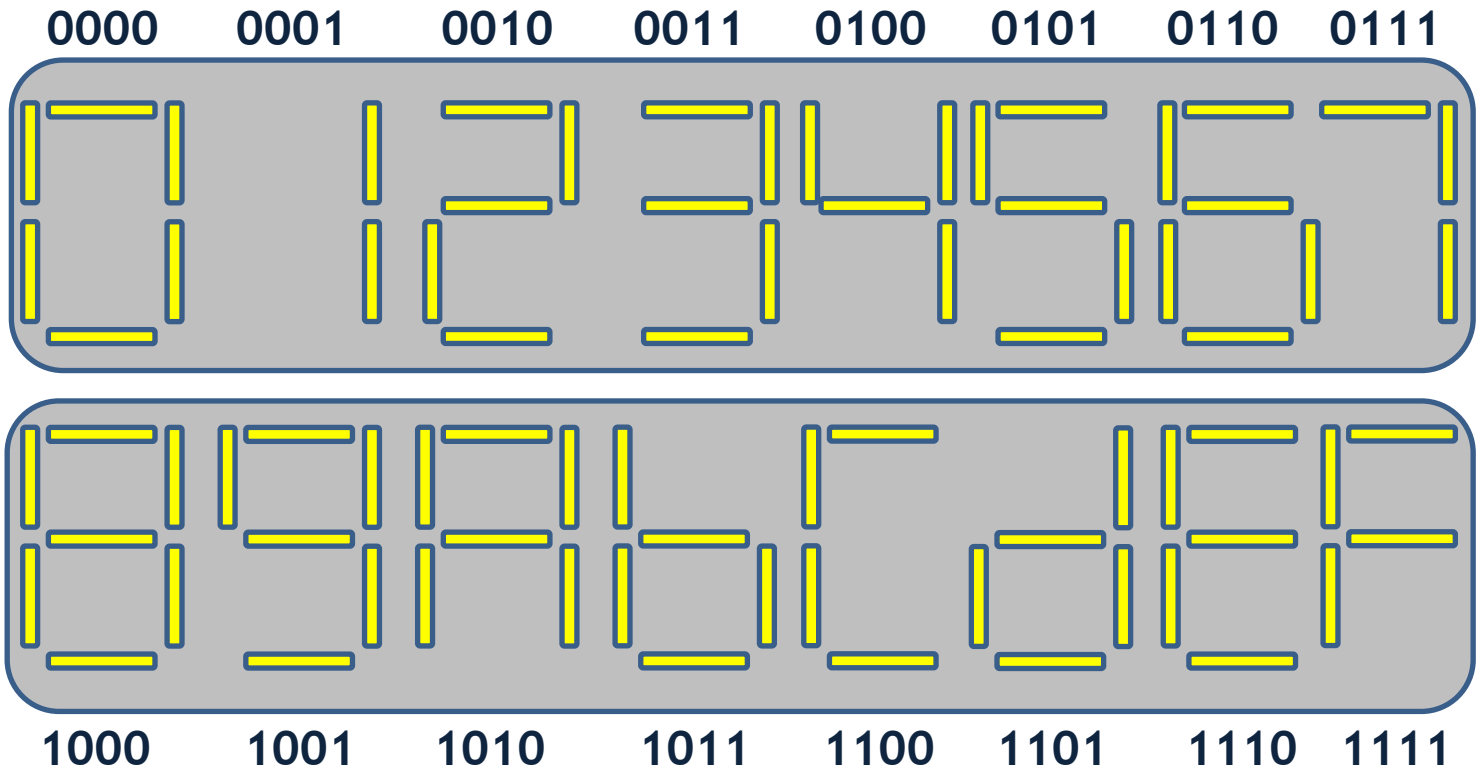
- Further Discussion on Gate-Level Design
- Basic Concepts of CPU Execution
- Verilog Data Types
- Verilog Data Flow
- 7-Segment Display

Today's class will help you:

1. Introduce the concepts of universal logic gates, CPU, and memory hierarchy
2. Understand various Verilog data types
3. Understand the difference among gate-level, data flow, and behavioral modeling techniques
4. Understand the basic concepts of 7-segment display

7-Segment Display

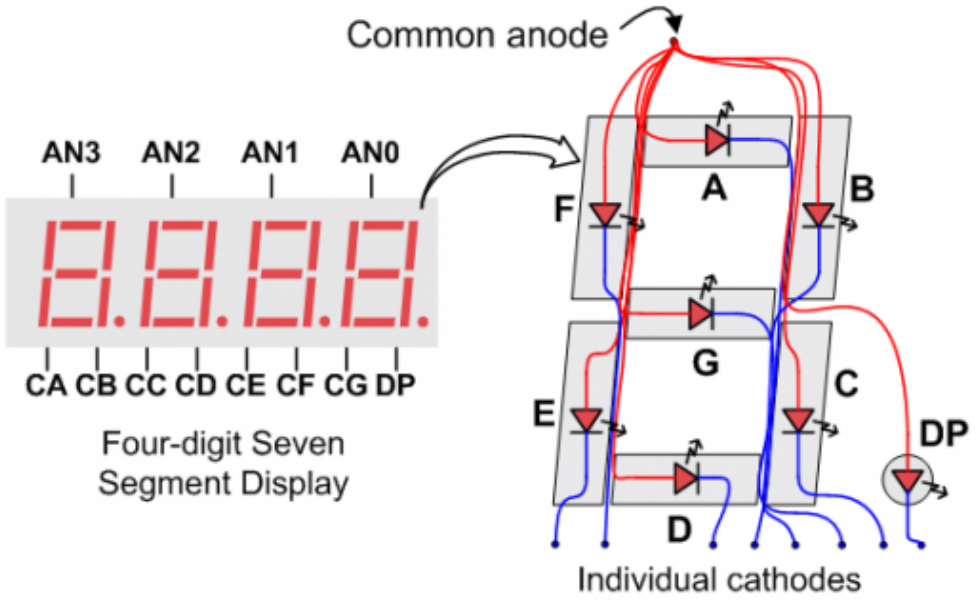
- As its name suggests, it has seven segments
  - Seven pins to control, plus one dot pin
  - Set the pin values to 1 or 0 to display digits



7-Segment Display Control (1/2)

- Four enable signals for the four display units
  - **AN[3:0]** correspond to **the four digits**
  - **AN** is used to **ENABLE** one of **the four digits**
  - Set one bit of AN[3:0] to **LOW** to illuminate one digit

AN Bit	FPGA Pin
3	W4
2	V4
1	U4
0	U2

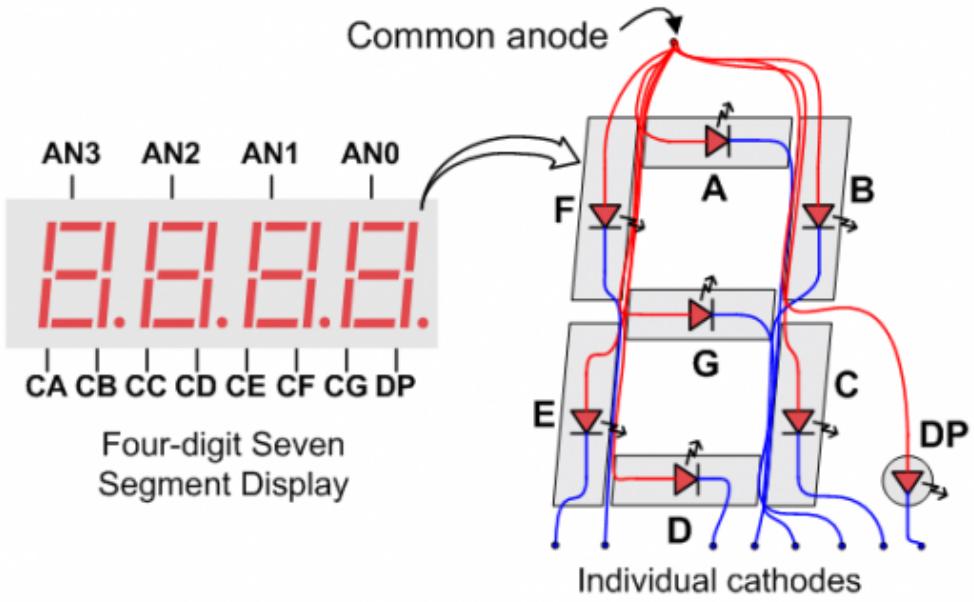


Digit	AN[3:0]
3	0111
2	1011
1	1101
0	1110

*Pictures cited from www.xilinx.com for education purpose only

7-Segment Display Control (2/2)

- Eight signals for the segments and dot
  - **seg[0:6]** in the XDC file correspond to **segments A~G** shown below
  - **dp** in the XDC file corresponding to **DP (dot)**
  - To illuminate a segment, set its value to **LOW**



Symbol	FPGA Pin
CA	W7
CB	W6
CC	U8
CD	V8
CE	U5
CF	V5
CG	U7
DP	V7

*Pictures cited from www.xilinx.com for education purpose only

Thank you for your attention!

*Bay Bridge night view taken at Ferry Building, San Francisco, CA, USA.
This picture is taken by Chun-Yi Lee himself, who is also a fan of photography