Design of Digital Circuits

Lab 2 Supplement: Mapping Your Circuit to an FPGA

Prof. Onur Mutlu
ETH Zurich

Spring 2021

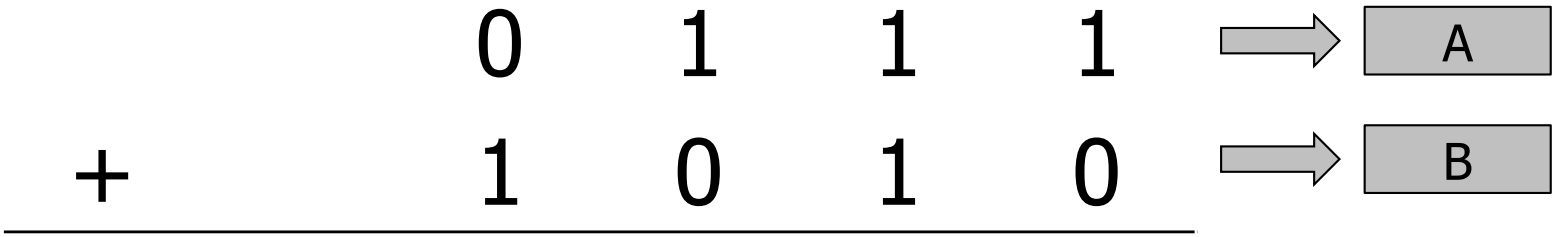
16 March 2021

What Will We Learn?

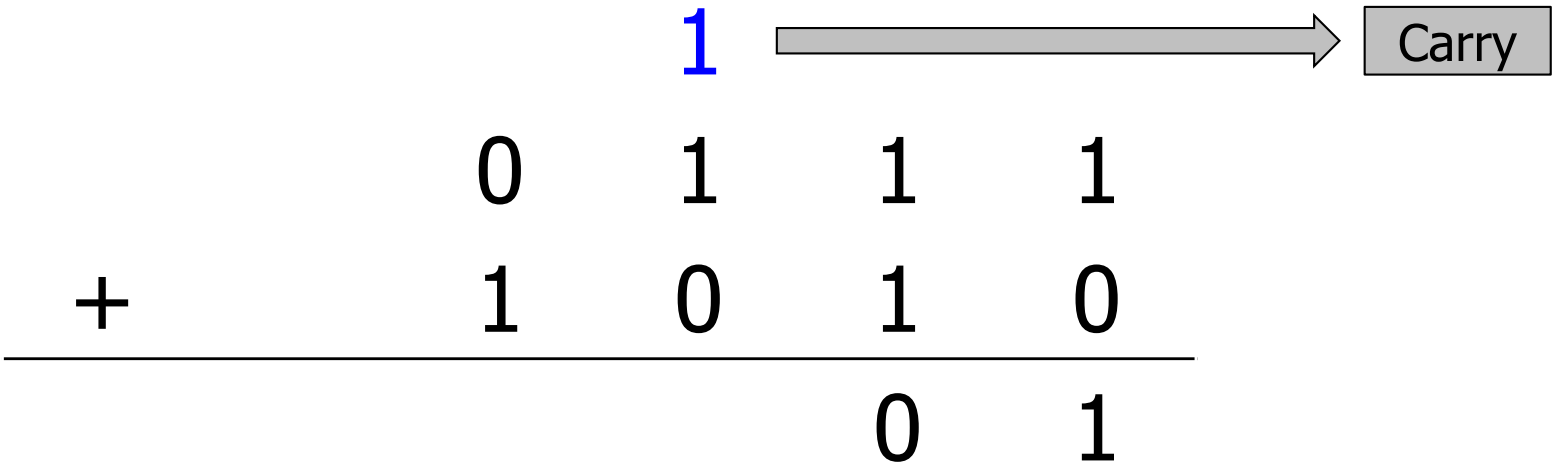
- In Lab 2, you will:
  - Design a 4-bit adder.
    - Design a 1-bit full-adder.
    - Use full-adders to design a 4-bit adder.
  - Learn how to map your circuits to an FPGA.
  - Program your FPGA using the Vivado Design Suite for HDL design.
  - Work with your FPGA board and see the results of your designs on the FPGA output.

Binary Addition

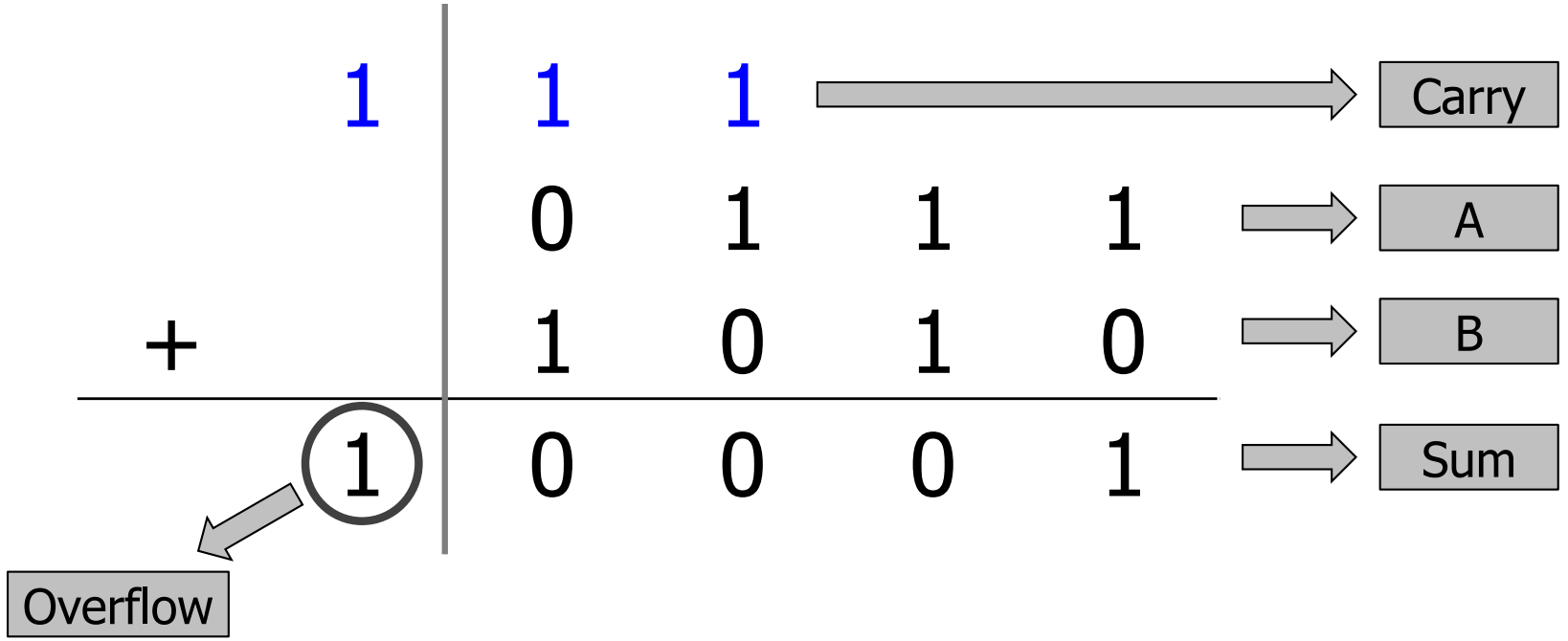
Binary Addition (1)



Binary Addition (2)



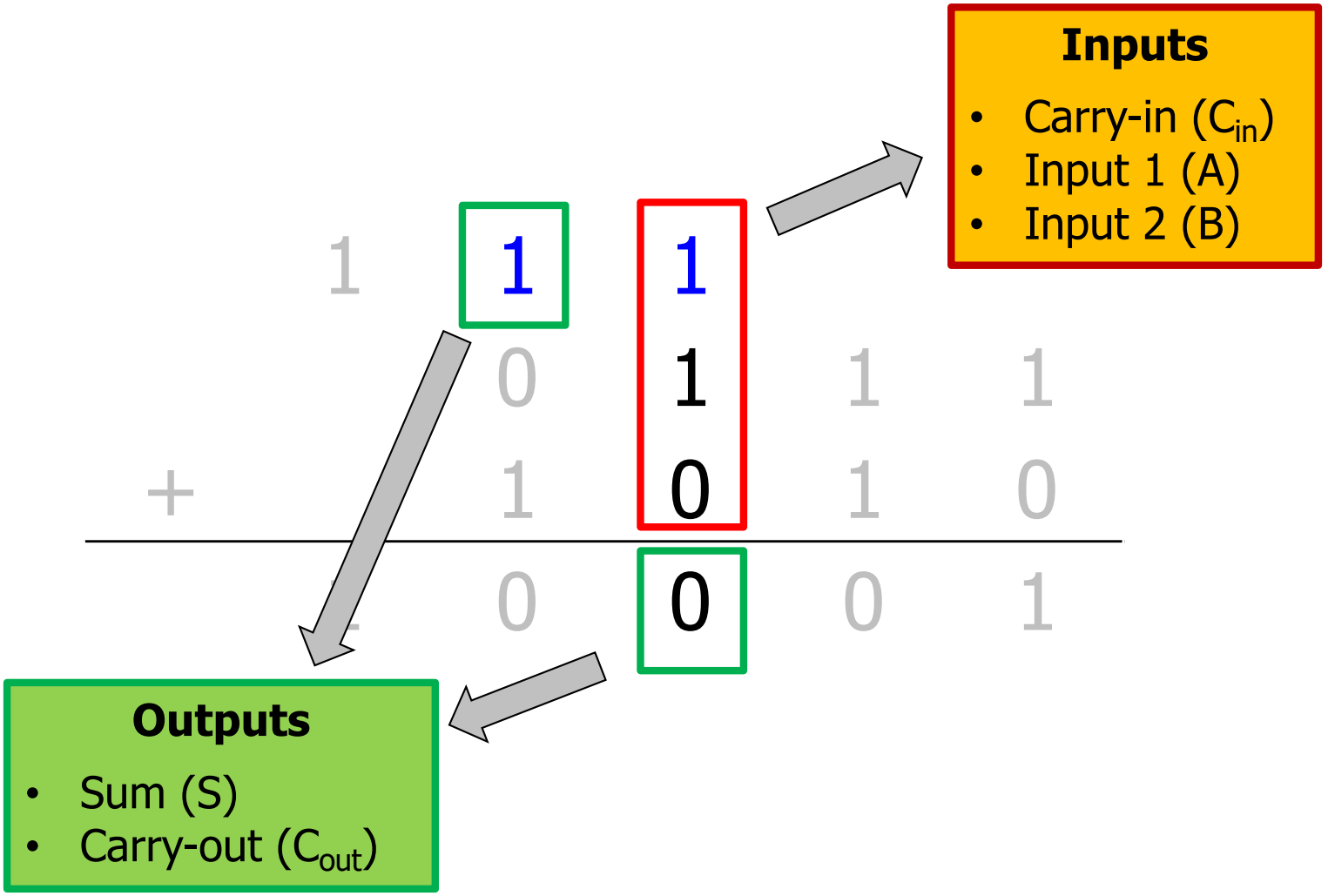
Binary Addition (3)



The Full-Adder (1)

		1	1	1		
			0	1	1	1
+			1	0	1	0
	1	0	0	0	1	

The Full-Adder (2)

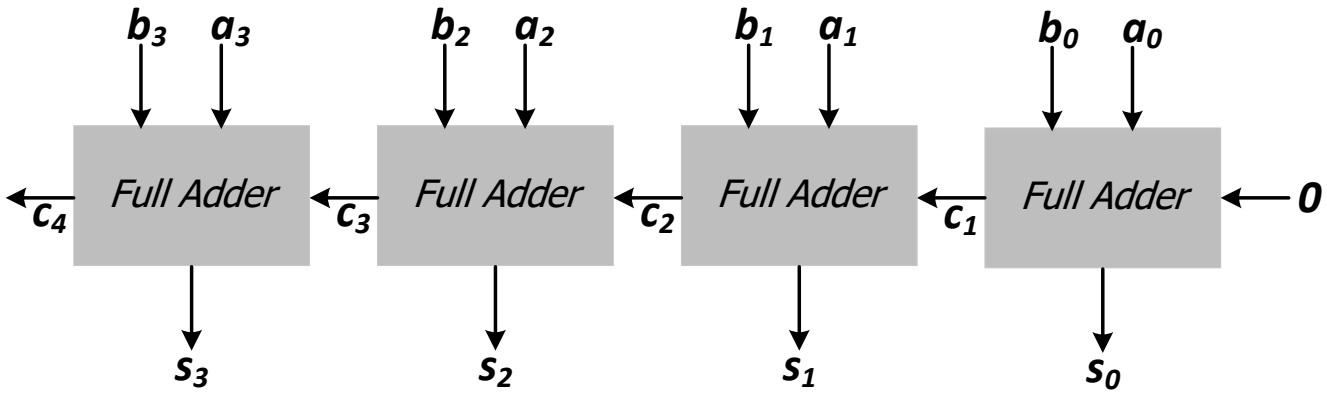


Design an Adder (1)

- Design a full-adder:
  - Inputs: input 1 (A), input 2 (B), carry-in (C_{in}).
  - Outputs: sum (S), carry-out (C_{out}).
  - All inputs and outputs are 1-bit wide.
- **Example:**
  - A = 1, B = 1, C_{in} = 1
  - S = 1, C_{out} = 1

Design an Adder (2)

- Design a 4-bit adder:
  - Receives two 1-bit numbers A and B and a 1-bit input carry (C_{in})
  - Returns outputs S and C as sum and carry of the operation, respectively.
- Example: A = 1110, B =0001, C_{in}=1
  - S = 0000
  - C = 1



- Hint: Use four full-adders to design a 4-bit adder.

Design an Adder (Overview)

1. You will use truth tables to derive the Boolean equation of the adder.
2. You will design the schematic of the circuit using logic gates.
3. You will use Vivado to write your design in Verilog.
4. You will use Vivado to program the FPGA.

Vivado Design Suite

- We will use the Vivado Design Suite for FPGA programming.
  - Vivado is installed in the computers in the lab rooms.
  - If you wish to use your own computer, you can follow these instructions:

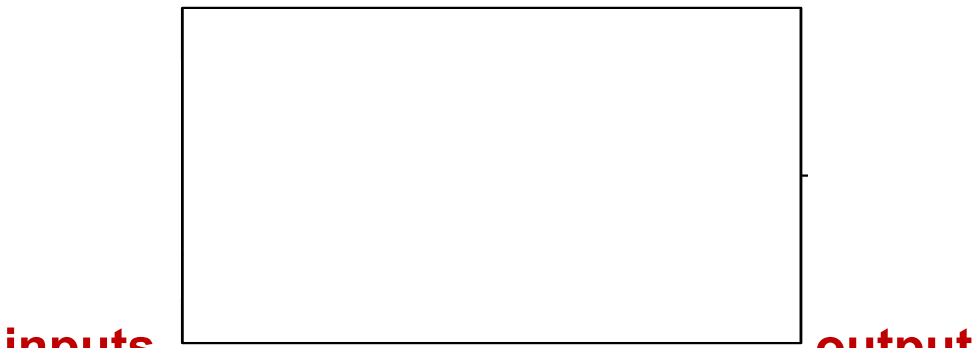
https://reference.digilentinc.com/learn/programmable-logic/tutorials/basys-3-getting-started/start

Verilog

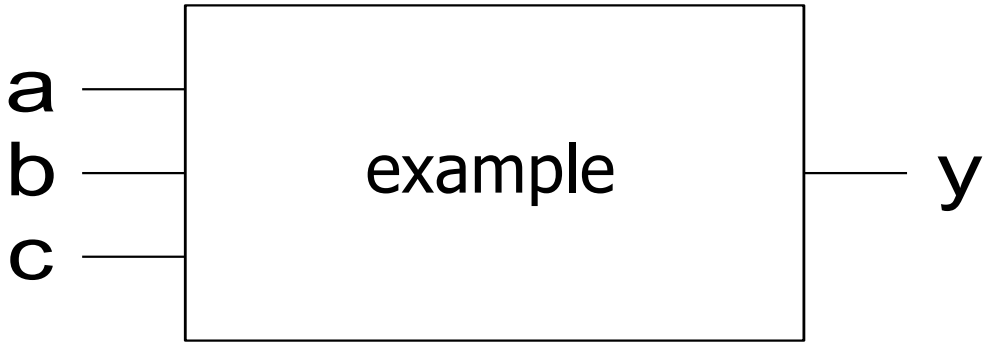
Defining a Module in Verilog

- A module is the main building block in Verilog.

- We first need to define:
  - Name of the module
  - Directions of its ports (e.g., input, output)
  - Names of its ports
- Then:
  - Describe the functionality of the module.



Implementing a Module in Verilog



name of module

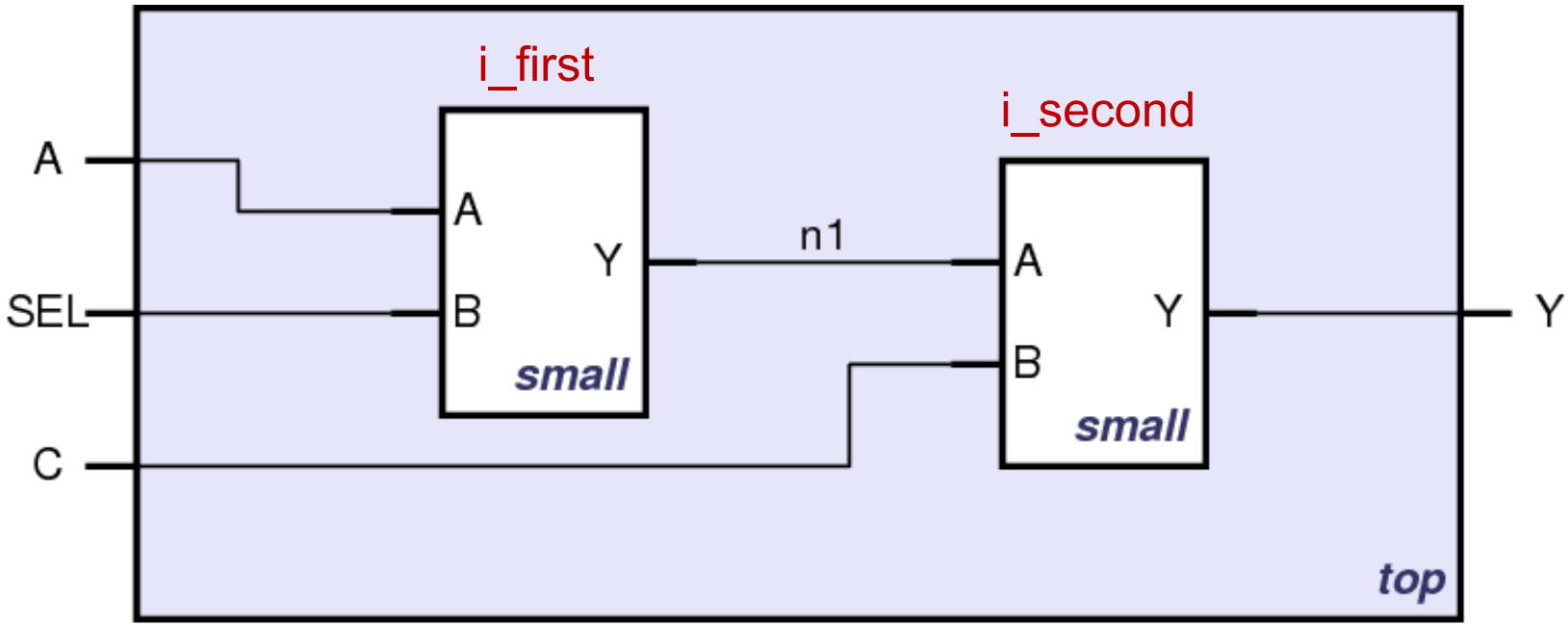
Port list (inputs and outputs)

```
module example (a, b, c, y);
        input a;
        input b;
        input c;
        output y;

// here comes the circuit description

endmodule
```

ports have a declared type

a module definition

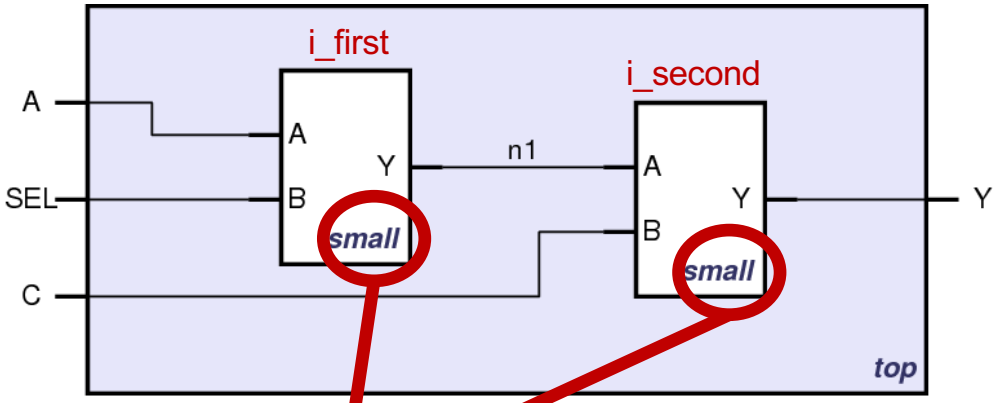
Structural HDL: Instantiating a Module



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

Structural HDL Example (1)

- **Module Definitions in Verilog**

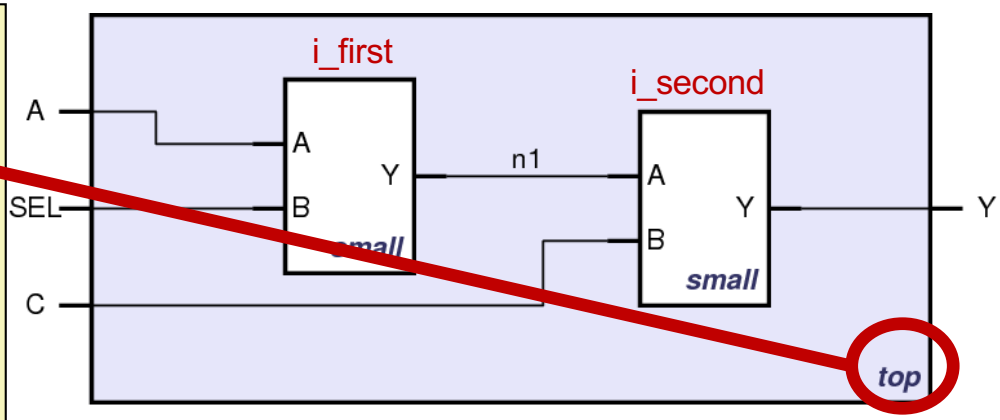



```
module small (A, B, Y);
  input A;
  input B;
  output Y;

// description of small

endmodule
```

Structural HDL Example (2)

- **Module Definitions in Verilog**

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









endmodule
```

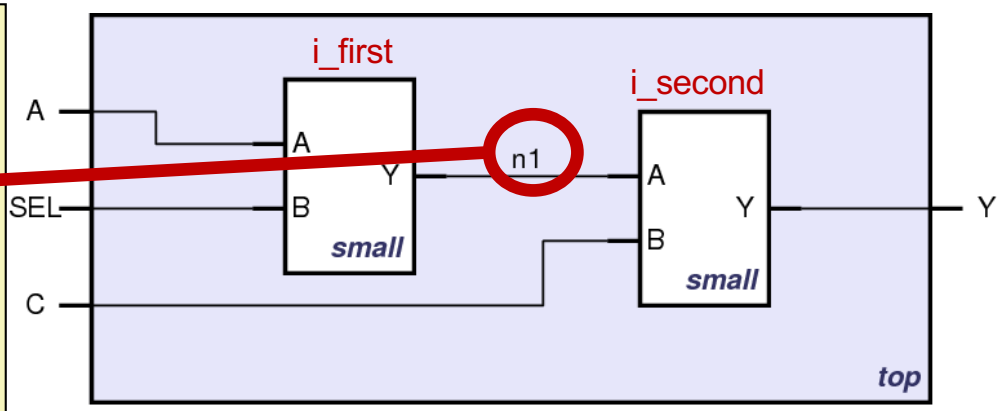



```
module small (A, B, Y);
  input A;
  input B;
  output Y;

// description of small

endmodule
```

Structural HDL Example (3)

- **Defining wires (module interconnections)**

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




endmodule
```

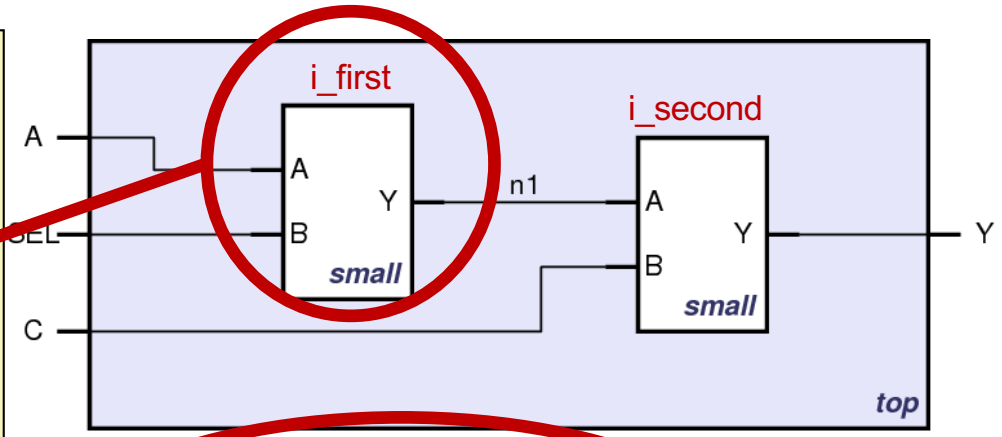



```
module small (A, B, Y);
  input A;
  input B;
  output Y;

// description of small

endmodule
```

Structural HDL Example (4)

- **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
```

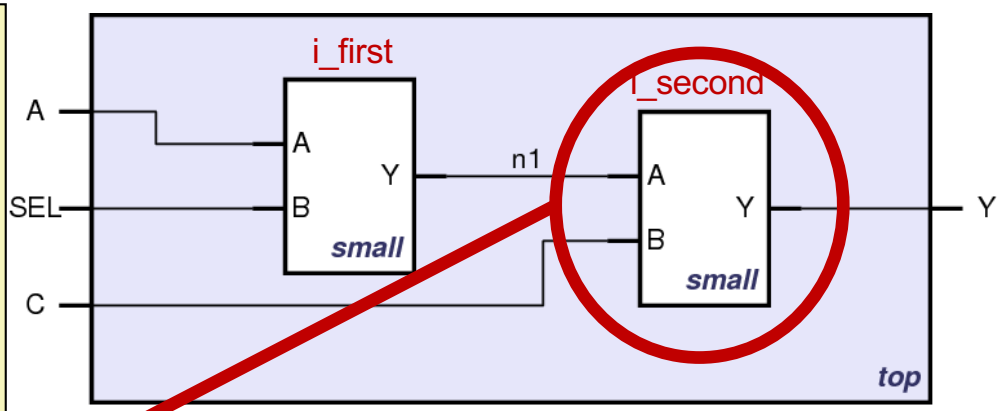



```
module small (A, B, Y);
  input A;
  input B;
  output Y;

// description of small

endmodule
```

Structural HDL Example (5)

- **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
```

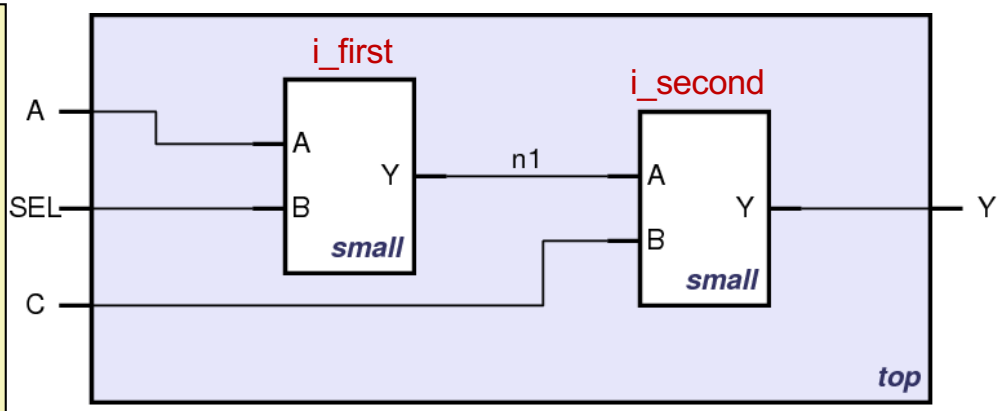



```
module small (A, B, Y);
  input A;
  input B;
  output Y;

// description of small

endmodule
```

Structural HDL Example (6)

- **Short form of module instantiation**

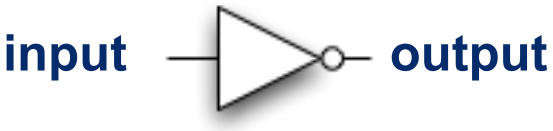
```
module top (A, SEL, C, Y);
  input A, SEL, C;
  output Y;
  wire n1;

// 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
```

What about logic gates?

Instantiating Logic Gates: The NOT Gate



A	$\overline{A}$
0	1
1	0

```
not my_not(output, input);
```

gate type — gate name — output signal — input signal

Instantiating Logic Gates: The AND Gate (1)

A	*B*	*A* • *B*
0	0	0
0	1	0
1	0	0
1	1	1

gate type gate name output signal input signals

```
and my_and(output, input1, input2);
```

Instantiating Logic Gates: The AND Gate (2)

A	B	A • B
0	0	0
0	1	0
1	0	0
1	1	1

gate type gate name output signal input signals

```
and my_and(output, input1, input2);
```

or...

```
and my_and(output, input1, input2, input3, ...);
```

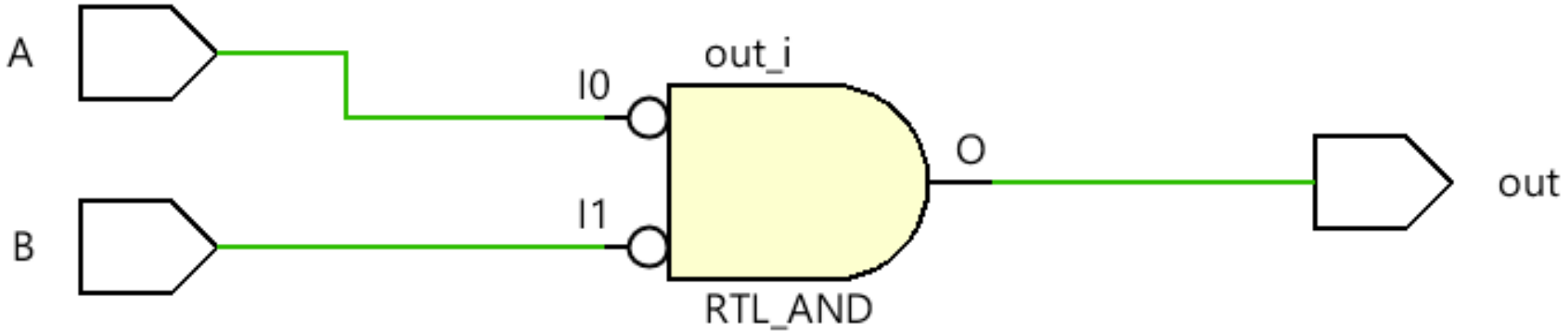
gate type gate name output signal input signals
(as many as you want)

Instantiating Logic Gates: More Gates…

```
      not my_not(output, input);


 and my_and(output, input1, input2, ...),
  or my_or(output, input1, input2, ...);
 xor my_xor(output, input1, input2, ...);


nand my_nand(output, input1, input2, ...);
 nor my_nor(output, input1, input2, ...);
xnor my_xnor(output, input1, input2, ...);
```

Instantiating Logic Gates: A Simple Example (1)

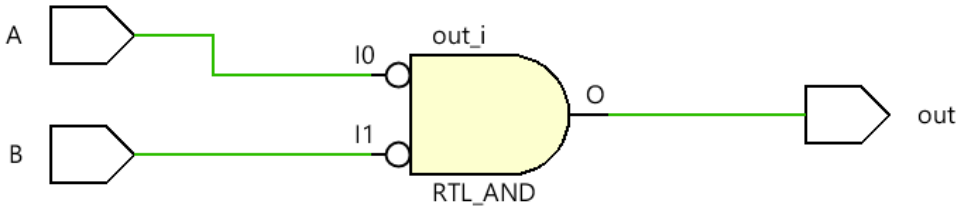
$$\boldsymbol{out = \overline{A} \cdot \overline{B}}$$



Instantiating Logic Gates: A Simple Example (2)

```
// Notice the alternative port declaration
module check_zeros (
    input A,
    input B,
    output out
    );

    // wires for the intermediate signals
    wire not_a, not_b;

    // instantiate the NOT gates
    not(not_a, A);
    not(not_b, B);

    // instantiate the AND gate
    and(out, not_a, not_b);

endmodule
```

$$out = \overline{A} \cdot \overline{B}$$

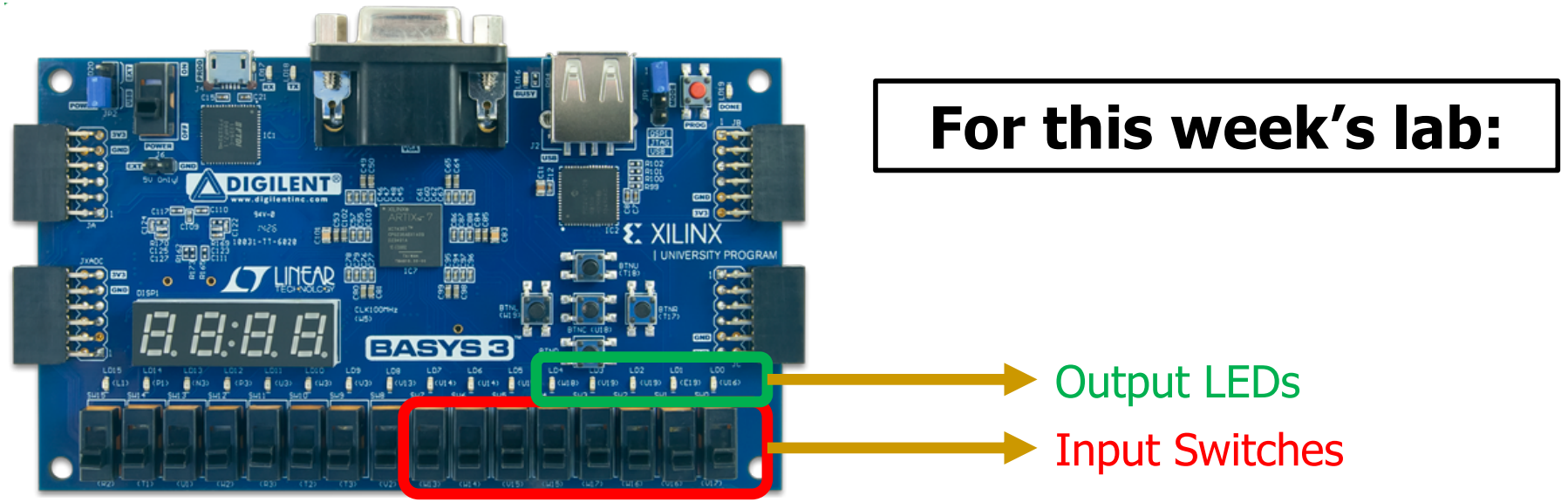


Basys 3 FPGA Board

In this course, we will be using the Basys 3 boards from Digilent, as shown below.

You can learn more about the Basys 3 Starter Board from:

http://store.digilentinc.com/basys-3-artix-7-fpga-trainer-board-recommended-for-introductory-users/



Last Words

- In this lab, you will map your circuit to an FPGA.
- First you will design a full-adder. You will then use the full-adder as a building block to build a 4-bit adder.
- Then, you will learn how to use Xilinx Vivado for writing Verilog and how to connect to the Basys 3 board.
- Finally, you will program the FPGA and get the circuit running on the FPGA board.
- You will find more exercises in the lab report.

Report Deadline

23:59, 2 April 2021

Design of Digital Circuits

Lab 2 Supplement:
Mapping Your Circuit to FPGA

Prof. Onur Mutlu
ETH Zurich

Spring 2021

16 March 2021