

CSED311 Computer Architecture

# Introduction to Verilog

**Jiwoong Shin**

Original slides created by **Gwangmu Lee** and translated by **Sanghwan Jang**



**POSTECH**  
POHANG UNIVERSITY OF SCIENCE AND TECHNOLOGY

# HDL & Verilog

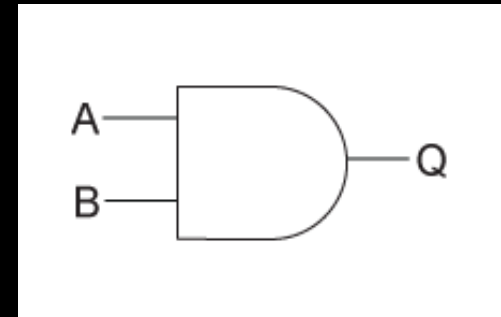
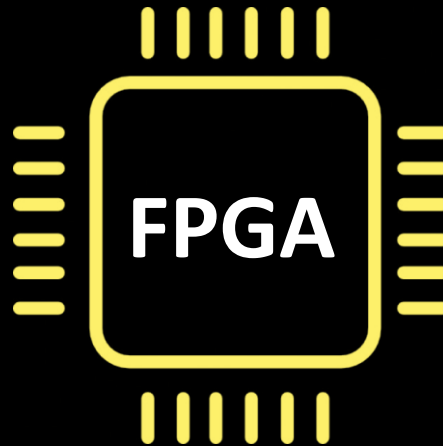


# » HDL: Hardware Description Language

Code written in HDL

```
input A, B;  
output Q;  
assign Q = A & B;
```

Synthesis



Programming language for programmable chips



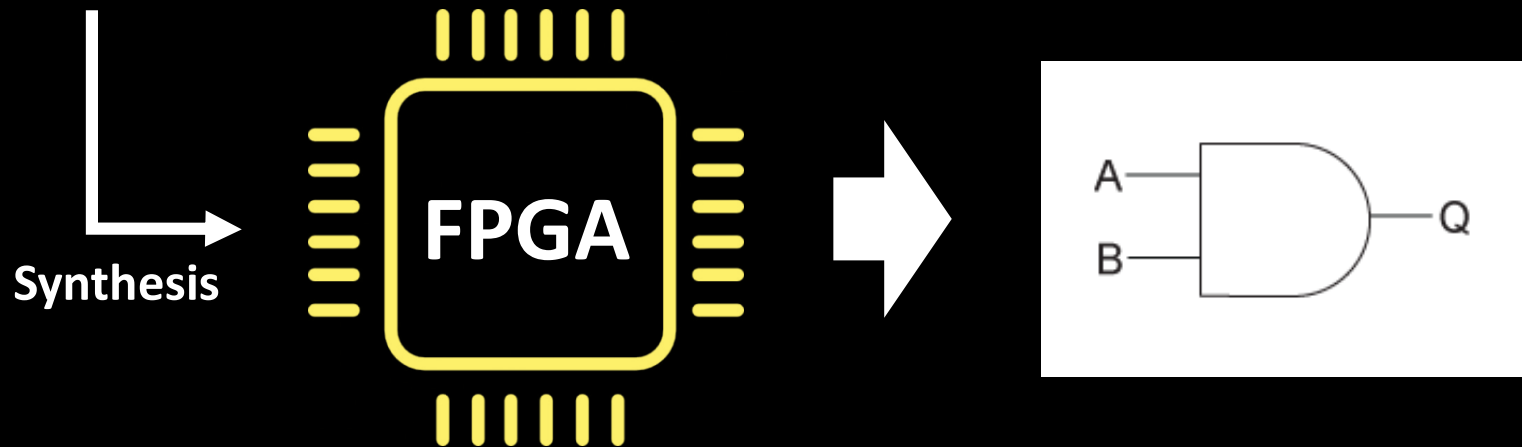
# » HDL: Hardware Description Language

Code written in HDL

```
input A, B;  
output Q;  
assign Q = A & B;
```

Examples of HDL: Verilog, VHDL, ...

*We will use **Verilog**!*



Programming language for programmable chips



# » ModelSim

- Tool for Verilog programming
- How to install and use?
  - Refer to Lab0\_ModelSim.pdf file! (LMS)



# Verilog Example



```
`timescale 1ns / 100ps
```

```
module AND (  
    input [3:0] A,  
    input [3:0] B,  
    output [3:0] Q);
```

```
    reg [3:0] Q;
```

```
    always begin  
        Q = A & B;
```

```
    end  
endmodule
```

*Let's start with this!*  
(bitwise AND module)



```
`timescale 1ns / 100ps
```

```
module AND (  
    input [3:0] A,  
    input [3:0] B,  
    output [3:0] Q);
```

```
    reg [3:0] Q;
```

```
    always begin  
        Q = A & B;
```

```
    end  
endmodule
```

← 1ns: Unit of Delay  
100ps: Unit of Simulation





```
`timescale 1ns / 100ps
```

```
module AND (  
    input [3:0] A,  
    input [3:0] B,  
    output [3:0] Q);  
  
    reg [3:0] Q;  
  
    always begin  
        Q = A & B;  
    end  
endmodule
```

← Start of Module

- Module

- Chip or Component of Chip
- Interface + Behavior
  - Interface: Input & Output
  - Behavior: Function
- Similar to OOP Class



```
`timescale 1ns / 100ps
```

```
module AND (
```

```
    input [3:0] A,  
    input [3:0] B,  
    output [3:0] Q);
```

```
    reg [3:0] Q;
```

```
    always begin  
        Q = A & B;
```

```
    end  
endmodule
```

← Declaration of Input & Output

- Input & Output (called Ports)
- [3:0]: Declaration of Bit Range
  - 4-bits (3 to 0)
  - Why not [0:3]? Explained later...

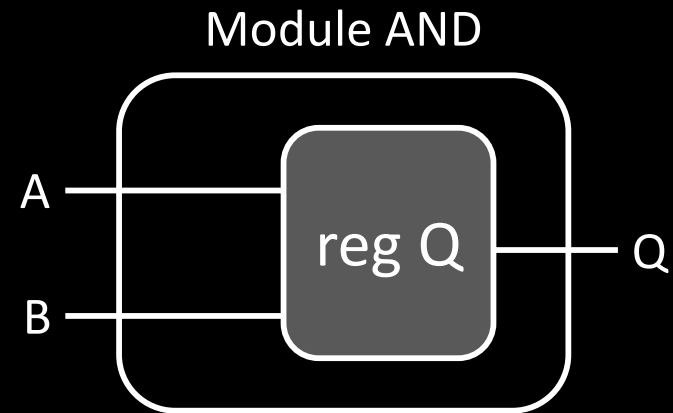


```
`timescale 1ns / 100ps
```

```
module AND (  
    input [3:0] A,  
    input [3:0] B,  
    output [3:0] Q);
```

```
    reg [3:0] Q;
```

```
    always begin  
        Q = A & B;  
    end  
endmodule
```



← Declare Output Q as Register



```
`timescale 1ns / 100ps
```

```
module AND (  
    input [3:0] A,  
    input [3:0] B,  
    output [3:0] Q);
```

```
    reg [3:0] Q;
```

```
    always begin
```

```
        Q = A & B;
```

```
    end
```

```
endmodule
```

← Always Executed  
if any of the Inputs are changed



```
`timescale 1ns / 100ps
```

```
module AND (  
    input [3:0] A,  
    input [3:0] B,  
    output [3:0] Q);
```

```
    reg [3:0] Q;
```

```
    always begin
```

```
        Q = A & B;
```

```
    end
```

```
endmodule
```

← Assign A & B to Register Q



```
`timescale 1ns / 100ps
```

```
module AND (  
    input [3:0] A,  
    input [3:0] B,  
    output [3:0] Q);
```

```
    reg [3:0] Q;
```

```
    always begin  
        Q = A & B;  
    end
```

```
endmodule
```

← End of Module



# Verilog Syntax



# » Timescale

```
`timescale 1ns / 100ps
```

- **Declare time units** needed for source code simulation

- Typically inserted at the very beginning of the code.

```
`timescale <Unit of Delay> / <Unit of Simulation>
```

- **Unit of Delay**

- Ex) #100;                      `// delay for 100 * 1ns`

- **Unit of Simulation**

- Time resolution of simulation





# » Module

- Corresponds to a chip or a component of a chip
- Interface + Design Behavior
  - Interface: Input and Output. Passage through which values pass. (Port)
  - Design Behavior:
    - What output will be given for the input?
    - How will the internal state change?
- Similar to Class in Object-Oriented Programming

```
module AND(input A, input B, output Q);  
    assign Q = A & B;  
endmodule
```

*Module Port*

*Module Body*

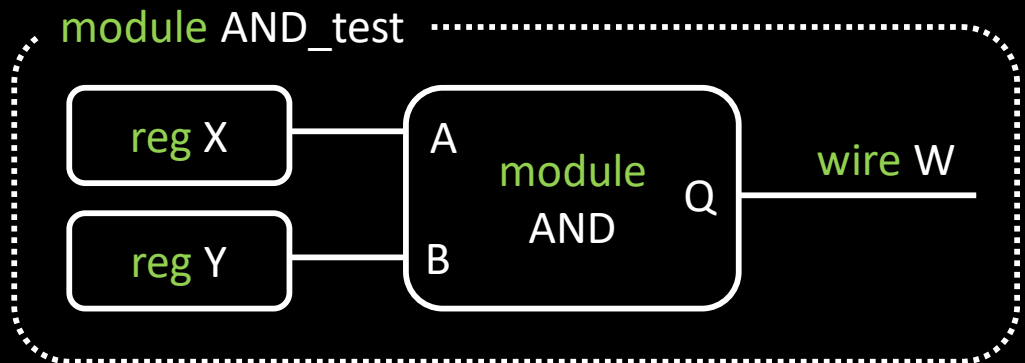


# » Module

```
module AND (input A, input B, output Q);  
    assign Q = A & B;  
endmodule
```

```
module AND_test ();  
    reg X, Y; wire W;  
    AND aaa (X, Y, W);  
endmodule
```

*Instantiation*



# » Port

- Collectively refers to the Input and Output of the Module.
- Basically declared as wire(Wire).
  - Optionally can be declared as reg(Register).

```
module always_one (output Q);  
  reg Q; Declare Output Q as reg  
  initial Q = 1;  
endmodule
```

- Only wire-wire and reg-wire connections are possible
  - Reg-reg connection is impossible.



# » Data Type

- *wire* (Wire)
  - Wire, Define physical connections
- *reg* (Register)
  - 1-bit D flip-flop. (1-bit Register)
  - Can assign values during simulation.
- Can be represented as an Bit Vector
  - ex) `wire [0:31] addr;       // Big-endian`  
          `reg [31:0] eax;        // Little-endian (Recommended)`



# » Data Type

- Value Reference

- ex) addr            // all bits of addr  
      addr[0]        // 0<sup>th</sup> bit of addr  
      addr[3:0]     // 3~0<sup>th</sup> bits of addr

- Value Concatenation

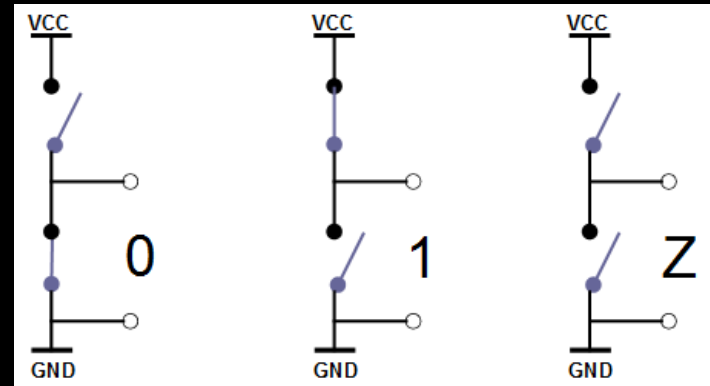
- ex) { eax[0], eax[31:1] }  
      // 0<sup>th</sup> bit of eax + 31~1<sup>st</sup> bit of eax



# » Bit-vector is the only data type in Verilog

- A bit can take on one of four values

Value	Meaning
0	Logic Zero
1	Logic One
X	Unknown Logic Value
Z	High Impedance, Floating



- An X bit might be a 0, 1, Z, or in transition. We can set bits to be X in situations where we don't care what the value is. This can help catch bugs and improve synthesis quality.

# Verilog 문법: Module Body

Verilog Syntax: Module Body



# » Structured Procedure

```
module AND (input A, input B, output Q);
```

```
  reg Q;  wire AnB;
```

```
  assign AnB = A & B;
```

```
    always begin
```

```
      Q = AnB;
```

```
    end
```

```
endmodule
```

*Structured Procedure*

- A type of code block.
  - `always`: Always running block without condition
  - `Initial`: Block that executes only once during module instantiation





# » Initial Procedure

- Runs only once during module instantiation.
- Can contain multiple statements using begin and end.

```
initial begin  
    clk = 0;  
    x = 1;  
end
```

=

```
initial clk = 0;  
initial x = 1;
```



# » Always Procedure

- (When certain conditions are met) always called.
- If there is no condition, called infinitely.

```
reg clk;  
initial clk = 0;  
always begin  
    #50;  
    clk = ~clk;  
end
```

clk value is assigned as 0  
→ clk value changed

Always block is called  
→ After a delay of 50 (Important!)  
→ clk value reversed



# »» Other Procedure

- Function, Task
  - Similar to function in C
  - Used to define a sub-module within a module.
  - (You don't need to use it,  
but if you are interested, study it yourself!)



# Verilog 문법: Assignments

Verilog Syntax: Assignments



# » Assignments

Revisiting AND example..

```
module AND (input A, input B, output Q);
```

```
  reg Q;  wire AnB;
```

```
  assign AnB = A & B;
```

*Continuous Assignments*

```
  always begin
```

```
    Q = AnB;
```

*Procedural Assignments*

```
  end
```

```
endmodule
```

- Used to assign wire connection or value.
  - Continuous: Define wire connection
  - Procedural: (In procedure) Store value in reg



# » Continuous Assignments

- Used to assign a wire connection to a wire
  - ex) `wire a, b, c;`  
`assign c = a & b; // wire c is defined as a & b.`
  - It is generally used outside procedure.
- Can be used in conjunction with wire declaration.
  - ex) `wire a, b;`  
`wire c = a & b;`
- Once assigned, it cannot be changed during simulation.



# » Continuous Assignments

- You can give delay to assignment itself.
  - ex) `wire a, b, c;`  
    `assign #10 c = a & b;`  
    // a & b is assigned to c after a delay of 10.
  - Can be used to model the delay in wire connections.
- Can be treated like an alias. (Remind reference type in C++)
  - ex) `wire a, b;`  
    `assign b = a;`      // b is exactly the same as a



# » Procedural Assignments

- (In procedure) Used to store value in reg.

– ex)

```
reg clk;  
initial begin  
    clk = 0;  
end
```

- Delay Modeling

– ex) #10 clk = 0;     // assign 0 to clk after a delay of 10  
      #10 clk = #20 tmp;   // load tmp after a delay of 20, and  
                          // store it to clk after a delay of 10





# » Types of Procedural Assns.

- **Blocking** Assignments

- Execute assignments according to statement order.
- Use '=' operator.
- The order between procedures(code blocks) is also valid.

```
reg a, b;  
initial begin  
    a = 0;  
    b = 1;  
end
```

```
reg a, b;  
initial begin  
    a = 0;  
end  
Initial begin  
    b = 1;  
end
```

*In both cases a = 0 is executed first,  
and then b = 1 is executed.*



# » Types of Procedural Assns.

- **Non-blocking** Assignments

- Execute assignments simultaneously without any order.
- Use ' $\leq$ ' operator.

```
reg a, b;  
initial begin  
    a  $\leq$  0;  
    b  $\leq$  1;  
end
```

```
reg a, b;  
initial begin  
    b  $\leq$  1;  
    a  $\leq$  0;  
end
```

*In both cases,  $a \leq 0$  and  $b \leq 1$  are executed simultaneously.*




# » Blocking vs. Non-blocking

\* Initially, a = 0 and b = 1

Blocking  
Assn.

```
reg a, b, c;  
initial begin  
    #50;  
    b = a;  
    c = b;  
end
```


Cycle	a	b	c
0	0	1	
50	0	0	0



Non-blocking  
Assn.

```
reg a, b, c;  
initial begin  
    #50;  
    b <= a;  
    c <= b;  
end
```

Cycle	a	b	c
0	0	1	
50	0	0	1



# »» Mixed Blocking

- Don't even think about it!
- There is no syntax problem, but the results are hard to predict.
- Use non-blocking assignment mostly.
  - It is closer to real hardware.



# Verilog 문법: Delay & Event

Verilog Syntax: Delay & Event



# » Delay

- Used when delaying module behavior during simulation.
  - ex) `#100;`      `// Wait for a delay of 100`
- Used for more realistic hardware modeling
  - Can even give a random value instead of a fixed value.
  - Not to be used to “debug” your Verilog design!!
- We limit the use of delay to two parts in Lab assignments.
  - Clock (Code to generate the signal)
  - Testbench (Code to test the module)



## » Delay

- In begin ... end, delay is cumulatively applied.

```
initial begin
```

```
    #50 clk = 0;
```

```
    #50 x = 1;
```

```
end
```

=

```
initial #50 clk = 0;
```

```
initial #100 x = 1;
```



# » Event

- Used to give execution condition of specific procedure block or procedural assignment.

*Always execute when event e occurs*

```
module dummy (input a,  
              output b);  
  reg b;  
  event e;  
  initial begin  
    #50;  
    -> e;  
  end
```

```
always @e begin  
  b <= 1;  
end  
endmodule
```

*Wait for a delay of 50,  
and raise event e.*





# » Event

- Used to give execution condition of specific procedure block or procedural assignment.

*Always execute when event e occurs*

```
module dummy (input a,  
              output b);  
  reg b;  
  event e;  
  initial begin  
    #50;  
    -> e;  
  end
```

*Wait for a delay of 50,  
and raise event e.*

```
always @e begin  
  b <= 1;  
end  
endmodule
```

If an event condition is given,  
execute only when this condition is met



# » Event

- All data(wire or reg) changes can be used as events.
  - ex) `reg a;`  
`always @(a) begin...` // execute when the value of 'reg a' changes
- Change of a value can be used as an event.
  - ex) `reg a;`  
`always @(posedge a) begin` // when 'reg a' switches from 0 to 1  
`always @(negedge a) begin` // when 'reg a' switches from 1 to 0
- Or-ing between events are allowed
  - ex) `always @(a or b) begin..`



# » Event

- Can be used within a procedure block or statement.

```
reg clk, a, b;  
initial begin  
    @(posedge clk) a <= 0;  
    @(posedge clk) b <= 1;  
end
```

Initially, wait until clk changes 0 → 1 and then execute.

Next, wait until clk changes 0 → 1 and then execute.



# Verilog 문법: Flow Control

Verilog Syntax: Flow Control



# » Flow Control

- Used only within a structured procedure.
- Syntax provided for ease of implementation.
  - It is far from real hardware implementation.
- Types
  - If Statement
  - Case Statement
  - Loop Statement



# » If Statement

- Conditionally executes statements.
- Can also be expressed with the "?" operator from C (implemented as a mux)

```
initial begin
  if (a == 5) begin
    b <= 15;
  end
  else begin
    b <= 25;
  end
end
```

=

```
wire c = (a == 5) ? 15 : 25;
initial begin
  b <= c;
end
```



# » Case Statement

- Optionally executes statements based on the value.

```
initial begin
  if (a == 5) begin
    b <= 15;
  end
  else begin
    b <= 25;
  end
end
```

=

```
initial begin
  case (a)
    5 : b <= 15;
    default : b <= 25;
  endcase
end
```



# » Loop Statement

- Repeat
  - Repeat the statement for the specified number of times.

```
initial begin
```

```
    repeat (4) c = c + 1;
```

```
end
```

=

```
initial begin
```

```
    c = c + 1;
```

```
    c = c + 1;
```

```
    c = c + 1;
```

```
    c = c + 1;
```

```
end
```





# » Loop Statement

- Forever
  - Repeat the statement indefinitely.

```
initial begin
```

```
    forever #5 c = c + 1;
```

```
end
```

=

```
initial begin
```

```
    #5 c = c + 1;
```

```
    #5 c = c + 1;
```

```
    #5 c = c + 1;
```

```
    #5 c = c + 1;
```

```
    ...
```

```
    // repeated indefinitely
```

```
end
```



# » Loop Statement

- While, For
  - Easy to use, but often impossible to implement with hardware.
  - Can be used without problems when initializing array.

```
reg c;  
initial begin  
    while (c < 10)  
        c = c + 1;  
end
```

```
reg c;  
initial begin  
    for (c = 0; c < 10; c = c + 1)  
        // Statement  
end
```



# Verilog 문법: 기타

Verilog Syntax: Etc.



# » Other General Syntax

- Comment
  - // or /\* \*/ (similar to C)
- Constant
  - Format: <Number of Bits>'<Base><Value>
    - <Base> → h(Base 16), d(Base 10), o(Base 8), b(Base 2)
    - <Value> → 0~9, A~F,  
X(Unknown Value), Z(High Impedance)
  - ex) 8'b01001111 (8-bit Binary 01001111)  
16'hAF (16-bit Hexadecimal AF)
  - Decimal numbers can be entered without any special format.



# » Other General Syntax

- Operators

- Basically similar to C operators (But no ++ & --)
- Unary/Binary Logical Operators  
and(&), nand(~&), or(|), nor(~|), xor(^),  
xnor(^~ or ~^), negation(~)
- Equal Operators  
a == b, a != b : return X if a or b contains X or Z  
a === b, a !== b : even compares X and Z



# »» Reference

- Verilog References

- Verilog Tutorial & Examples

- <http://www.asic-world.com/verilog/index.html>

- Verilog Syntax: Compact Summary

- <https://www.csee.umbc.edu/portal/help/VHDL/verilog/summary.html>





>> Thank You!



**POSTECH**  
POHANG UNIVERSITY OF SCIENCE AND TECHNOLOGY