



Welcome to The Logic Design Lab!

Fall 2021
Verilog Introduction and
FPGA Implementation

Prof. Chun-Yi Lee

Department of Computer Science
National Tsing Hua University

Announcement

- Please register an account at the CAD server
 - After your account is approved, please take a look at our **CAD Tutorial**
 - We will give a tutorial on how to use the CAD server on 9/16 (Thu)
- Due to COVID-19 and the remote class, please install Vivado on your desktops or laptops by following our **Vivado Installation Guide**

Agenda

- Verilog Introduction
- FPGA Implementation



*Pictures cited from Marvel.com for education purpose only

IC Design Flow



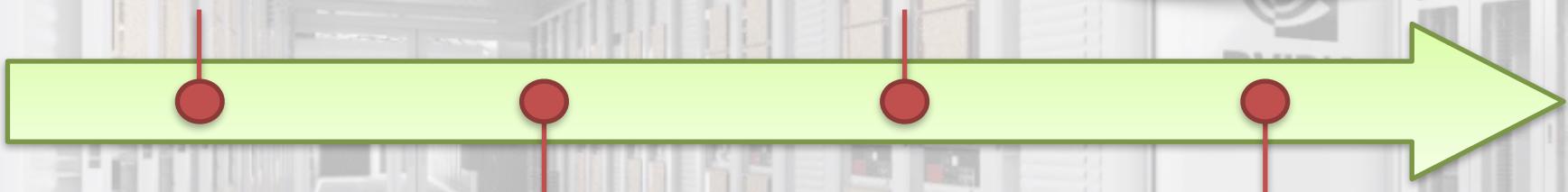
System specification

WE
ARE
HERE

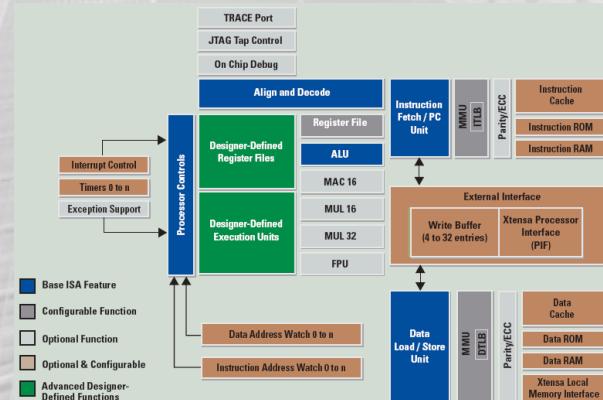
We are here

Verilog

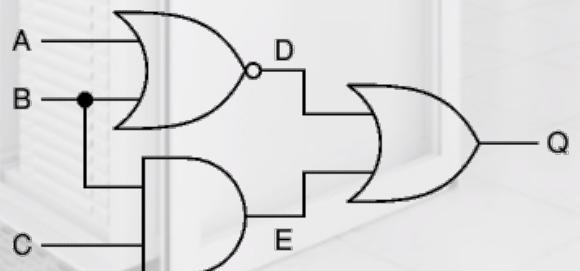
Logic and circuit
design



Function and
architecture design



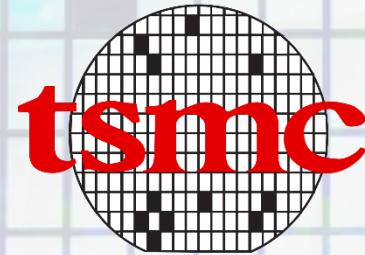
Logic synthesis



IC Design Flow (Con'd)

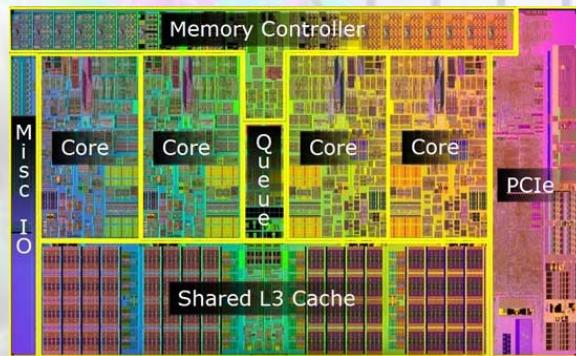


Verification



台灣積體電路製造股份有限公司
Taiwan Semiconductor Manufacturing Company, Ltd.

Fabrication



FPGA: Fast Prototyping

- FPGA: Field Programmable Gate Array
- FPGA can be used to test the circuit design before fabrication
- Fast prototyping: concept similar to 3D printer



FPGA

3D
printer



FPGA: Programmability

- Programmable hardware
- Customizable for different functionality
- Fast and easy prototyping and deployment



Embedded mobile devices



Drone (Unmanned Aerial Vehicles)



Robotics



Data centers

Hardware Description Language

C++, C, Java,
Android,
Swift, etc.

Develop programs on
CPUs and/or GPUs



HDL:
Verilog & VHDL
Design, model, and
verify microelectronics



Schematic vs. HDL

Schematic

GUI-based design

Slow but easy

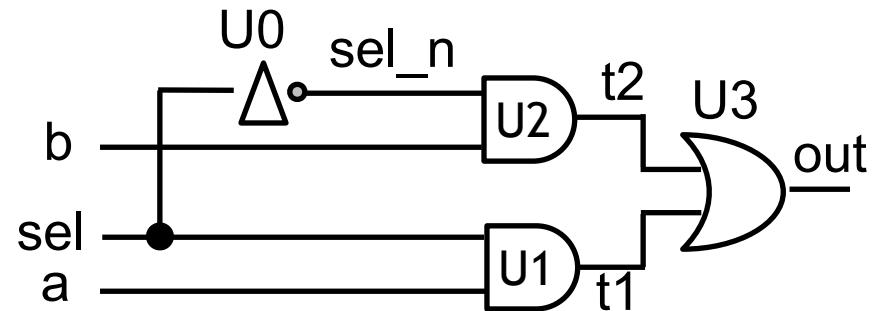
User Friendly

Verilog

Text-based design

Fast, need experience

Highly portable



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

```
output out;  
input a,b,sel;
```

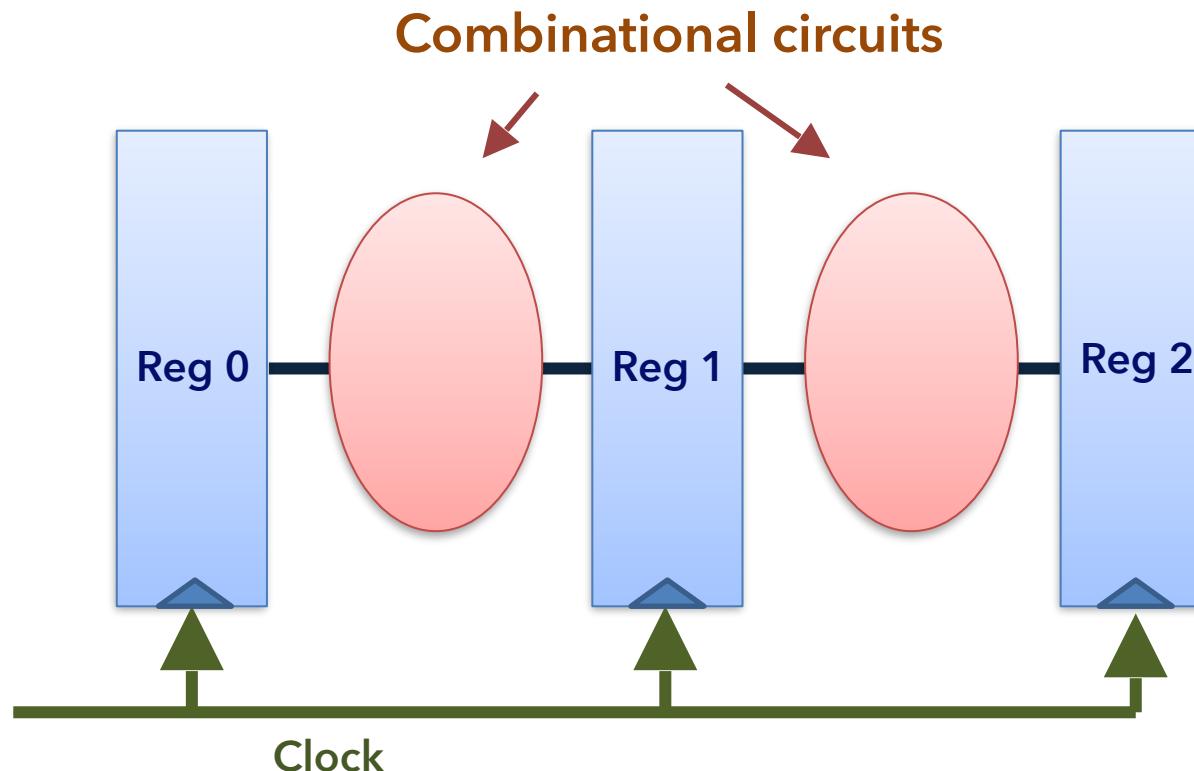
```
wire sel_n,t1,t2;
```

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

```
endmodule
```

Register Transfer Level (RTL)

- Describes the behavior of combinational circuits between registers



Design Concerns



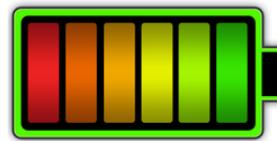
Time to market



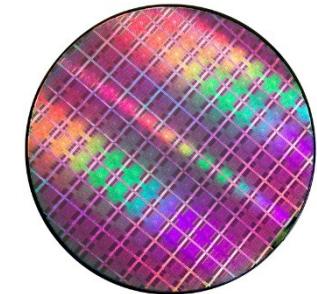
Money

3GHz

Timing



Power



Area

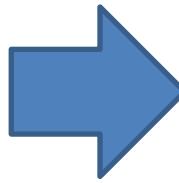
Timing

- High clock rate requirement
- Margin and reliability

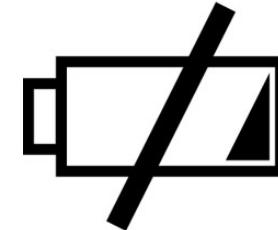


Power

- Battery life
- Need for multimedia, games, and high performance apps

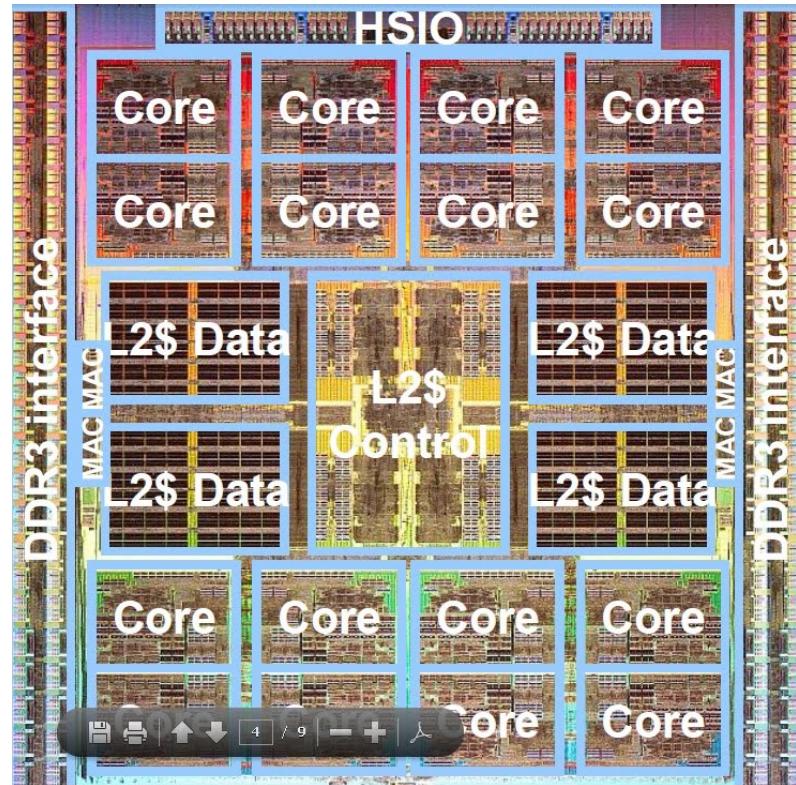


No battery!



Area

- Tape out is costly
 - Millions of dollars for the most advanced technologies
- More functionality
- Miniaturization



Agenda

- **Verilog Introduction**
 - **Modules**
 - **Lexical conventions**
- **FPGA Implementation**



Modules

- Basic building block in Verilog

Verilog Module

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

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

endmodule
```

C++ Class

```
class hello_world
{
public:
    int x;
private:
    void get_x();

};
```

Module Structure

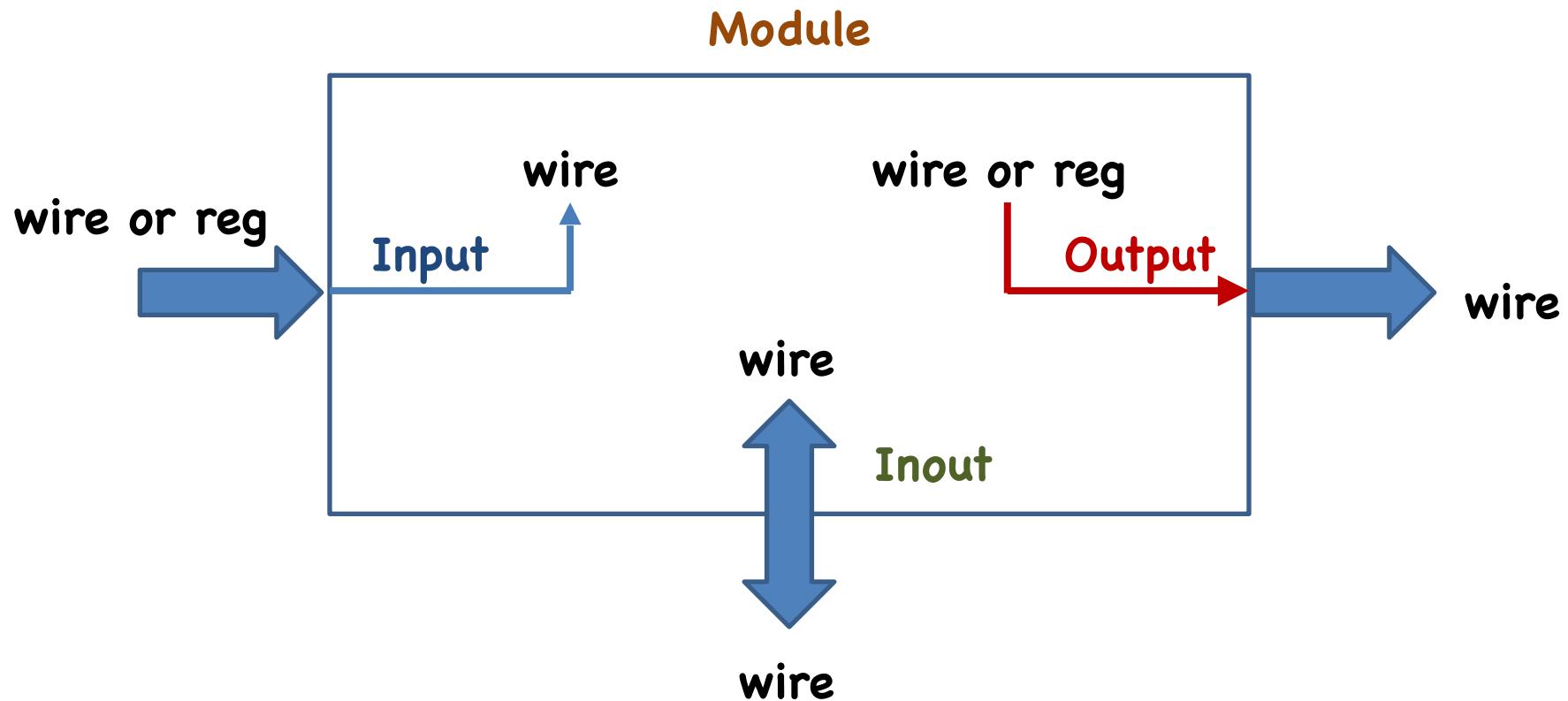
- Module name is **CASE SENSITIVE**

Module Name	Module Ports	
module Add_half (sum, c_out, a, b);		
input	a,b;	Declaration of Ports
output	sum, c_out;	
wire	c_out_bar;	
xor	g1(sum, a, b);	Declaration of Internal Signal
nand	g2(c_out_bar, a, b);	
not	g3(c_out, c_out_bar);	
endmodule		

Note: All bold-faced items are Verilog keywords.

Module Ports

- Three types of module ports
 - **input**, **output**, or **inout** (only for experienced users)



Primitive Logic Gates

- Basic Logic gates are provided as pre-defined primitives
 - **and, or, xor, nand, nor, xnor, not**
- The first terminal is the output and the rests are the inputs
 - How to use them? Easy!

wire	OUT, IN, IN1, IN2, IN3;
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);

Smart Primitives

```
module      nand3 (O, A1, A2, A3);
```

```
    input      A1, A2, A3;
```

```
    output     O;
```

```
nand      NAND1 (O, A1, A2, A3);
```

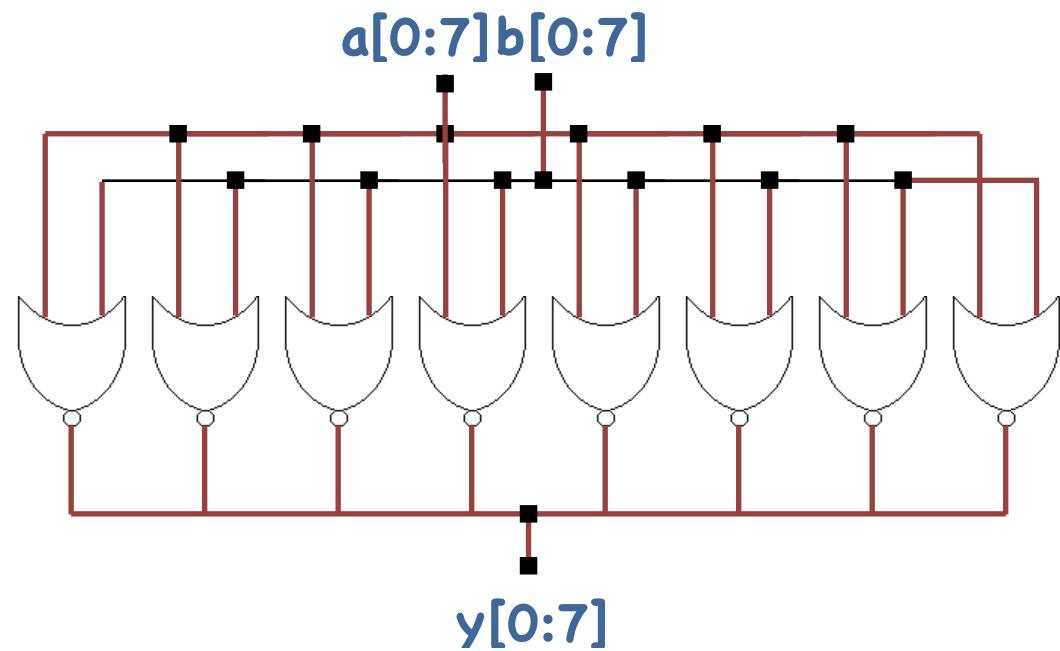
```
endmodule
```

Remember to give a name for it!

- Same primitive can be used to describe for **arbitrary number** of inputs
- Except for “**not**”, which takes **ONLY ONE** input
- Not gate can generate **multiple number of outputs**
 - E.g., not U (O1, O2, ..., ON, IN), where O1~ON equal the inverse of IN

Array of Instances

```
module array_of_nor (y, a, b);  
    input [7:0] a, b;  
    output [7:0] y;  
  
    nor Nor_Array [7:0] (y, a, b);  
  
endmodule
```

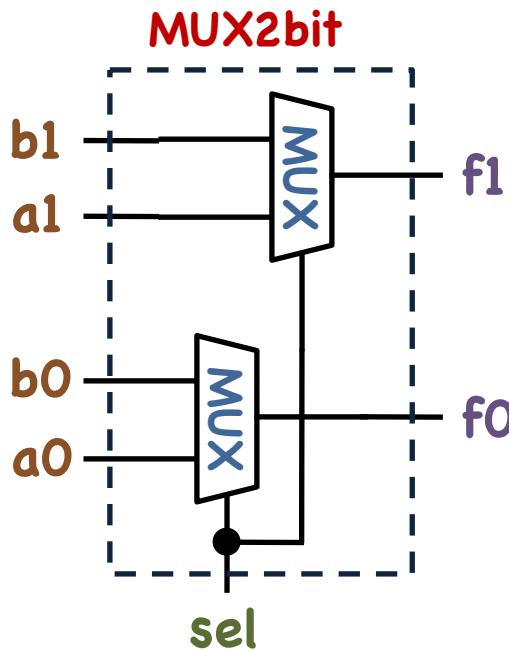


Gate Level Design

- At gate level, the circuit is described in terms of primitive logic gates
 - Typically multiple sizes of **and**, **or**, **xor**, **nand**, **nor**, **xnor**, **not**
- Hardware design at this level is about one-to-one correspondence between the logic circuit design and the Verilog description
 - Essential for new fabrication technologies
 - Gate level design allows the designer to have full control of the fabricated circuits
- If you plan to work for large technology companies such as Apple, Qualcomm, AMD, Intel, etc., gate level design is **EXTREMELY** important
 - Gate level design allows better control of **timing**, **power**, and **area**

Instantiation of Modules

- Modules can be instantiated as **objects** in another module
 - Use **module name** for instantiation

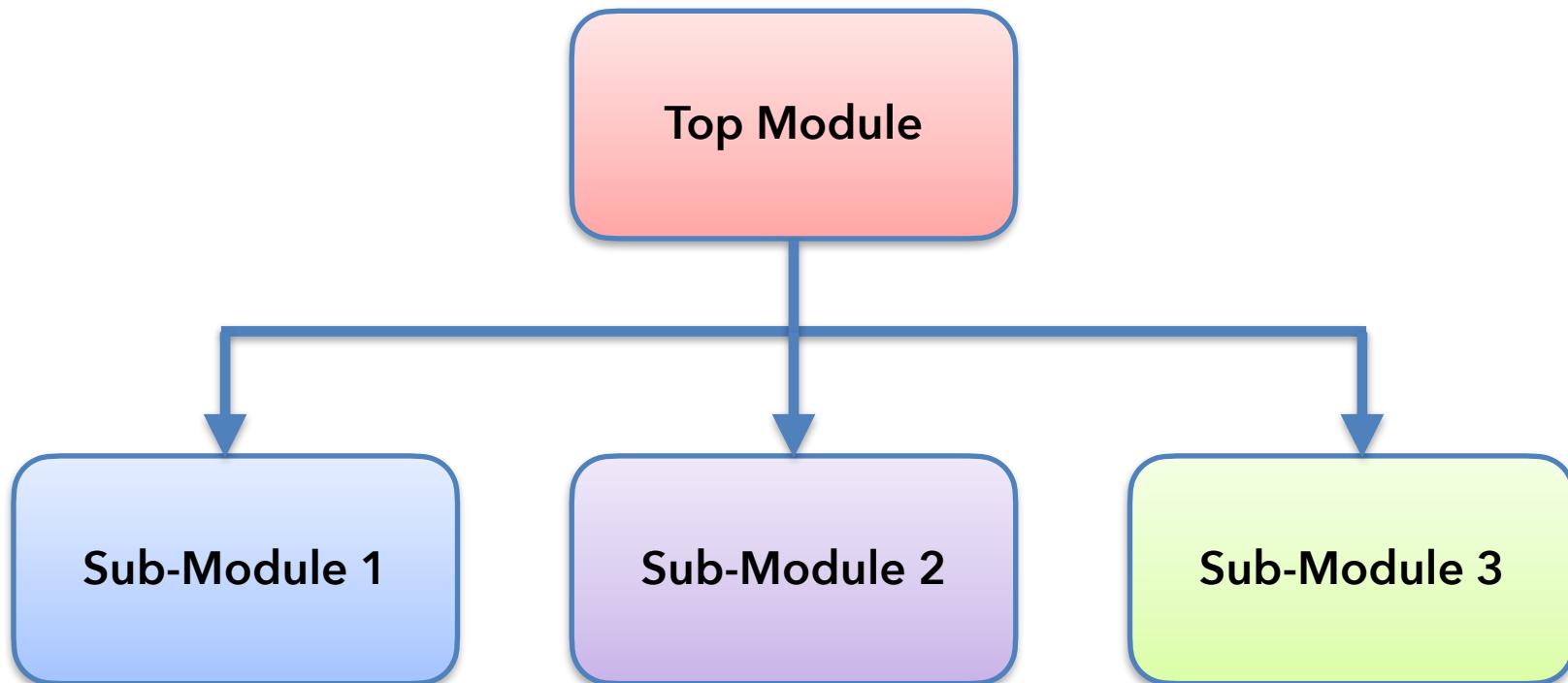


```
module MUX2bit (f1, f0, a1, a0,  
b1, b0, sel);  
  
input a1, a0, b1, b0, sel;  
output f1, f0;  
  
MUX bit1 (f1, a1, b1, sel);  
MUX bit0 (f0, a0, b0, sel);  
  
endmodule
```

Instantiate
two MUX
modules,
name them,
and specify
connections.

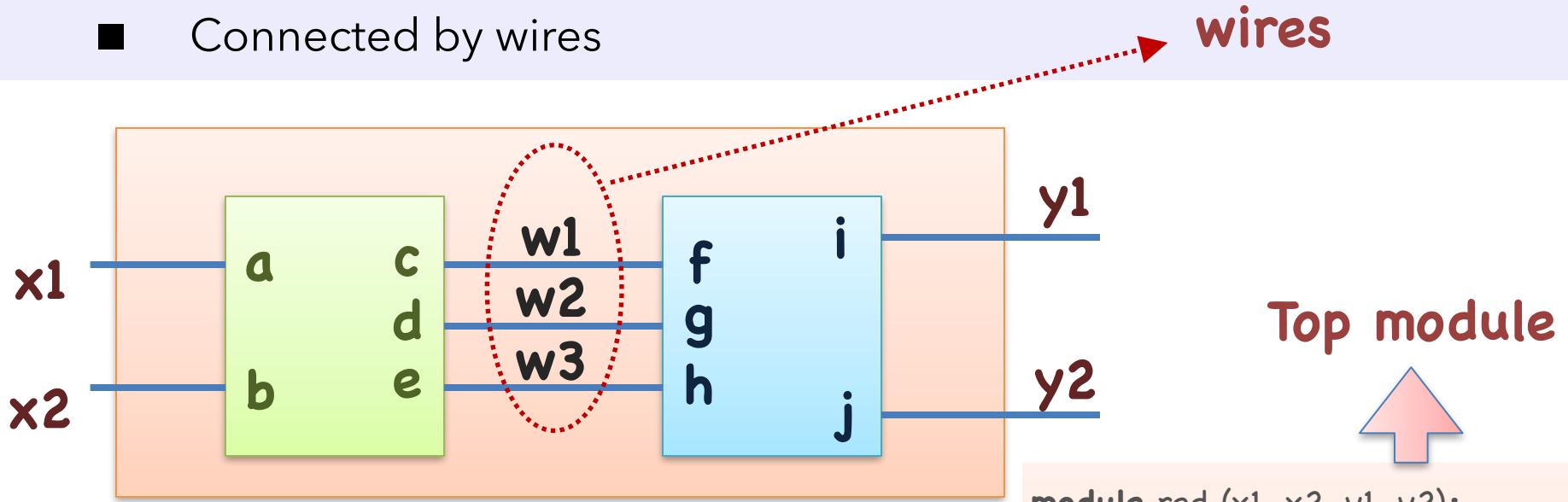
Hierarchy of Modules (1/2)

- A typical Verilog design consists of multiple hierarchies of modules
- A module may be implemented by several sub-modules
- The outermost module is called the “**top module**”



Hierarchy of Modules (2/2)

- Module reuse, time to market
- Connected by wires



```
module green (a, b, c, d, e);  
    input a, b;  
    output c, d, e;  
    .....  
endmodule
```

```
module blue (f, g, h, i, j);  
    input f, g, h;  
    output i,j;  
    .....  
endmodule
```

```
module red (x1, x2, y1, y2);  
    input x1, x2;  
    output y1, y2;  
    wire w1, w2, w3;  
    green U1 (x1, x2, w1, w2, w3);  
    blue U2 (w1, w2, w3, y1, y2);  
endmodule
```

Port Mapping

- Port mapping can be done in either of the following two ways

Connect by ordered lists

```
module red (x1, x2, y1, y2);  
    input x1, x2;  
    output y1, y2;  
    wire w1, w2, w3;  
    green U1 (x1, x2, w1, w2, w3);  
    blue U2 (w1, w2, w3, y1, y2);  
endmodule
```

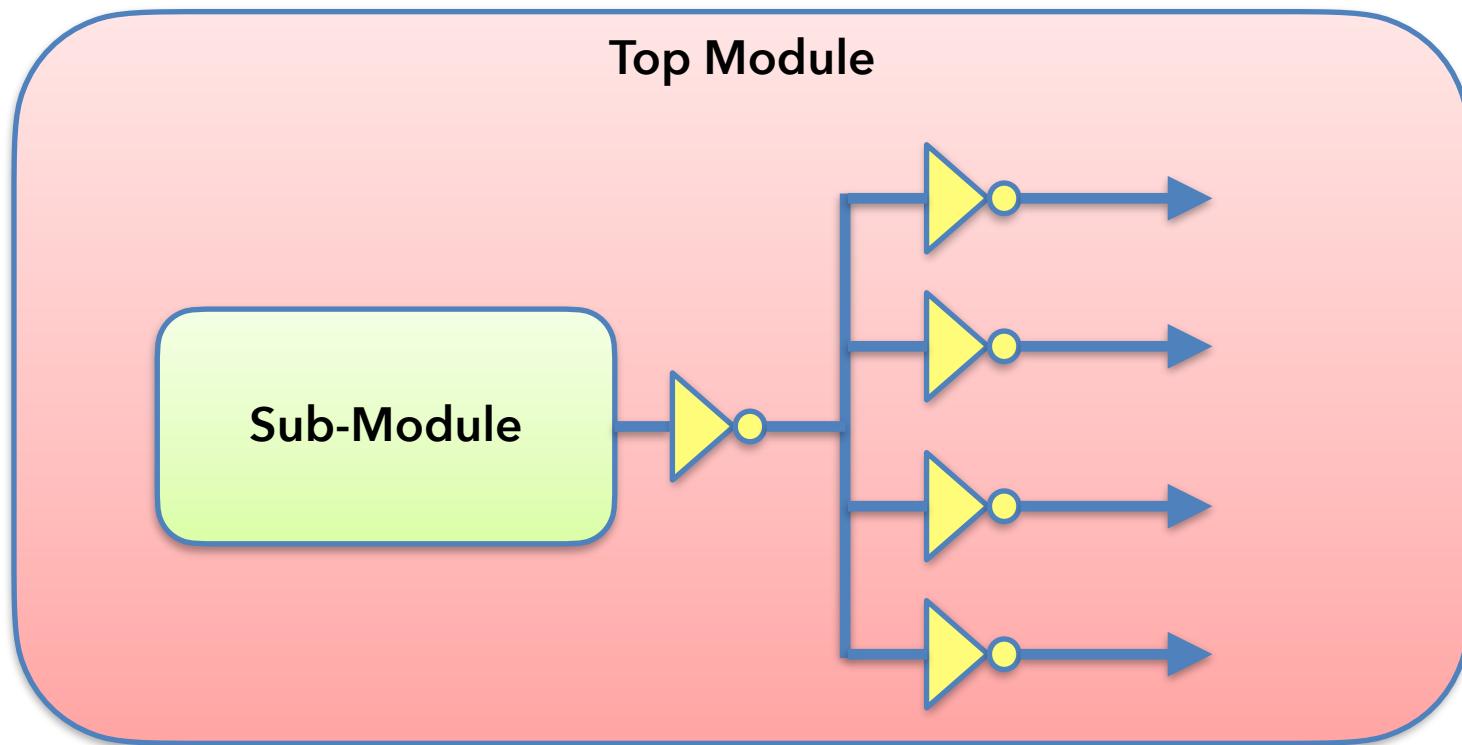
Connect by port names

```
module red (x1, x2, y1, y2);  
    input x1, x2;  
    output y1, y2;  
    wire w1, w2, w3;  
    green U1 (.b(x2), .c(w1), .a(x1), .d(w2), .e(w3));  
    blue U2 (.i(y1), .j(y2), .f(w1), .g(w2), .h(w3));  
endmodule
```

Port order can be random!

Fan-outs

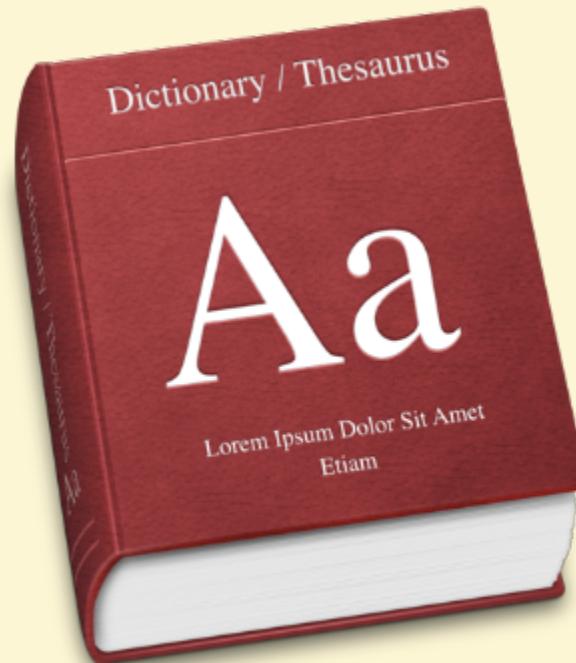
- Fan-outs means the number of output wires that a signal drives
- Fan-outs may cause additional delay and power consumption
- The fan-out gates can be **any types of logic gates**



An example of $\text{fan-out} == 4$

Agenda

- **Verilog Introduction**
 - Modules
 - Lexical conventions
- FPGA Implementation



Lexical Conventions

- Verilog lexical conventions
 - White space and comments
 - Representation of numbers
 - Continuous assignment and conditional operator

White Space and Comments

- White space for readability
 - Black space, tabs, and carriage return
- Comments
 - Understandability
 - Reusability

```
/*  
    Yes, you know, this is a BLOCK COMMENT
```

```
*/
```

```
// Write comments for other people to understand
```

```
/* Your manager will be mad if there is no comments in your codes */
```

Representation of Numbers

- Numbers are represented **UNSIGNED** by default
- Syntax:
 - **<# of bits>'<Radix><Number>**

of bits The number of BINARY bits that the number is comprised of. (default 32 bits)

Radix Radix of the number

b or B:	Binary	E.g., 8'b1011_0011
d or D:	Decimal	E.g., 4'd5
o or O:	Octal	E.g., 3'o5
h or H:	Hexadecimal	E.g., 5'h1A

Number Any legal number in selected (**# of bits**) & (**Radix**)

Continuous Assignment

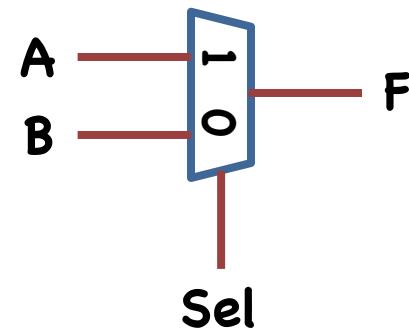
- Syntax:
 - **assign** [the name of **lvalue**] = **rvalue**;
- Assignment can be done for a single wire or a bus
 - **E.g.,** **assign** a[7:0] = b[7:0];
- **lvalue** must be of type **wire** or **output**
- **rvalue** can be an **expression**
 - **E.g.,** **assign** a[7:0] = b[7:0] & c[7:0]; (bitwise AND)
- Cascade is allowed
 - **E.g.,** **assign** {c_out, sum[3:0]} = a[3:0] + b[3:0] + c_in;

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
```

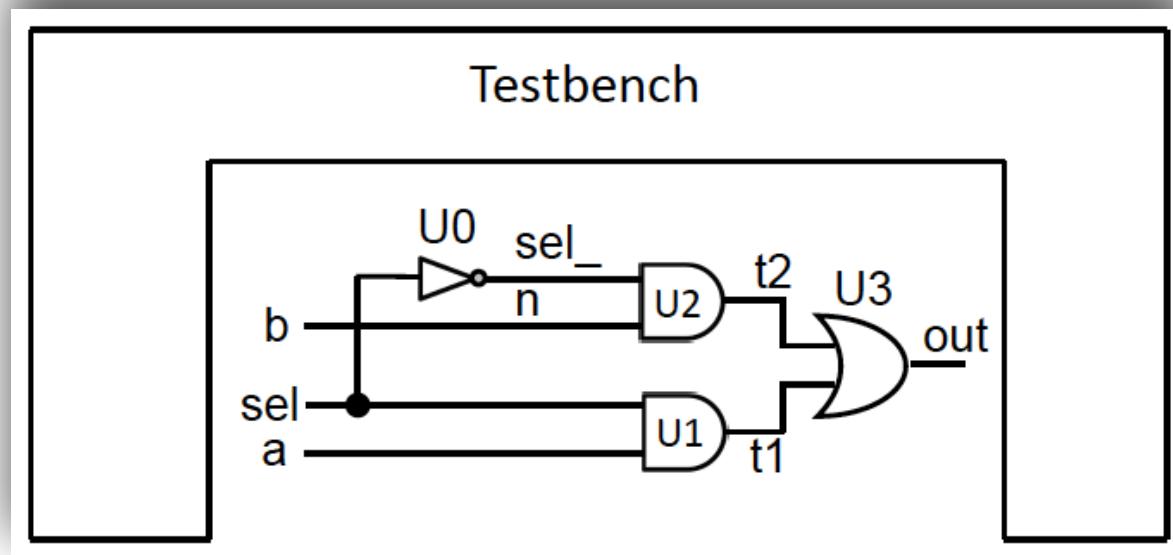
Verilog



Schematic

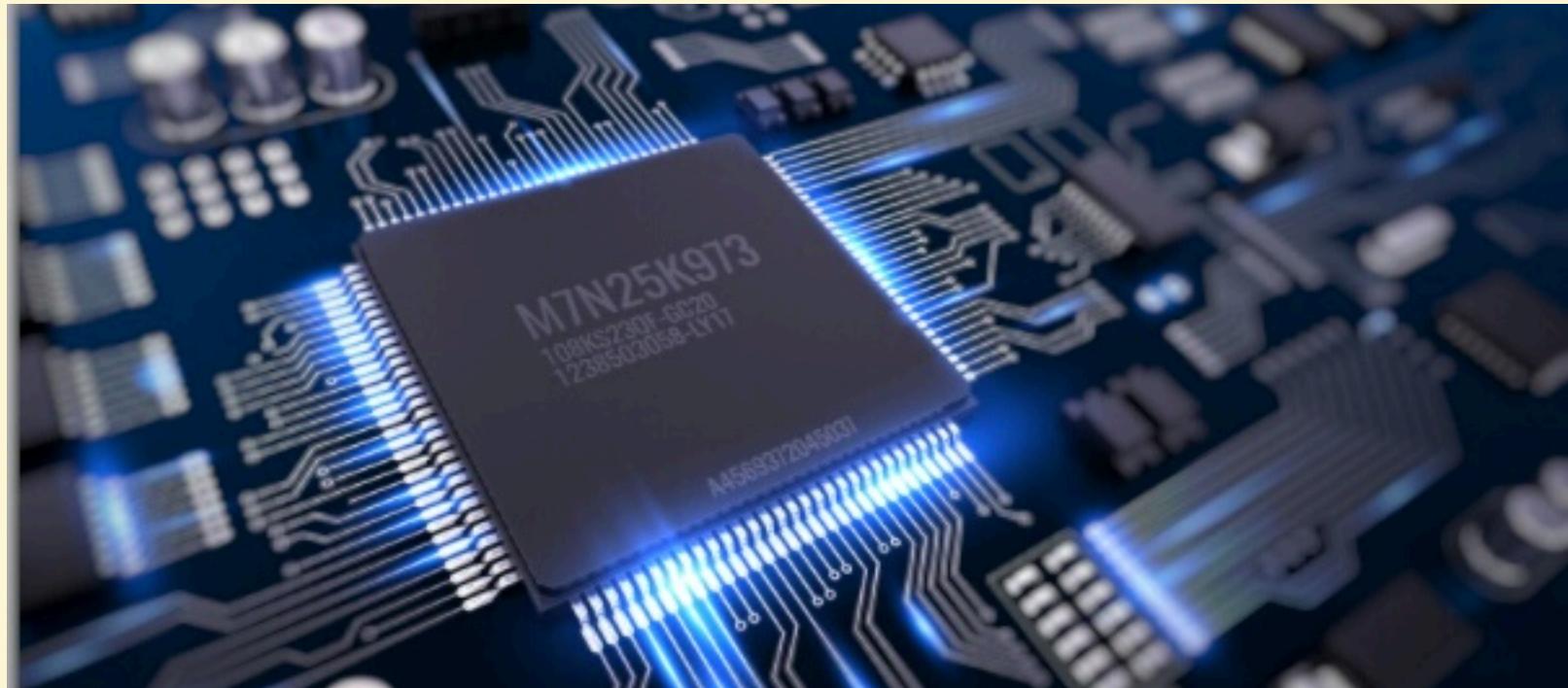
Verilog Simulation Framework

- Testbench verifies whether a module is correct or not
- Similar to the main function in C++
- Generate stimulus and check the outputs



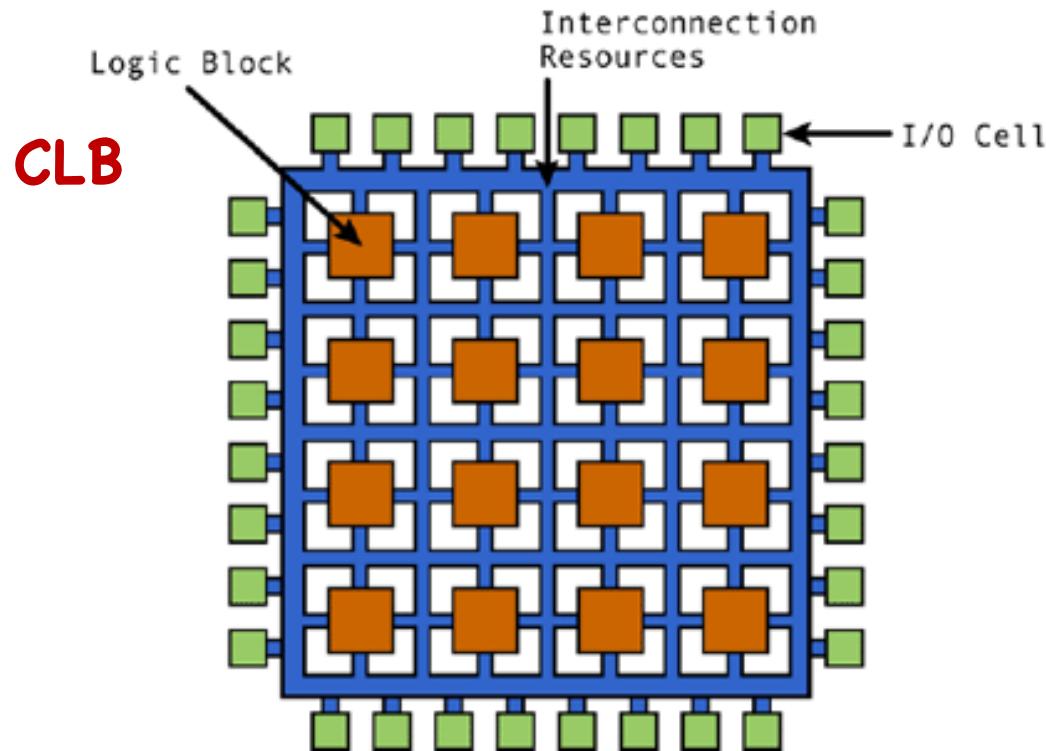
Agenda

- Verilog Introduction
- FPGA Implementation



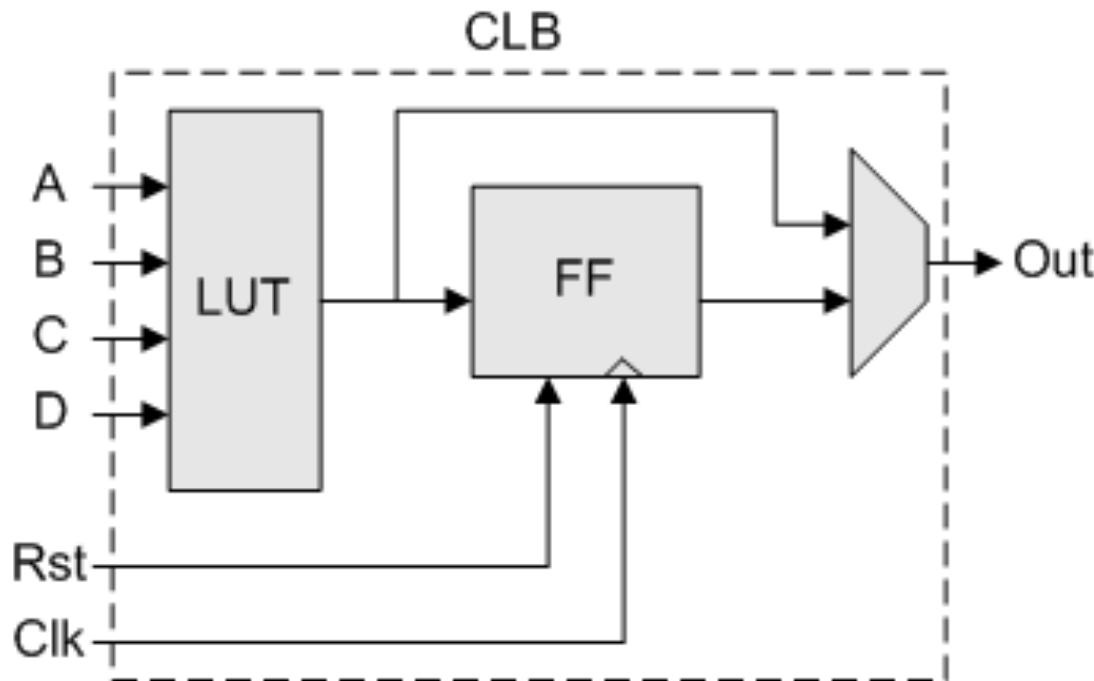
FPGA Structure

- FPGA consists of many configurable logic blocks (CLBs)
- Connected by configurable interconnection switches



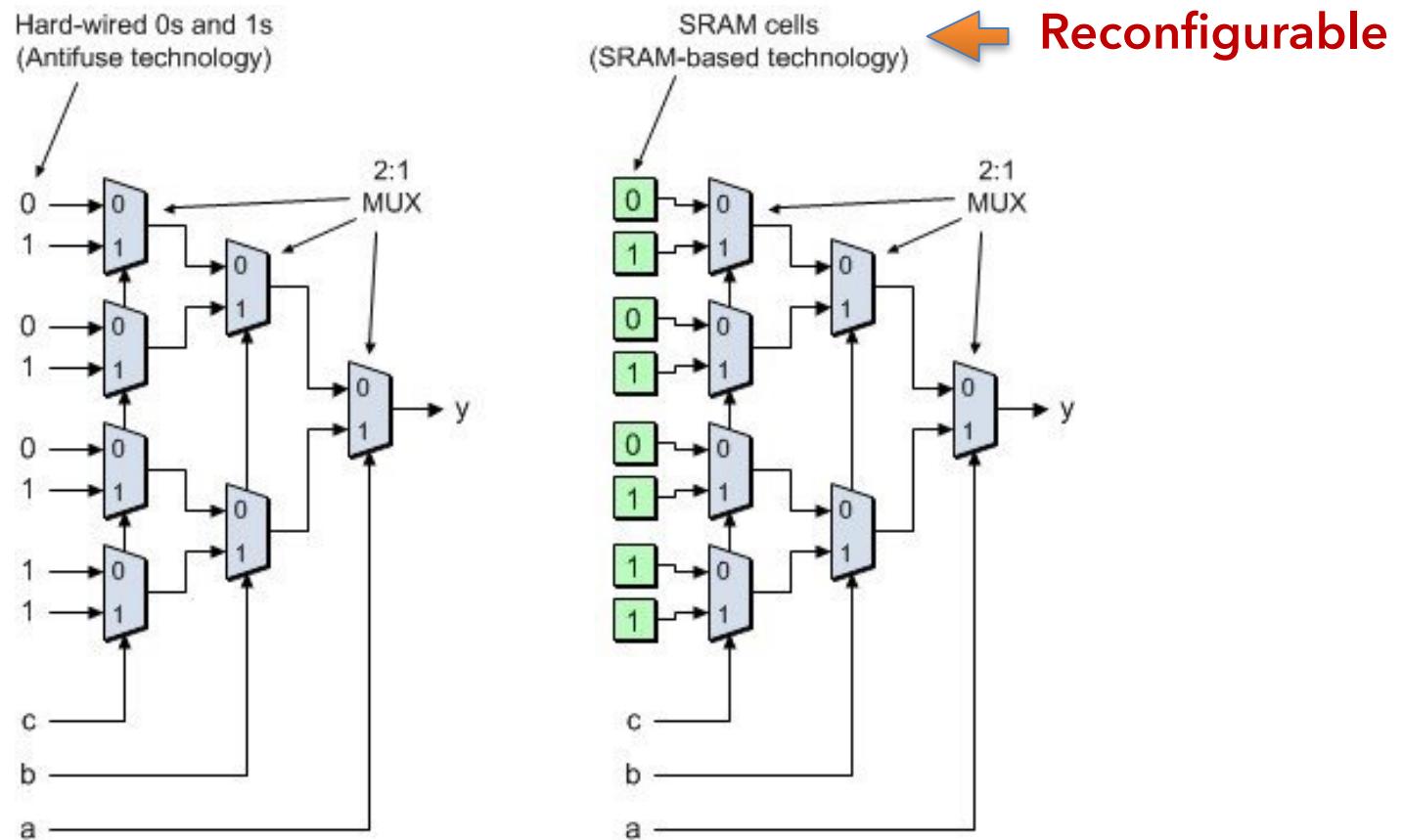
CLB Structure

- A basic CLB consists of the following components
 - Look-up tables (LUTs) (can be more than one)
 - A flip-flop (FF)
 - MUXes



LUT Structure

- Implements a **truth table** in FPGA
- The truth table is reconfigurable



Components

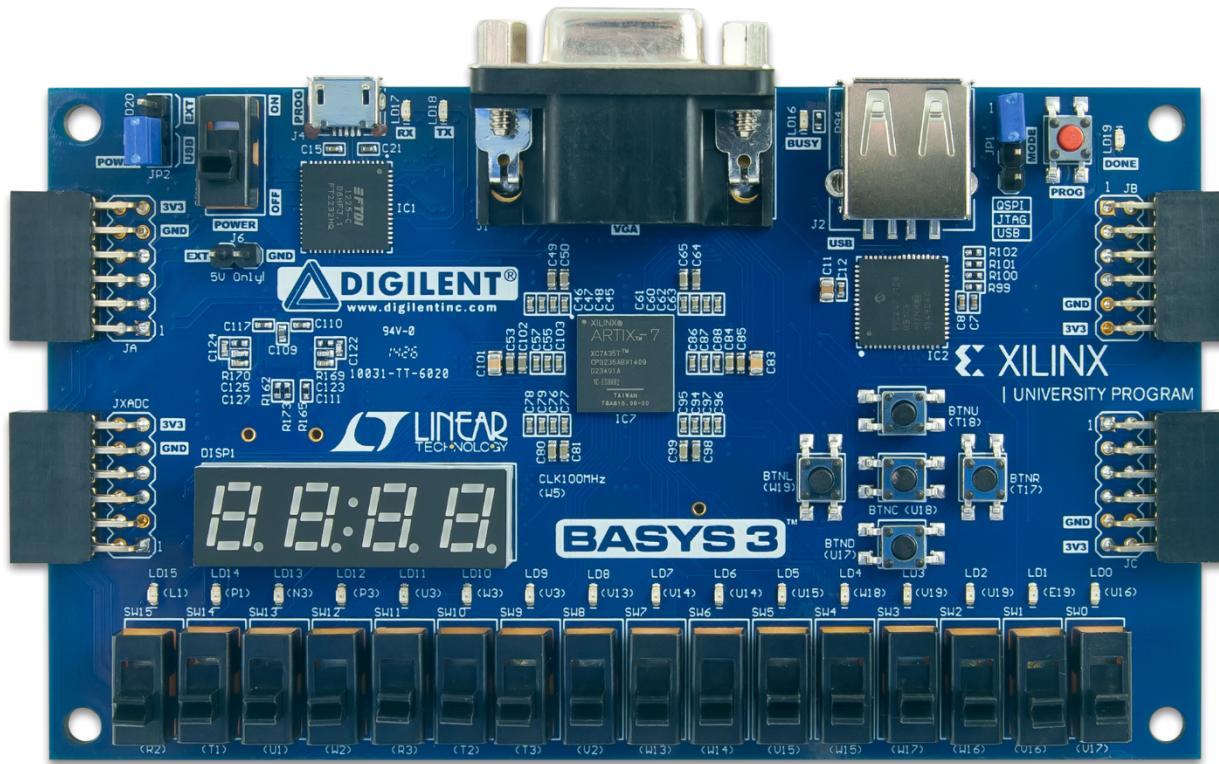
■ Input components

- Maps to input port in Verilog
- Switches, push bottoms, USB, etc.

■ Output components

- LEDs, 7-segment displays, USB, VGA port, etc.

Basys3



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

FPGA Pins

- A number of components are provided on FPGA
 - Switches, LEDs, 7-Segment Displays, etc.
 - Each component has a pin name with it

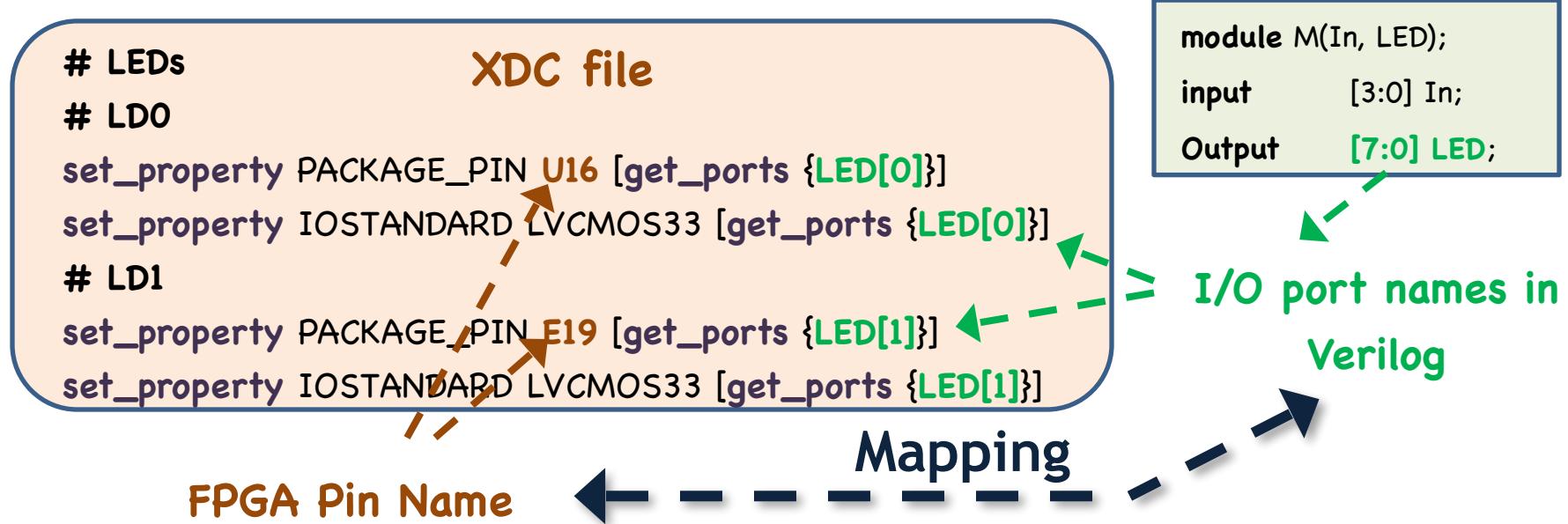


LEDs	LD15	LD14	LD13	LD12	LD11	LD10	LD9	LD8
Pin Name	L1	P1	N3	P3	U3	W3	V3	V13
LEDs	LD7	LD6	LD5	LD4	LD3	LD2	LD1	LD0
Pin Name	V14	U14	U15	W18	V19	U19	E19	U16

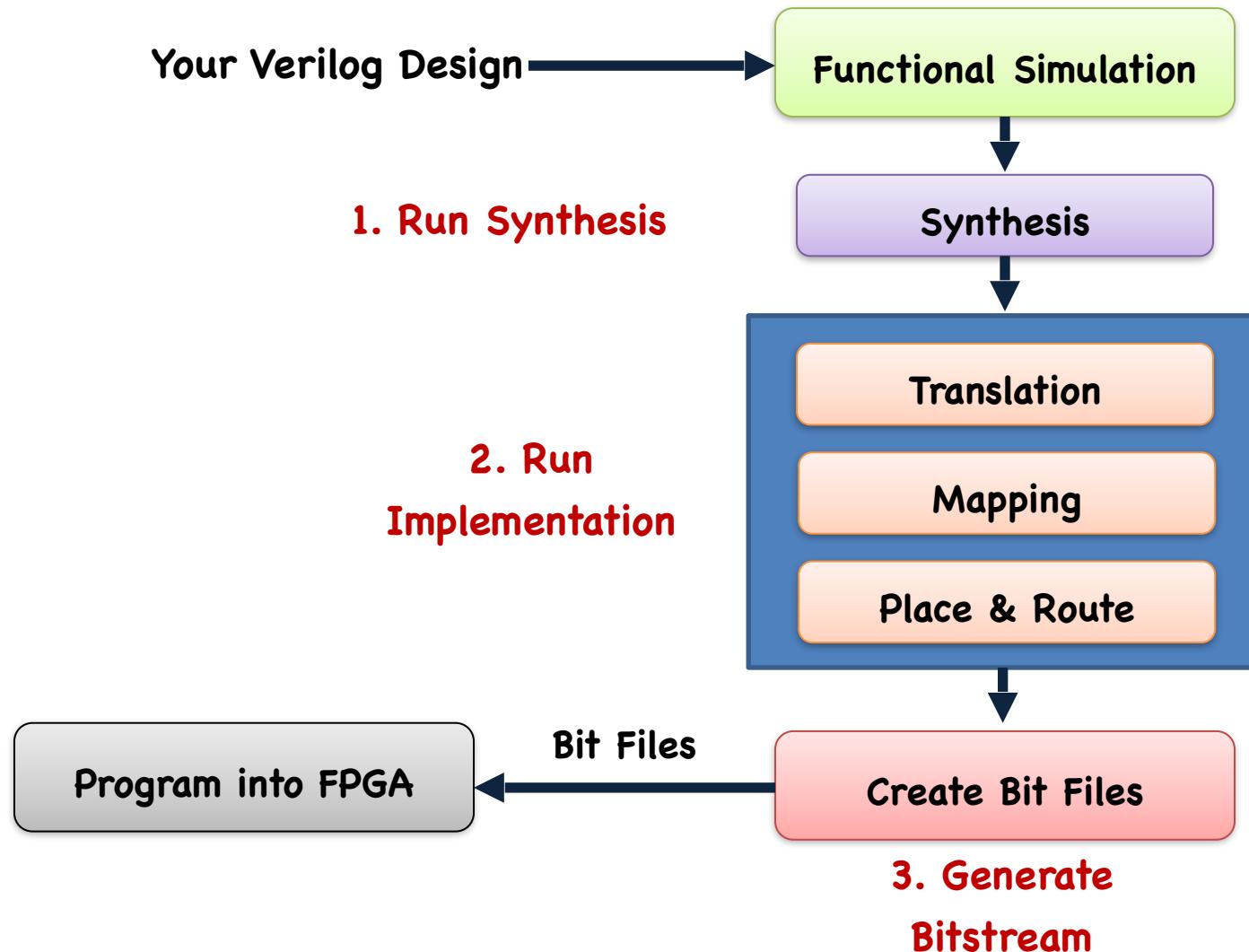
*Pictures cited from www.xilinx.com for education purpose only

Port Mapping

- Maps your inputs and outputs to FPGA pins
 - One-to-one mapping
- Defined in an **XDC** (Xilinx Design Constraints) file



FPGA Design Flow



FPGA Implementation Files

- In summary, the following files are required when configuring your Basys3 FPGA
 - Your Verilog file(s)
 - Your XDC file



Thank you for your attention!

*Seattle night view taken at Kerry Park, Seattle, WA.
This picture is taken by Chun-Yi Lee himself, who is also a fan of photography