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)

\[
\begin{array}{cccccc}
0 & 1 & 1 & 1 \\
+ & 1 & 0 & 1 & 0 \\
\hline
1 & 0 & 1 & 0 \\
\end{array}
\]
Binary Addition (2)

\[
\begin{array}{cccc}
0 & 1 & 1 & 1 \\
+ & 1 & 0 & 1 & 0 \\
\hline
0 & 1 & 0 & 1 \\
\end{array}
\]

Carry
Binary Addition (3)

\[
\begin{array}{ccc|cc}
1 & 1 & 1 & 0 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 \\
+ & 1 & 0 & 1 & 0 & 0 & 1 \\
\hline
1 & 0 & 0 & 0 & 0 & 1 & 1 \\
\end{array}
\]

Carry
A
B
Sum
Overflow
The Full-Adder (1)

\[
\begin{array}{c}
\begin{array}{c}
1 \\
0 \\
1
\end{array}
\end{array}
\begin{array}{c}
+ \\
1 \\
0
\end{array}
\begin{array}{c}
1 1 1 \\
1 1 1 \\
1 0 1 0
\end{array}
\begin{array}{c}
\hline
1 0 0 0 1
\end{array}
\]
The Full-Adder (2)

Inputs
• Carry-in ($C_{in}$)
• Input 1 (A)
• Input 2 (B)

Outputs
• Sum (S)
• Carry-out ($C_{out}$)

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

1 0 1 0
0 0 0 1
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 \((C_{\text{in}})\)
  - Returns outputs S and C as sum and carry of the operation, respectively.

- **Example:** \(A = 1110, B = 0001, C_{\text{in}} = 1\)
  - \(S = 0000\)
  - \(C = 1\)

- **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**.
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

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

    // here comes the circuit description
endmodule
```
Schematic of module "top" that is built from two instances of module "small"
Module Definitions in Verilog

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

    // description of small

endmodule
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)

```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
```
The first instantiation of the “small” module

```verilog
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
```
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
Short form of module instantiation

```verilog
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

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

<table>
<thead>
<tr>
<th>A</th>
<th>( \overline{A} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Instantiating Logic Gates: The AND Gate (1)

A
\[ A \cdot B \]
B

<table>
<thead>
<tr>
<th>( A )</th>
<th>( B )</th>
<th>( A \cdot B )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

```
and my_and(output, input1, input2);
```
Instantiating Logic Gates: The AND Gate (2)

The AND Gate (2)

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

or...

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

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>$A \cdot B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The truth table for the AND gate:

- Gate type: AND
- Gate name: my_and
- Output signal: output
- Input signals: input1, input2

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)

\[ \text{out} = \overline{A} \cdot \overline{B} \]
// 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

\[ \text{out} = \overline{A} \cdot \overline{B} \]
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/

For this week’s lab:

- Output LEDs
- Input Switches
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.