Verilog Tutorial*

* This course assumes experience with Verilog coding as prerequisite from Logic Design (논리설계). This deck of slides is provided for review purposes only. For more information please refer to other materials.

Prof. Jae W. Lee (jaewlee@snu.ac.kr)

Department of Computer Science and Engineering

Seoul National University

TA (snu-arc-uarch-ta@googlegroups.com)

Caution

- This slides has quoted a portion of the logic design lecture
- Please refer to the logic design materials for detailed explanations
- The following content has been written only to remind the basics of Verilog

Contents

Hardware Description Language (HDL)

- **HDL is an language to describe hardware**
  - Describe hardware design at varying levels of abstraction
  - Can be used for hardware simulation
- **Examples of HDL**
  - Abel (circa 1983) - developed by Data-I/O
  - ISP (circa 1977) - research project at CMU
  - Verilog (circa 1985) - developed by Gateway (absorbed by Cadence)
  - VHDL (circa 1987) - DoD sponsored standard
- **This tutorial covers basics of Verilog HDL**

Verilog Notations

- **Verilog is:**
  - Case sensitive
    - e.g "Wire" is not equal to "wire"
  - Based on the programming language C
- **How to use Comments?**
  - // this is a comment Single Line
  - /* this is a comment */ Multiple line
- **List element separator: ,**
- **Statement terminator: ;**

Verilog Notations

- **Binary Values for Constants and Variables**
  - 0 - logic 0 or a false condition
  - 1 - logic 1 or a true condition
  - X, x- unknown logic value
  - Z, z– High impedance, floating state (open circuit)

size in bits

Value

- **Constants**
  - n'b[integer]: 1'b1 = 1, 8'b1 = 000000001, 4'b0101 = 0101, 8'bxxxx = 0000xxxx

Base format: binary format

  - n'h[integer]: 8'hA9 = 10101001, 16'hf1= 0000000011110001

Hexadecimal format

- **Identifier Examples**
  - Scalar: A, C, RUN, stop, m, n
  - Vector: sel[2:0], f[5:0], ACC[31:0], SUM[15:0], sum[15:0]

Verilog Operators

- **Bitwise Operators**

~a	**NOT** a
a & b	a **AND** b
a | b	a **OR** b
a ^ b	a **XOR** b

- **Bitwise Operators Example**
  - input [3:0] A, B;
  - output [3:0] C;
  - assign C = A | ~B;
  - Assume A is 4b'0001 and B is 4b'1010

    then C became 0001 | ~ (1010) = 0001 | 0101 = 0101

Verilog Operators

- **Logical Operators**
  - !, &&, ||, etc

!a	**NOT** a
a && b	a **AND** b
a || b	a **OR** b

- **Logical Operators Example**
  - wire [3:0] A, B;
  - wire C;
  - assign C = (A || !B);
  - Assume A is 4b'0001 and B is 4b'1010

    then C became 0001 || 0 = 1

Verilog Operators

- **Relational Operators**
  - == , != , >= , <= , etc
  - a == b   if a = b : 1, else : 0
  - a > b   if a > b : 1, else : 0
  - etc..
- **Relational Operators Example**
  - wire [3:0] A, B, C;
  - assign C = (A > B) ? 3’b111 : 3’b100;
  - Assume A is 4b’0001 and B is 4b’1010

    then C became 3b’100 (because A > B is false)

Verilog Operators

- **Arithmetic Operators**
  - + , - , * , / , %, etc
  - a + b   returns a + b
  - a - b   returns a - b
  - etc..
- **Arithmetic Operators Example**
  - wire [3:0] A, B, C;
  - assign C = A + B;
  - Assume A is 4b’0001 and B is 4b’1010

    then C became 4b’1011

Verilog Operators

- **Concatenation Operators**
  - {a, b, c, …} : concatenates members
- **Concatenation Operators Example**
  - wire [3:0] A, B;
  - wire [7:0] C;
  - assign C = {A , B};
  - Assume A is 4b’0001 and B is 4b’1010

    then C became 8b’00011010

Module

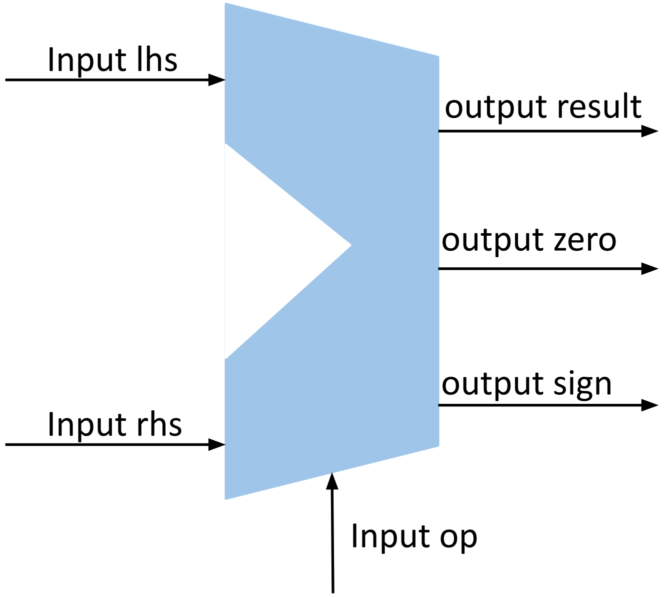
- **Module is the fundamental building block for Verilog design**
  - Used to construct design hierarchy
  - Module definition cannot be nested
  - But, can be used as an instance in other modules
  - inputs / outputs (IO Port) work as interfaces of a module
  - Contains wires, registers and submodules

Module

- **Module declaration**
  - start with **module**, end with **endmodule**
- **IO port declaration**
  - Inputs use **keyword input**, Outputs use **keyword output**
  - Scalar
    - input list of input/output identifiers
    - Example : **input** A_in, B_in;
  - Vector
    - input[range] list of input/output identifiers
    - Example : **output [15:0]** C_out;

Module (Code Example)

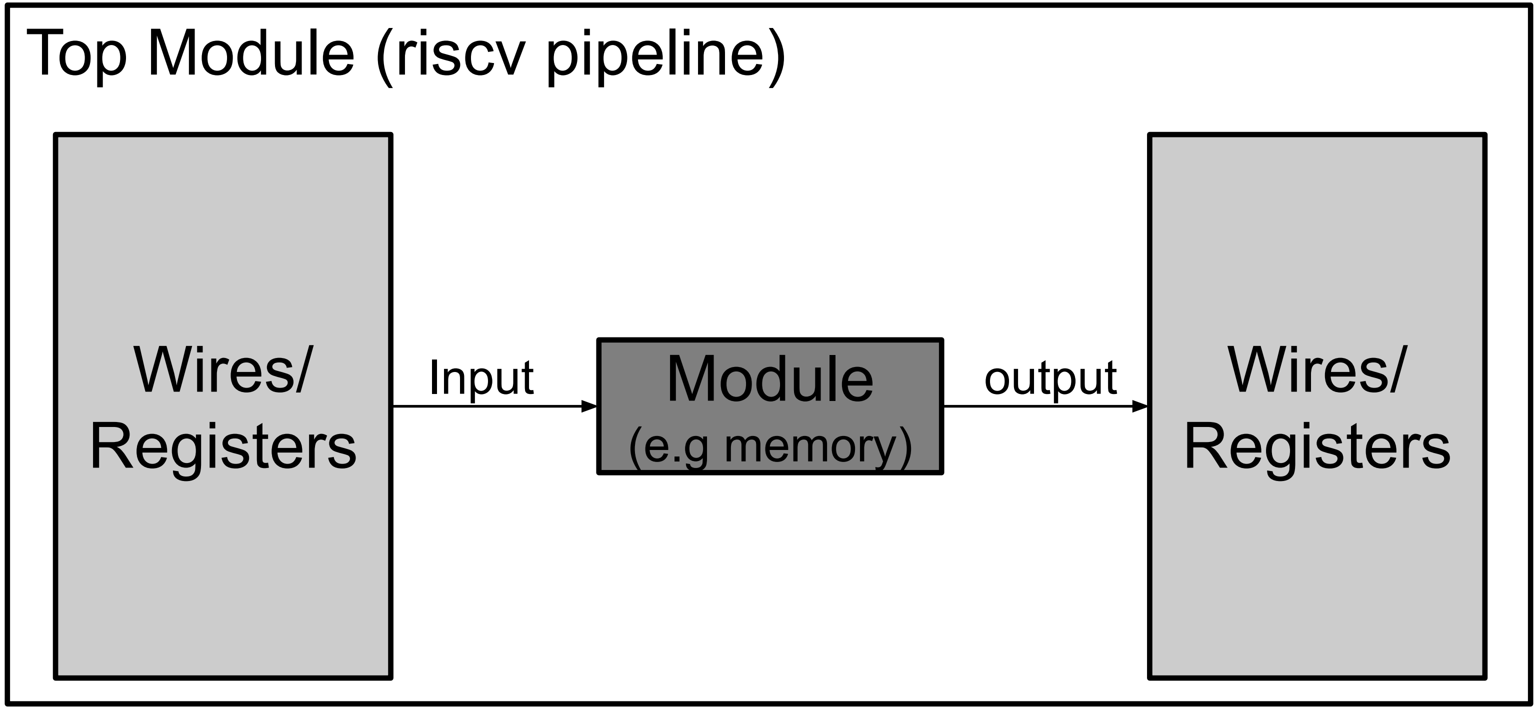
- **ALU**



```
module alu
(
        input [2:0] op,
        input [63:0] lhs,
        input [63:0] rhs,
        output [63:0] result,

        //Assume signed input only.
        output zero,  //Set 1 if result == 0
        output sign   //Set 1 if result < 0
);

...

endmodule
```

Module

- **Example of nested module (I/O of top module omitted)**



Module (Code Example)

```
module riscv_pipeline (
     input clk,
     input reset,
 );
…
reg [31:0] IF_pc;
wire [31:0] IF_inst;
reg [31:0] ID_inst;
…
//—------IF Stage—--------
rom imem(IF_pc, IF_inst)
always @(posedge clk)
begin
     IF_pc <= reset ? 32’h0 : IF_pc + 4;
     ID_pc <= reset ? 32’h0 : IF_pc;
     ID_inst <= reset ? IF_inst;
end
…
endmodule // End of module definition
```

```
module rom64
(
     input [63:0] address,
     output [31:0] data_out
);

…
reg [31:0] data[127:0];
…
assign data_out = (address &
(~(ROM64_BITMASK)) ? 32'h0 :
data[address[8:2]];
…
endmodule
```

Module

- **Wire declaration**
  - Data propagates **instantly** through wire
  - Use with **assign** statement
  - Usage example : wire t1; wire [7:0] t2;
- **Register declaration**
  - Value changes only at **specified condition**
  - Use with **always, initial** statement
  - Can be used as temporary storage
  - Usage example : reg r1; reg [31:0] r2;

Procedural expression

- **Process**
  - The body of process consists of procedural statement to make desired outputs from inputs as like a common programming
  - Processes are running in **parallel**
  - **initial** - executes only once at t= 0
  - **always** - executes at t = 0 and repeatedly thereafter following repeat condition
  - Example

```
always Repeat condition
Statement1;
Statement2;
```

Procedural expression

- **Blocking vs non-blocking assignment**

```
module name (/* I/O Definition
*/);

    reg ra, rb, rc;
    ...

    always @(posedge clk)
    begin
        rb = ra;
        rc = rb;
    end

    ...

endmodule
```

Blocking assignment example

```
module name (/* I/O Definition
*/);

    reg ra, rb, rc;
    ...

    always @(posedge clk)
    begin
        rb <= ra;
        rc <= rb;
    end

    ...

endmodule
```

Non-blocking assignment example

Procedural expression

- **Blocking assignment**
  - Execute sequentially

```
module name (/* I/O Definition
*/);

    reg ra, rb, rc;
    ...

    always @(posedge clk)
    begin
        rb = ra;
        rc = rb;
    end

    ...

endmodule
```

```
Initial value:

    ra = 1
    rb = 2
    rc = 3

Resulting value:

    ra = 1
    rb = 1
    rc = 1
```

Procedural expression

- **Non-Blocking assignment**
  - Execute simultaneously

```
module name (/* I/O Definition
*/);

    reg ra, rb, rc;
    ...

    always @(posedge clk)
    begin
        rb <= ra;
        rc <= rb;
    end

    ...

endmodule
```

```
Initial value:

    ra = 1
    rb = 2
    rc = 3

Resulting value:

    ra = 1
    rb = 1
    rc = 2
```

Conditional expression

- **If-else statement / Case statement**
  - Only available in always statement

```
...
wire wa, wb, wc;
reg ra, rb, rc;

always @(posedge clk)
begin
     ...
     if(rb == 1’b1) begin
          ra <= wa;
     end
     else begin
          ra <= wc;
     end
end
```

Code Example

```
module riscv_pipeline (
     input clk,
     input reset,
 );
…
reg [31:0] IF_pc;
wire [31:0] IF_inst;
reg [31:0] ID_inst;
…
//—------IF Stage—--------
rom imem(IF_pc, IF_inst)
always @(posedge clk)
begin
     IF_pc <= reset ? 32’h0 : IF_pc + 4;
     ID_pc <= reset ? 32’h0 : IF_pc;
     ID_inst <= reset ? IF_inst;
end
…
endmodule // End of module definition
```