KYUNGPOOK NATIONAL UNIVERSITY

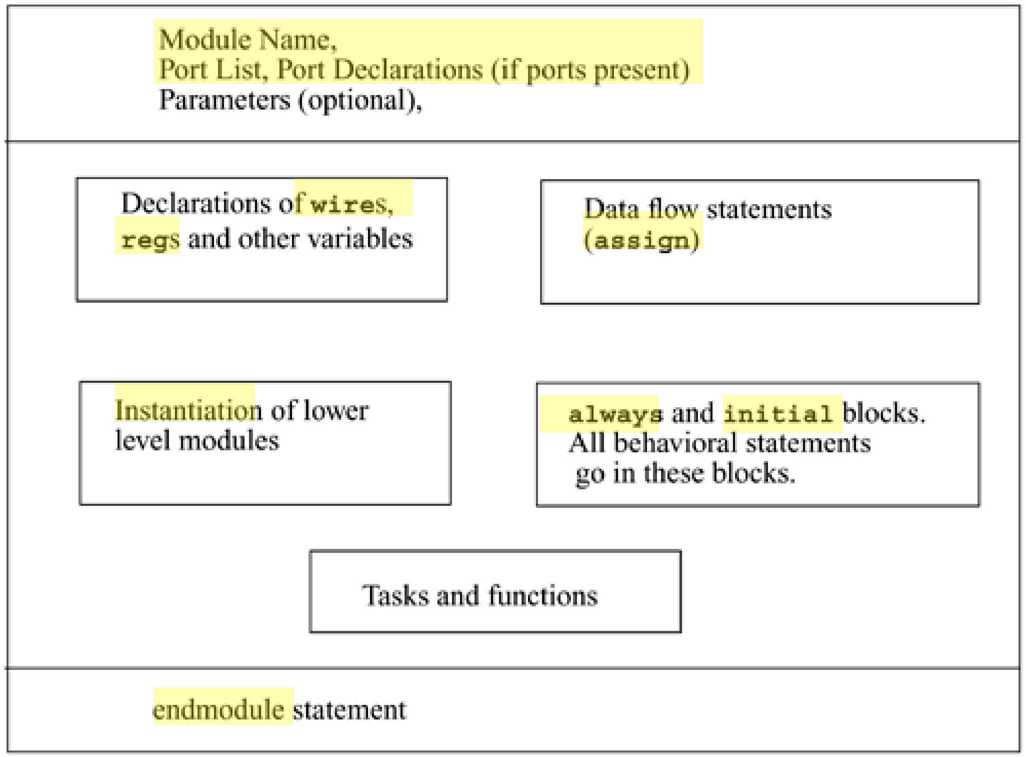
COMP311-1 Logic Circuit Design
Chap. 4

Instructor: Taigon Song (송대건)

2019 Fall

4.1. Modules

- Module: The basic building block in Verilog
  - Essential: module, module name, endmodule
  - Note: If no compile error, ordering is not an issue…?

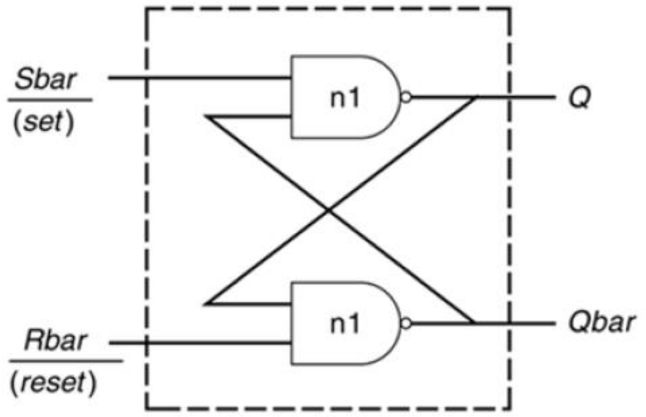

Module Name,
Port List, Port Declarations (if ports present)
Parameters (optional),
Declarations of wires, regs and other variables
Data flow statements (assign)
Instantiation of lower level modules
always and initial blocks. All behavioral statements go in these blocks.
Tasks and functions
endmodule statement

4.1. Modules

A Latch example

```
// This example illustrates the different components of a module

// Module name and port list
// SR_latch module
module SR_latch(Q, Qbar, Sbar, Rbar);

//Port declarations
output Q, Qbar;
input Sbar, Rbar;

// Instantiate lower-level modules
// In this case, instantiate Verilog primitive nand gates
// Note, how the wires are connected in a cross-coupled fashion.
nand n1(Q, Sbar, Qbar);
nand n2(Qbar, Rbar, Q);

// endmodule statement
endmodule
```

```
// Module name and port list
// Stimulus module
module Top;

// Declarations of wire, reg, and other variables

 wire q, qbar;
 reg set, reset;

 // Instantiate lower-level modules
 // In this case, instantiate SR_latch
 // Feed inverted set and reset signals to the SR latch
 SR_latch m1(q, qbar, ~set, ~reset);

 // Behavioral block, initial
 initial
 begin
   $monitor($time, " set = %b, reset= %b, q= %b\n",set,reset,q);
   set = 0; reset = 0;
   #5 reset = 1;
   #5 reset = 0;
   #5 set = 1;
 end

 // endmodule statement
 endmodule
```

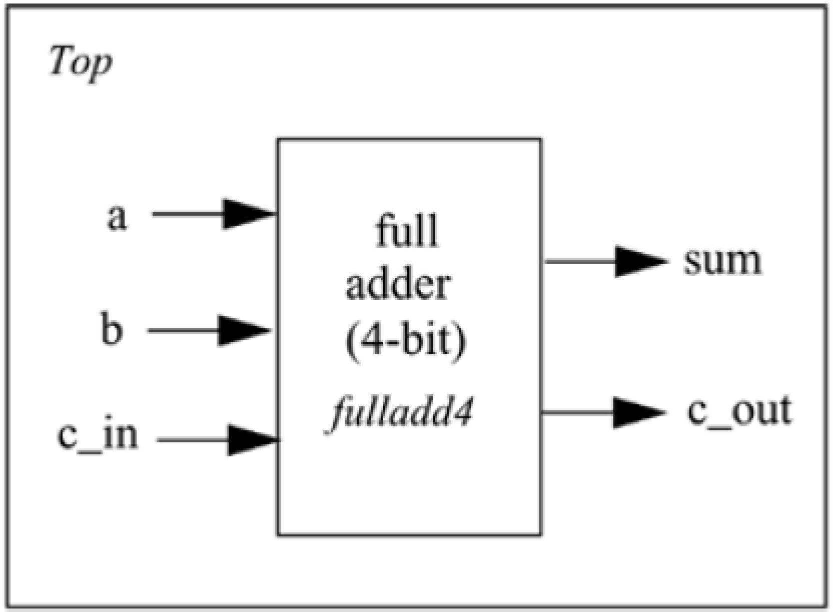


4.2. Ports

List of Ports

- Ports: Provide the interface by which a module can communicate with its environment
    - Either Input or Output (I/O)
    - “terminals” in other words

- Details of Fig. 4-3.
    - Top: top-lv module
    - Input: a, b, c_in
    - Output: sum, c_out



```
module fulladd4(sum, c_out, a, b, c_in); //Module with a list of ports
module Top; // No list of ports, top-level module in simulation
```

4.2. Ports

Port Declaration

- Input, output, or inout
  - If a port holds value, it should be **reg**
  - Inout cannot be reg

```
module fulladd4(sum, c_out, a, b, c_in);

//Begin port declarations section
output[3:0] sum;
output c_cout;

input [3:0] a, b;
input c_in;
//End port declarations section
...
<module internals>
...
endmodule
```

Verilog Keyword	**Type of Port**
input	Input port
output	Output port
inout	Bidirectional port

```
module DFF(q, d, clk, reset);
output q;
reg q; // Output port q holds value; therefore it is declared as reg.
input d, clk, reset;
...
...
endmodule
```

4.2. Ports

Port Declaration

- Can be declared like this too
    - ANSI C style

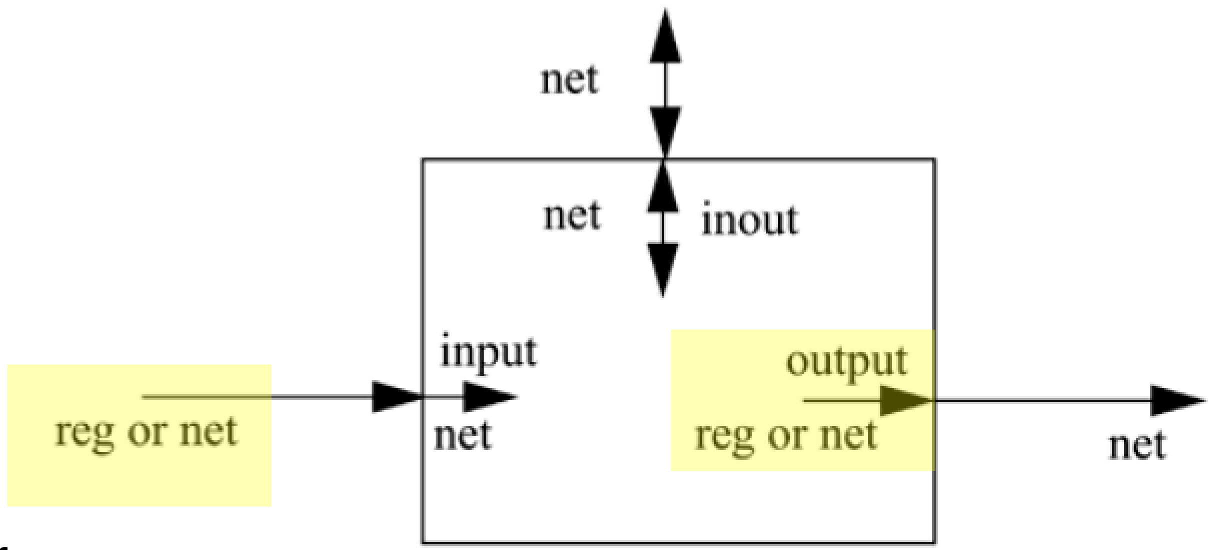
```
module fulladd4(output reg [3:0] sum,
                output reg c_out,
                input [3:0] a, b, //wire by default
                input c_in); //wire by default
...
<module internals>
...
endmodule
```

4.2. Ports

Port Connection Rules

- Connection rules must be followed
  - Inputs: internally, input ports are always ‘net’ type
  - Outputs: internally, output ports can be ‘reg’ or ‘net’ type
  - Inouts: internally, inout ports must always be ‘net’ type
  - Width matching: internal and external port widths should match
    - Warning when not matched

Figure 4-4. Port Connection Rules



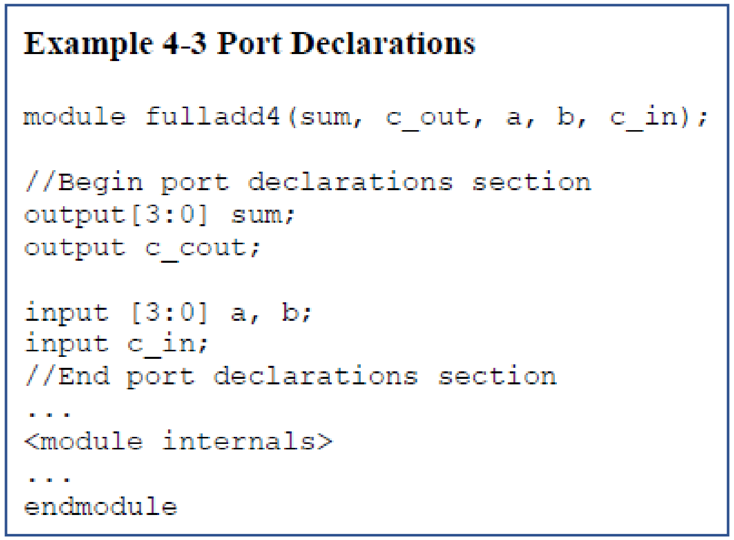
4.2. Ports

Port Connection Rules

- Unconnected ports
  - Ports can be unconnected for certain reasons

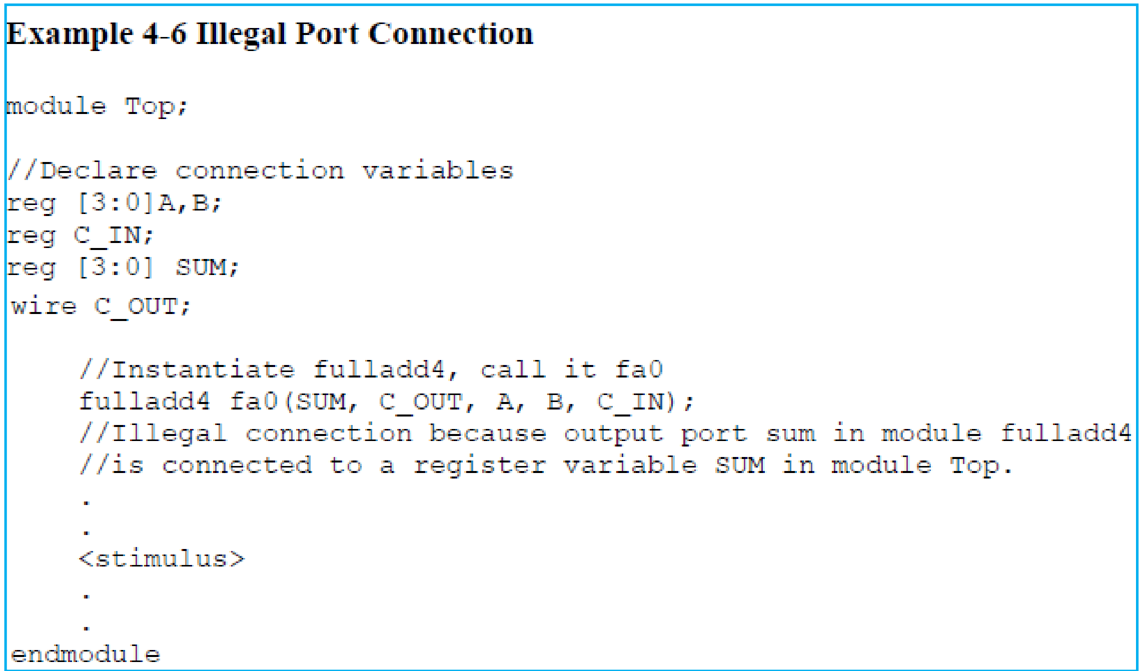
```
fulladd4 fa0(SUM, , A, B, C_IN); // Output port c_out is unconnected
```

- Illegal situations
  - Internal output connected to a reg



Example 4-3 Port Declarations

```
module fulladd4(sum, c_out, a, b, c_in);

//Begin port declarations section
output[3:0] sum;
output c_cout;

input [3:0] a, b;
input c_in;
//End port declarations section
...
<module internals>
...
endmodule
```



Example 4-6 Illegal Port Connection

```
module Top;

//Declare connection variables
reg [3:0]A,B;
reg C_IN;
reg [3:0] SUM;
wire C_OUT;

    //Instantiate fulladd4, call it fa0
    fulladd4 fa0(SUM, C_OUT, A, B, C_IN);
    //Illegal connection because output port sum in module fulladd4
    //is connected to a register variable SUM in module Top.
    .
    .
    <stimulus>
    .
    .
endmodule
```

4.2. Ports

Connecting Ports to External Signals

- Two methods possible, cannot be mixed
- Connecting by ordered list
    - Same order as the ports in the module definition

Example 4-7 Connection by Ordered List

```
module Top;

//Declare connection variables
reg [3:0]A,B;
reg C_IN;
wire [3:0] SUM;
wire C_OUT;

    //Instantiate fulladd4, call it fa_ordered.
    //Signals are connected to ports in order (by position)
    fulladd4 fa_ordered(SUM, C_OUT, A, B, C_IN);
    ...
    <stimulus>
    ...
endmodule
```

```
module fulladd4(sum, c_out, a, b, c_in);
output[3:0] sum;
output c_cout;
input [3:0] a, b;
input c_in;
    ...
    <module internals>
    ...
endmodule
```

- Connecting by name
    - Port name can be rearranged, ports can be dropped

```
// Instantiate module fa_byname and connect signals to ports by name
fulladd4 fa_byname(.c_out(C_OUT), .sum(SUM), .b(B), .c_in(C_IN),
.a(A),);
```

(.sum(SUM), .b(B), .c_in(C_IN), .a(A),);

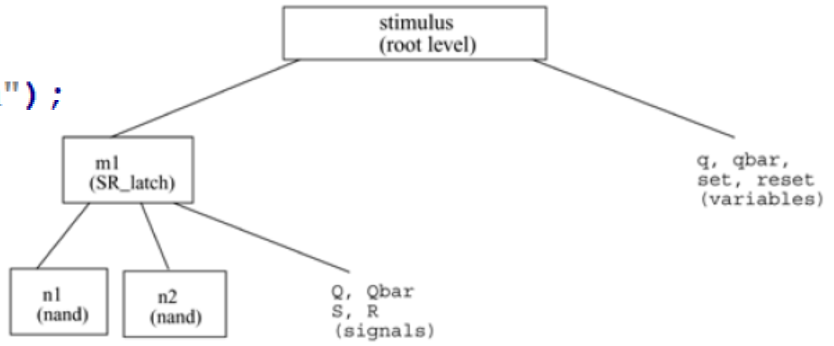
4.3. Hierarchical Names

- Every module instance, signal, or variable is defined with an identifier
  - A particular identifier has a unique place in the design hierarchy
  - A hierarchical name is a list of identifiers separated by ‘dots’ for each level of hierarchy
  - %m when printing in $display

  - E.g.) at any module:

```
initial $display("Hierarchy of this module: %m");
```

Figure 4-5. Design Hierarchy for SR Latch Simulation



Example 4-8 Hierarchical Names

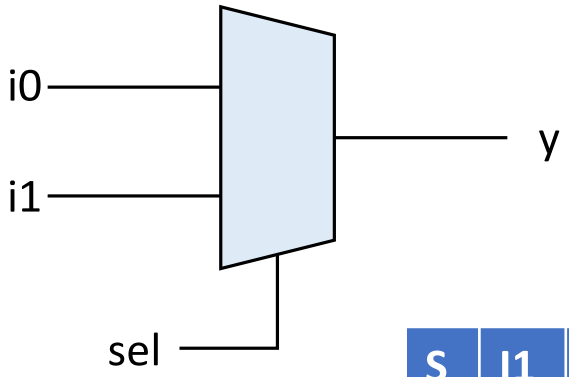
```
stimulus                    stimulus.q
stimulus.qbar               stimulus.set
stimulus.reset              stimulus.m1
stimulus.m1.Q               stimulus.m1.Qbar
stimulus.m1.S               stimulus.m1.R
stimulus.n1                 stimulus.n2
```

In class assignment

Design a 2:1 MUX

- Y = I0◦S’ + I1◦S

- By
  1. Primitive gates (1)
  2. (2) – not using INVs
  3. (3) – using only one AND keyword in the .v file
  4. Using the “assign” statement (1)
  5. (2) – using ternary operator (a ? b : c)
  6. Using “if/else” statement
  7. Using “case” statement

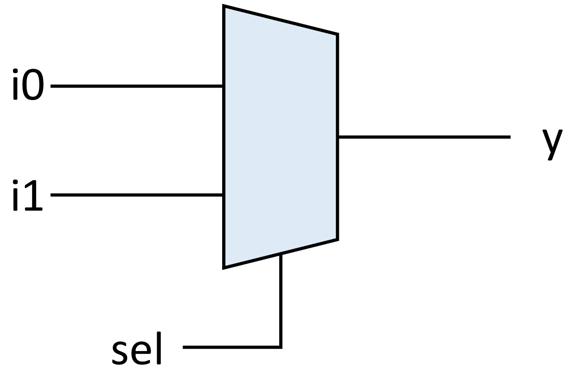


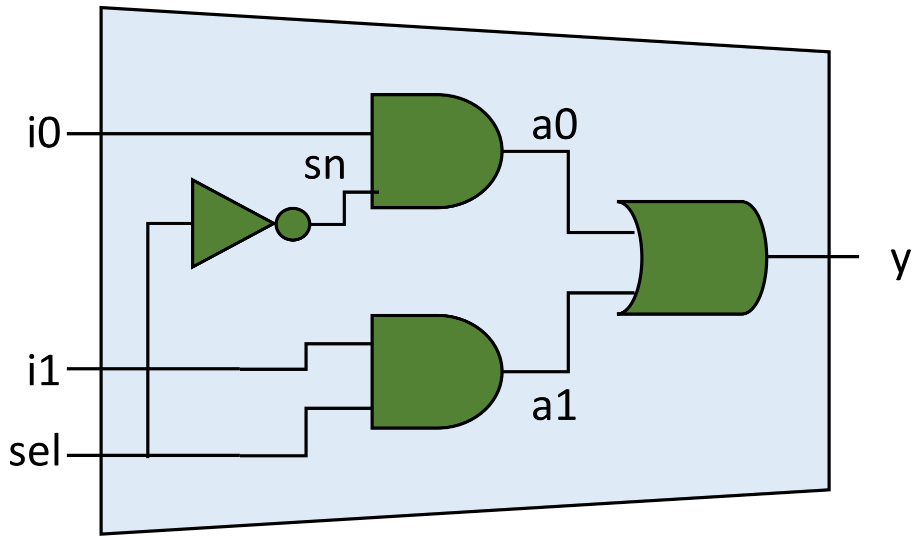
S	I1	I0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

In class assignment

Design a 2:1 MUX

- Primitive gates
  - not (out, in1)
  - buf (out, in1)
  - and (out, in1, in2, …, inN)
  - or (out, in1, in2, …, inN)
  - nand (out, in1, in2, …, inN)
  - nor (out, in1, in2, …, inN)
  - xor (out, in1, in2, …, inN)
  - xnor (out, in1, in2, …, inN)





In class assignment

Design a 2:1 MUX

- If (inside initial/always)

```
if (condition-1)
  statement-1;
else if(condition-2)
  statement-2;
…
else
  statement N
```

- Case (inside initial/always)

```
case (ctrl_sig)
  value-1: statement1;
  value-2: statement1;
  value-3: statement1;
  …
  default: statementN;
endcase
```

Homework #2

Design a 4:1 mux

- Design a 4:1 mux
  - 4bit input
  - 4bit output

  - Use $random when assigning initial values for your input
  - Use ‘for’ statement to change your sel values
  - Use any design methodology you prefer

- Due: 9/23 8:59am
  - Explain in simple words how you designed your module and testbench.
  - Screen capture your waveform results.
  - Write down both your .v and _tb.v in your report
  - Homework report should not exceed 3 pages. No cover page required. Report required to be submitted in English.
  - 3 pages, PDF online, and paper offline.