EE-421: Digital System Design

Verilog Operators
Expressing Numbers in Verilog
Hierarchical Design – Module Instantiation

Instructor: Dr. Rehan Ahmed [rehan.ahmed@seecs.edu.pk]

NUST
SCHOOL OF ELECTRICAL ENGINEERING & COMPUTER SCIENCE (SEECS)

Verilog Operators

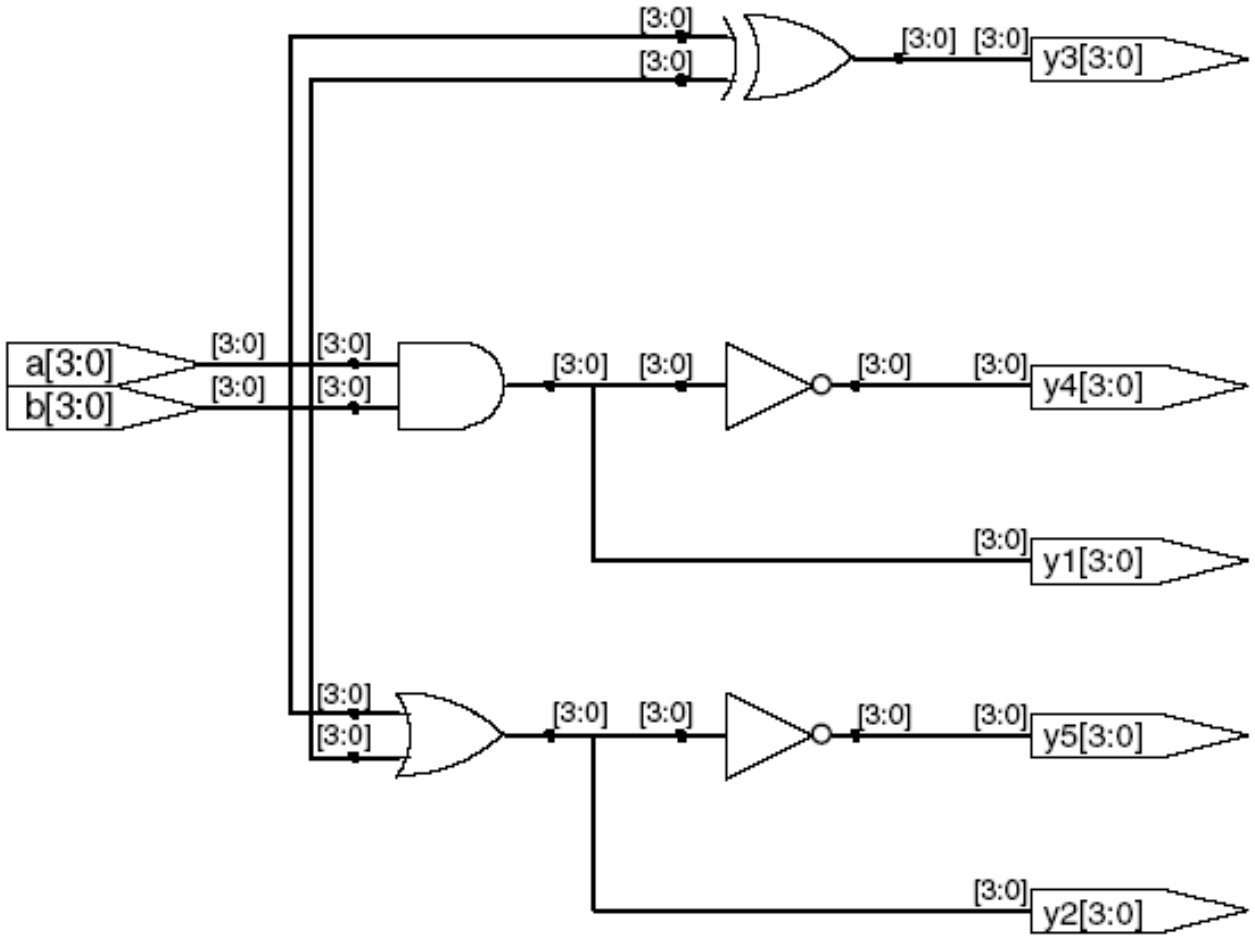
Verilog Operators and Bit Lengths: For Reference

Category	**Examples**	**Bit Length**
Bitwise	~A, +A, –A A & B, A | B, A ~^ B, A ^~ B	L(*A*) MAX (L(*A*), L(*B*))
Logical	!A, A && B, A || B	1 bit
Reduction	&A, ~&A, |A, ~|A, ^~ A, ~^ A	1 bit
Relational	A == B, A != B, A > B, A < B A >= B, A <= B A === B, A !== B	1 bit
Arithmetic	A + B, A – B, A * B, A / B A % B	MAX (L(*A*), L(*B*))
Shift	A << B, A >> B	L(*A*)
Concatenate	{A,…, B}	L(*A*) + ⋯ + L(*B*)
Replication	{B{A}}	B * L(*A*)
Condition	A ? B : C	MAX (L(*B*), L(*C*))

Bitwise Operators in Behavioral Verilog

```
module gates(input  [3:0]  a, b,
             output [3:0] y1, y2, y3, y4, y5);

   /* Five different two-input logic
      gates acting on 4 bit buses */

   assign y1 = a & b;     // AND
   assign y2 = a | b;     // OR
   assign y3 = a ^ b;     // XOR
   assign y4 = ~(a & b); // NAND
   assign y5 = ~(a | b); // NOR

endmodule
```

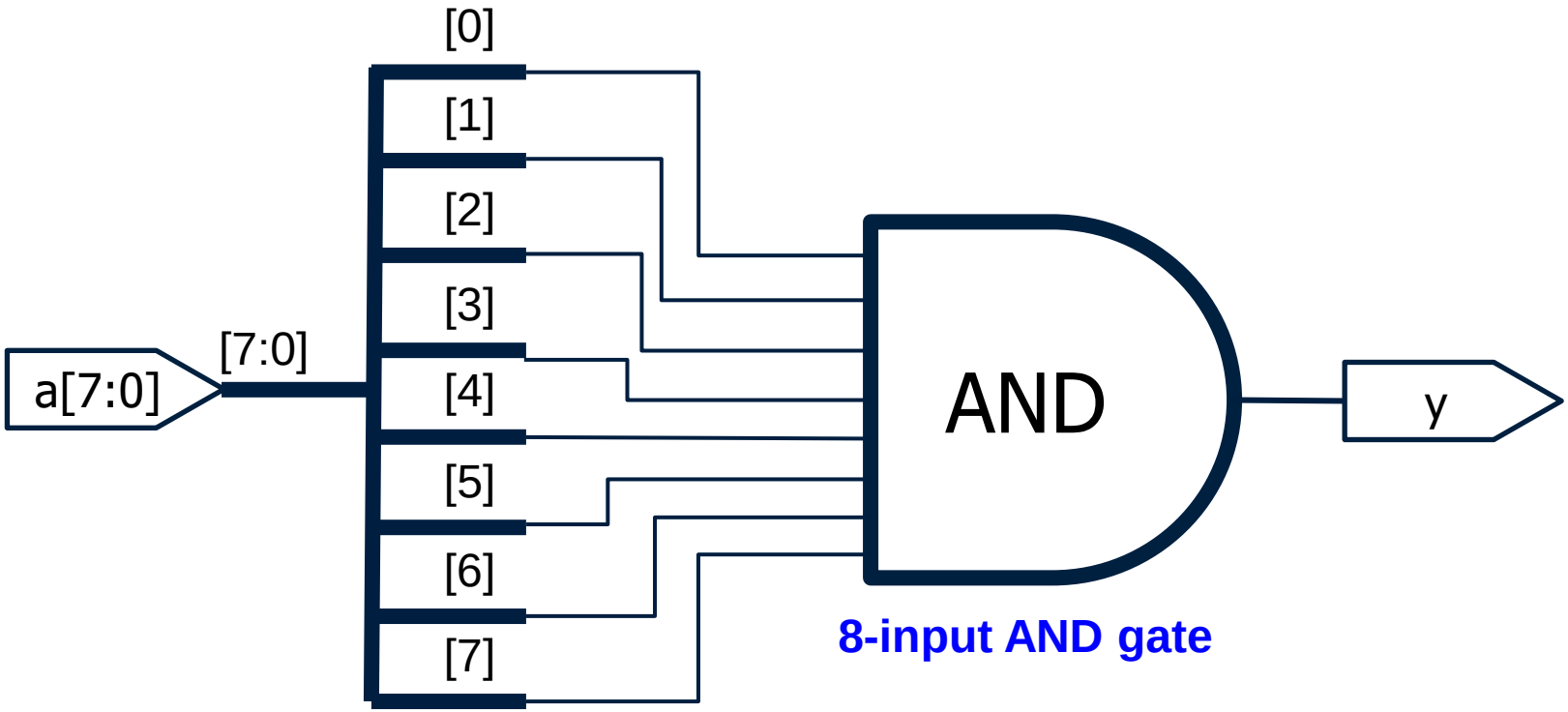
Bitwise Operators: Schematic View



Reduction Operators in Behavioral Verilog

```
module and8(input  [7:0] a,
            output       y);

   assign y = &a;

   // &a is much easier to write than
   // assign y = a[7] & a[6] & a[5] & a[4] &
   //            a[3] & a[2] & a[1] & a[0];

endmodule
```

Reduction Operators: Schematic View

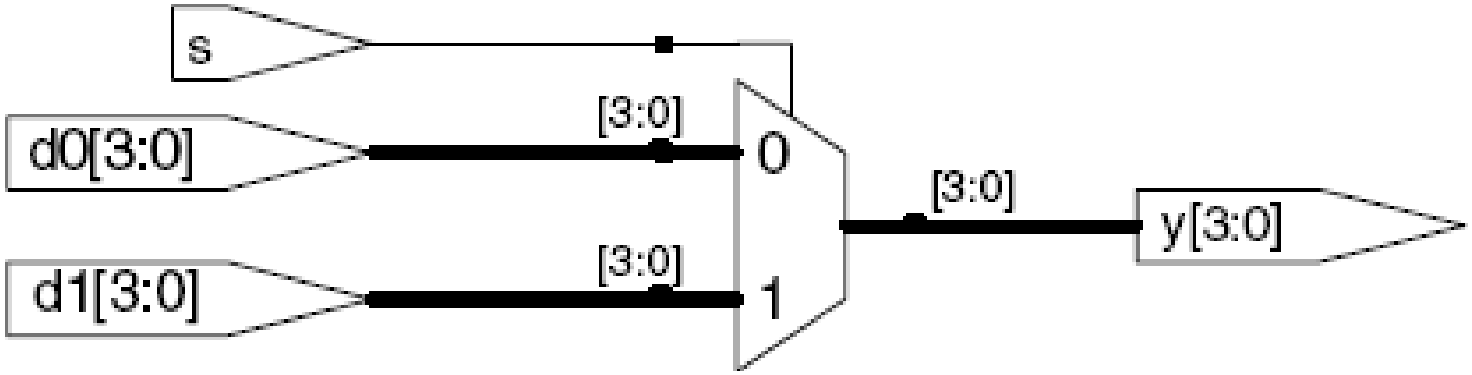


Conditional Assignment in Behavioral Verilog

```
module mux2(input  [3:0] d0, d1,
            input        s,
            output [3:0] y);

   assign y = s ? d1 : d0;
   // if (s) then y=d1 else y=d0;

endmodule
```

- ? : is also called a ternary operator as it operates on three inputs:
  - s
  - d1
  - d0

Conditional Assignment: Schematic View



More Complex Conditional Assignments

```
module mux4(input  [3:0] d0, d1, d2, d3
            input  [1:0] s,
            output [3:0] y);

   assign y = s[1] ? ( s[0] ? d3 : d2)
                   : ( s[0] ? d1 : d0);
  // if (s1) then
  //       if (s0) then y=d3 else y=d2
  // else
  //       if (s0) then y=d1 else y=d0

endmodule
```

Even More Complex Conditional Assignments

```
module mux4(input  [3:0] d0, d1, d2, d3
            input  [1:0] s,
            output [3:0] y);

   assign y = (s == 2'b11) ? d3 :
              (s == 2'b10) ? d2 :
              (s == 2'b01) ? d1 :
              d0;
// if      (s = "11" ) then y= d3
// else if (s = "10" ) then y= d2
// else if (s = "01" ) then y= d1
// else                     y= d0

endmodule
```

Verilog Operators and Bit Lengths: For Reference

Category	**Examples**	**Bit Length**
Bitwise	~A, +A, –A A & B, A | B, A ~^ B, A ^~ B	L(*A*) MAX (L(*A*), L(*B*))
Logical	!A, A && B, A || B	1 bit
Reduction	&A, ~&A, |A, ~|A, ^~ A, ~^ A	1 bit
Relational	A == B, A != B, A > B, A < B A >= B, A <= B A === B, A !== B	1 bit
Arithmetic	A + B, A – B, A * B, A / B A % B	MAX (L(*A*), L(*B*))
Shift	A << B, A >> B	L(*A*)
Concatenate	{A,…, B}	L(*A*) + ⋯ + L(*B*)
Replication	{B{A}}	B * L(*A*)
Condition	A ? B : C	MAX (L(*B*), L(*C*))

Precedence of Operations in Verilog

Highest

~	NOT
*, /, %	mult, div, mod
+, -	add,sub
<<, >>	shift
<<<, >>>	arithmetic shift
<, <=, >, >=	comparison
==, !=	equal, not equal
&, ~&	AND, NAND
^, ~^	XOR, XNOR
|, ~|	OR, NOR
?:	ternary operator

Lowest

Expressing Numbers in Verilog

How to Express Numbers ?

N'Bxx

8'b0000_0001

- (N) Number of bits
  - Expresses how many bits will be used to store the value
- (B) Base
  - Can be b (binary), h (hexadecimal), d (decimal), o (octal)
- (xx) Number
  - The value expressed in base
  - Can also have X (invalid) and Z (floating), as values
  - Underscore _ can be used to improve readability

Number Representation in Verilog

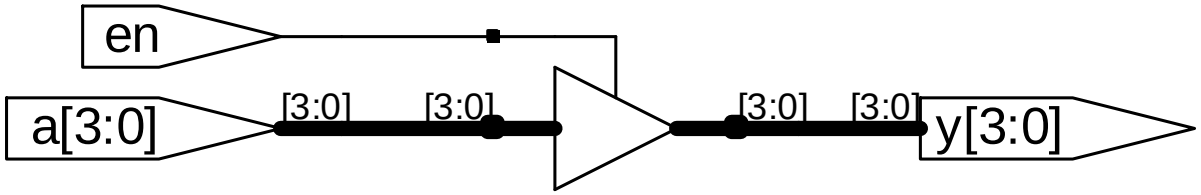
Verilog	Stored Number	Verilog	Stored Number
4’b1001	1001	4’d5	0101
8’b1001	0000 1001	12’hFA3	1111 1010 0011
8’b0000_1001	0000 1001	8’o12	00 001 010
8’bxX0X1zZ1	XX0X 1ZZ1	4’h7	0111
‘b01	0000 .. 0001	12’h0	0000 0000 0000

32 bits (default)

Reminder: Floating Signals (Z)

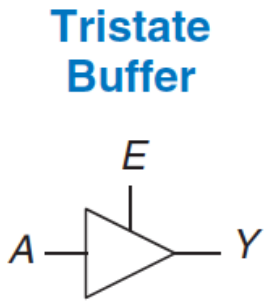
- Floating signal: Signal that is not driven by any circuit
  - Open circuit, floating wire
- Also known as: high impedance, hi-Z, tri-stated signals

```
module tristate_buffer(input  [3:0] a,
                       input        en,
                       output [3:0] y);

   assign y = en ? a : 4'bz;

endmodule
```



Recall: Tri-State Buffer

- A tri-state buffer enables gating of different signals onto a wire

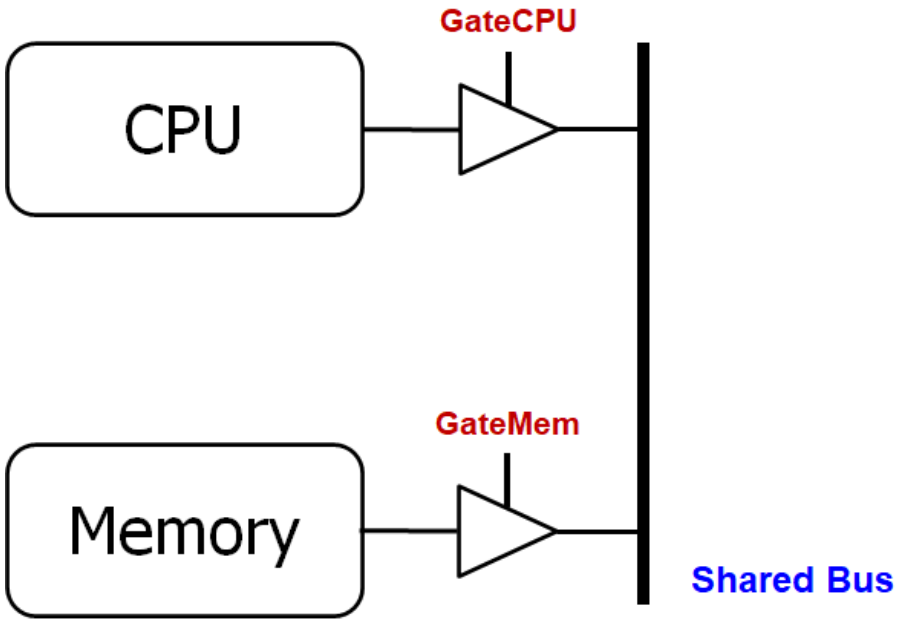


E	A	Y
0	0	Z
0	1	Z
1	0	0
1	1	1

Figure 2.40 Tristate buffer

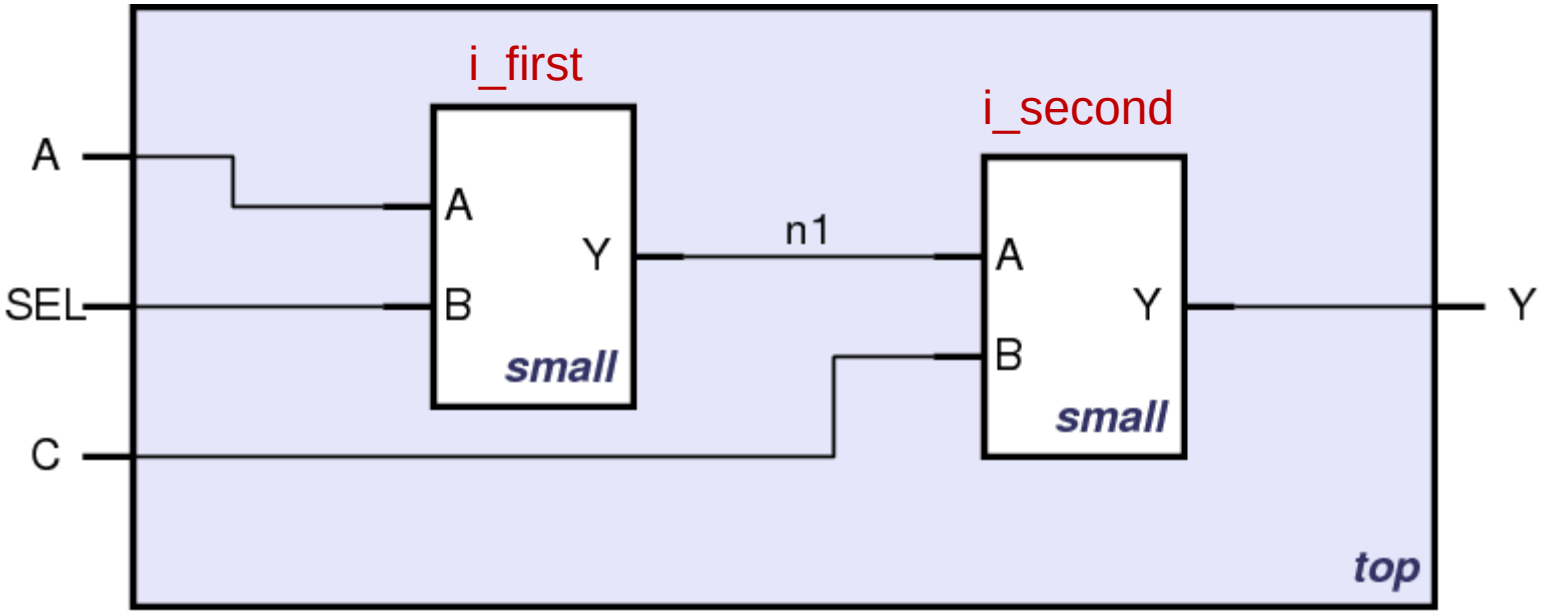
Recall: Example Use of Tri-State Buffers

- Imagine a wire connecting the CPU and memory
  - At any time only the CPU or the memory can place a value on the wire, both not both
  - You can have two tri-state buffers: one driven by CPU, the other memory; and ensure at most one is enabled at any time



Module Instantiation

Instantiating a Module

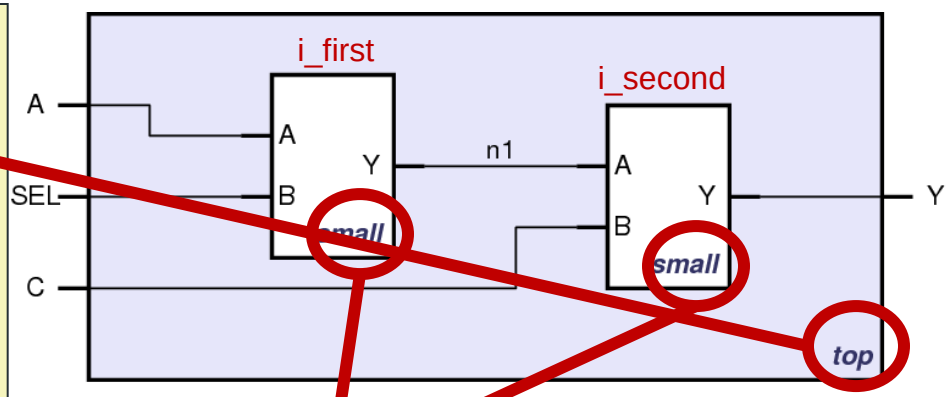


Schematic of module "top" that is built from two instances of module "small"

Instantiating a Module

- **Module Definitions in Verilog**

```
module top (A, SEL, C, Y);
  input A, SEL, C;
  output Y;
  wire n1;









endmodule
```

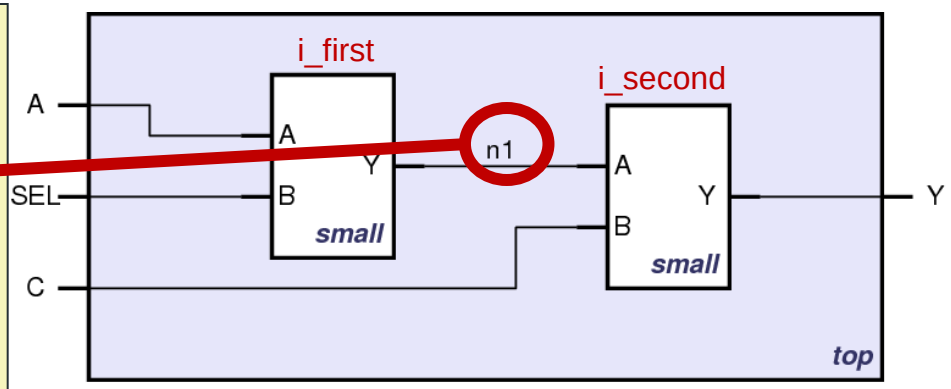



```
module small (A, B, Y);
  input A;
  input B;
  output Y;

// description of small

endmodule
```

Instantiating a Module

- **Defining wires (module interconnections)**

```
module top (A, SEL, C, Y);
  input A, SEL, C;
  output Y;
  wire n1;




endmodule
```

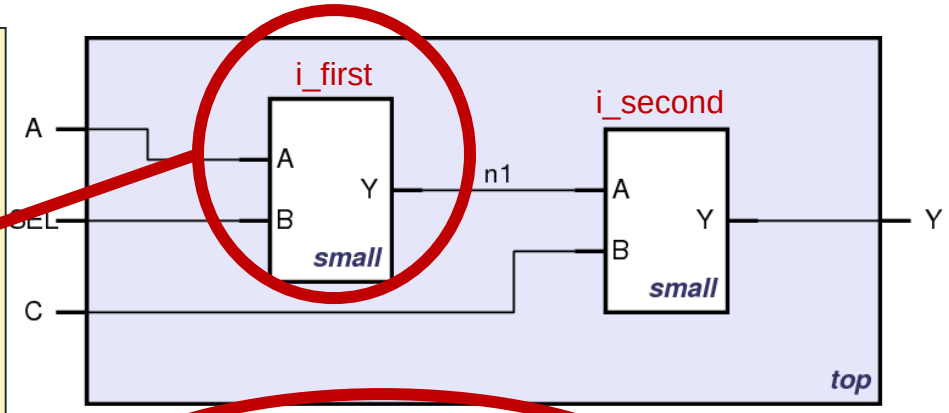



```
module small (A, B, Y);
  input A;
  input B;
  output Y;

// description of small

endmodule
```

Instantiating a Module

- **The first instantiation of the "small" module**

```
module top (A, SEL, C, Y);
  input A, SEL, C;
  output Y;
  wire n1;

// instantiate small once
small i_first ( .A(A),
                .B(SEL),
                .Y(n1)   );




endmodule
```

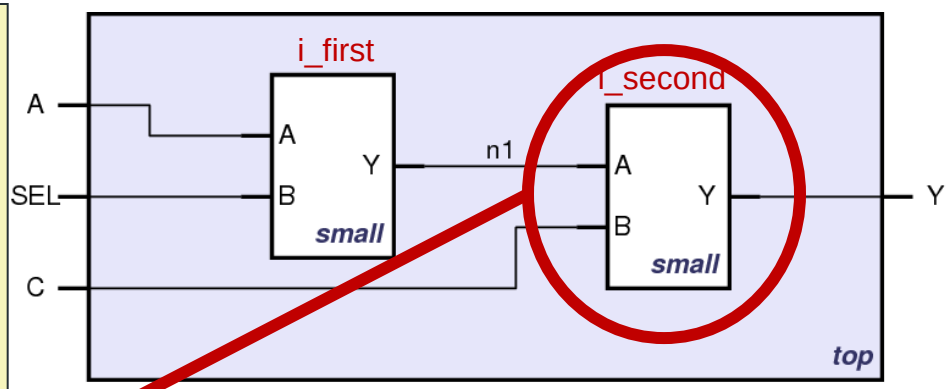



```
module small (A, B, Y);
  input A;
  input B;
  output Y;

// description of small

endmodule
```

Instantiating a Module

- **The second instantiation of the "small" module**

```
module top (A, SEL, C, Y);
  input A, SEL, C;
  output Y;
  wire n1;

// instantiate small once
small i_first ( .A(A),
                .B(SEL),
                .Y(n1)   );

// instantiate small second time
small i_second ( .A(n1),
           .B(C),
           .Y(Y) );

endmodule
```

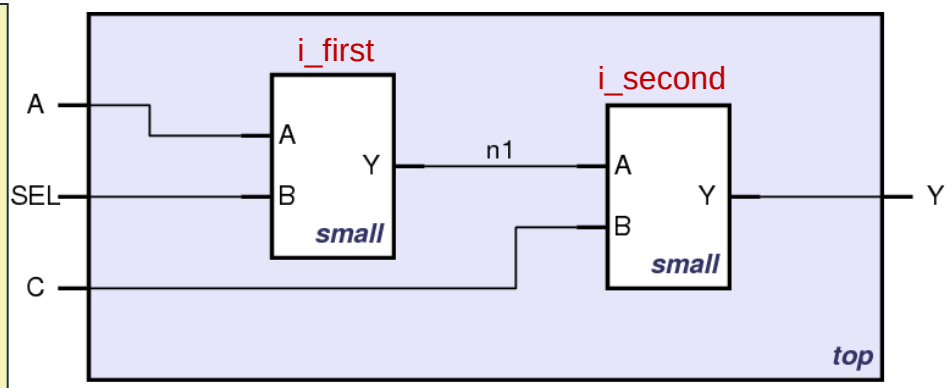



```
module small (A, B, Y);
  input A;
  input B;
  output Y;

// description of small

endmodule
```

Instantiating a Module

- **Short form of module instantiation**

```
module top (A, SEL, C, Y);
  input A, SEL, C;
  output Y;
  wire n1;

// alternative
small i_first ( A, SEL, n1 );

/* Shorter instantiation,
   pin order very important */

// any pin order, safer choice
small i_second ( .B(C),
                 .Y(Y),
                 .A(n1) );

endmodule
```




```
module small (A, B, Y);
  input A;
  input B;
  output Y;

// description of small

endmodule
```

Short form is not good practice
as it reduces code maintainability

Module Parametrization

Writing More Reusable Verilog Code

- We have a module that can compare two 4-bit numbers
- What if in the overall design we need to compare:
  - **5**-bit numbers?
  - **6**-bit numbers?
  - …
  - **N**-bit numbers?
  - Writing code for each case looks tedious
- What could be a better way?

Parameterized Modules

In Verilog, we can define module parameters

```
module mux2
  #(parameter width = 8)  // name and default value
   (input  [width-1:0] d0, d1,
    input              s,
    output [width-1:0] y);

  assign y = s ? d1 : d0;
endmodule
```

We can set the parameters to different values when instantiating the module

Instantiating Parameterized Modules

```
module mux2
  #(parameter width = 8)  // name and default value
   (input  [width-1:0] d0, d1,
    input              s,
    output [width-1:0] y);

  assign y = s ? d1 : d0;
endmodule
```

```
// If the parameter is not given, the default (8) is assumed
   mux2 i_mux (d0, d1, s, out);

// The same module with 12-bit bus width:
   mux2 #(12) i_mux_b (d0, d1, s, out);

// A more verbose version:
   mux2 #(.width(12)) i_mux_b (.d0(d0), .d1(d1),
                               .s(s), .out(out));
```