
Department of Electrical and Computer Engineering
ENEL 453: Digital System Design
Fall 2017

# Lab 1: Introduction to Simulation, Synthesis and Implementation on an FPGA

## Pre-lab Exercise

## 1 Overview

**This lab is worth 5% of your term grade and broken up as:**

- Pre-lab: 1%
- In-lab simulation: 2%
- In-lab demonstration: 2%

The pre-lab exercises must be completed **individually** prior to the scheduled lab period. Yellow boxes in this document indicate what must be completed for the pre-lab. You may simply hand-write your answers.

## 2 The Basys3 Board

In the labs you will be using an FPGA (Field-Programmable Gate Array) Artix-7 FPGA chip from Xilinx on the development board Basys3 from Digilent.

An FPGA is a programmable device that is different from a conventional microcontroller:

- With a microcontroller, like an Arduino, the chip is already designed for you; you write software usually in C or C++ and compile it to a hex file that you load onto the microcontroller; the microcontroller stores the program in flash memory until it is erased or replaced. Thus, you have control over the software.

- With FPGAs, you are the one designing the circuit. There is no processor to run software on, no registers, no program counter, no ALU built in to the chip. You can configure an FPGA to be as simple as a logic gate, or as complex as a multi-core processor. You can design the FPGA to be a processor that you then can write software for! In fact, companies like Intel or nVidia, often use FPGAs to prototype their chips before creating them.

- To create your design on an FPGA, you use a HDL (Hardware Description Language), which in our labs is VHDL; you then synthesize your HDL into a bit file used to configure the FPGA. Thus, you have control over the hardware.

- HDL statements execute concurrently (this is how logic circuits work!), making it a general purpose programming language for parallel implementations. Due to this, FPGA designs are often much faster than the sequential processors.

- Unfortunately, FPGAs store their configuration in RAM, not flash, meaning that once they lose power they lose their configuration; so they must be configured every time power is applied. But, there are flash chips that will automatically configure the stored bit file on power up.

- With a microprocessor, you have dedicated pins for specific features; if you want to use some other pins, your only solution is to use software to emulate a serial port (for example), thus wasting valuable processor time. With an FPGA, it is up to you to decide what pins your ports connect to, or create as many ports as you want; the only limitations are the number of physical I/O pins and the size of the FPGA. You have to manually specify how to connect everything that you design to the outside world. For this we use "user constraint files" - you will get a chance to make a few of these in the lab!

Figure 1 shows a labeled photo of a Basys3. For this part of the pre-lab, you will need to inspect its manual

`https://reference.digilentinc.com/_media/basys3:basys3_%20rm.pdf`

and find the pin definitions for the following elements:

- Clock (e.g. the pin definition for Clock is W5)
- Push button BTND
- Slide switches SW0 - SW2
- LEDs LD0 - LD2.

**Begin your pre-lab:**
Record your answers. These values will be used when we need to specify how to connect our design.

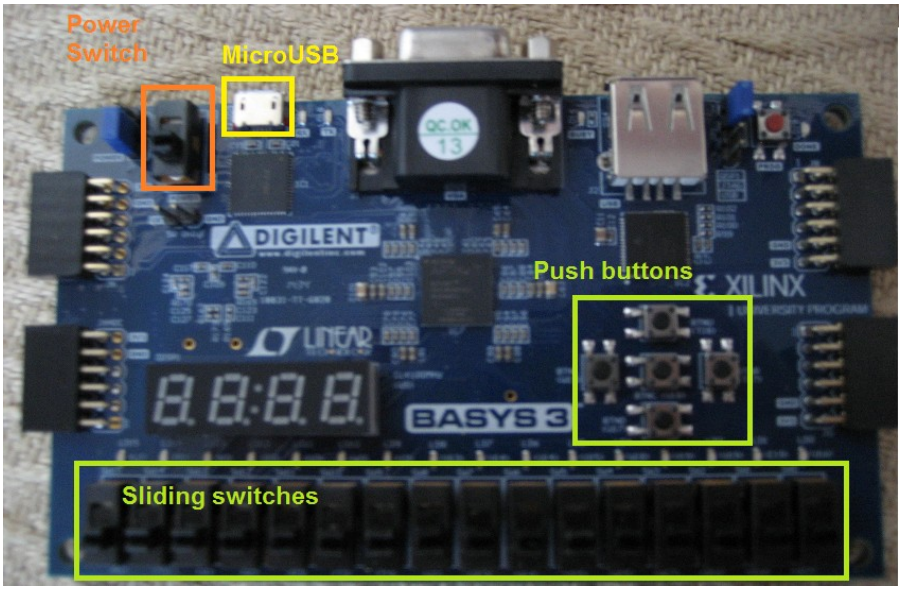


Figure 1: Basys3 Board labeled with relevant components for Lab 1.

# 3 Logic circuit design

Consider the logic function W (it is Exclusive-OR, called EXOR or XOR):

```
W = A xor B
```

**Add a labeled diagram:**
Draw a logic gate diagram for this expression for W; use only NOT, AND, and OR gates.

**Add a labeled table:**
Now create a truth table for W (and either draw it in one of the boxes, or on your own sheet of paper). This will be used for testing the input-output combinations when you implement the design.

Next, consider a new logic circuit,

```
V = C and W
```

Substitute W into the expression for V and rewrite it in terms of A, B and C.

**Add an expression:**
Write down your final expression for V.

**Add a labeled diagram:**
Draw a logic gate diagram for this expression for V; build it out of NOT, AND, and OR gates only.

**Add a labeled table:**
Add a truth table for V.

# 4 Testing

Digital designs are tested using testbenches that assign input values and prompts the output changes, like this:

```
-> assign a value to A, B and C
-> wait a bit for the output to become stable (depends on design)
-> check if output is what we want
-> assign new values to A, B and C
-> wait a bit for the output to become stable
-> check if output is what we want
... etc ...
```

This can easily get tedious, if you are trying to get full coverage of a truth table. Conveniently, you can force any signal to toggle between ’0’ and ’1’ periodically (duty cycle doesn’t have to be 50/50 though).

Analyze Figure 4. Copy the diagram down onto a page where you are recording your answers. Notice that A is ticking along at a certain frequency, and B is ticking away at 1/2 that (twice as slow).

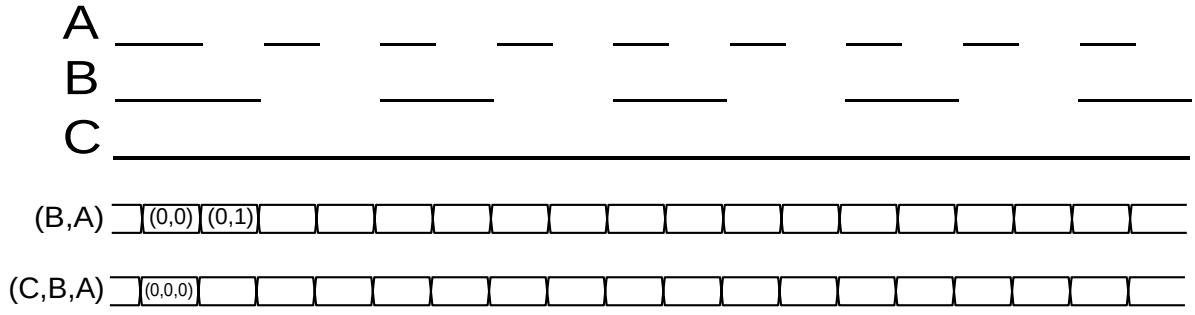


Figure 2: Timing diagram for use in the testbench.

**Complete the timing chart:**
Fill in the blanks for the row labeled `(B,A)`; it has been started with `(0,0)` and `(0,1)`. Each box corresponds to the states/values of A and B immediately above it. You will need it to test for function W.

You should notice that this method systematically tests all combinations of A and B.

**Complete the timing chart:**
Now, draw the waveform that C must follow in order to automatically cover the entire truth table (that you previously created). Verify this by filling out the row labeled `(C,B,A)`. You will need it to test for function V.

# 5 D Flip Flop

This section reviews the D type flip-flop (DFF) which:

- samples the input data on an edge of the clock (either falling or rising edge).
- D is the input of a DFF, and Q is the output. Some DFFs have an inverted output called $\overline{Q}$.
- DFFs can have `set` and `reset` pins that are used to clear the output. We will only implement the `reset` - this pin will always drive the output to 0 if activated (no matter what the input is).

**Complete the timing chart for DFF:**
Fill in the signal labeled Q in Figure 5. This flip flop is sensitive to the **rising `clk` edge**, and the `reset` **is synchronous** and **is active-high**.

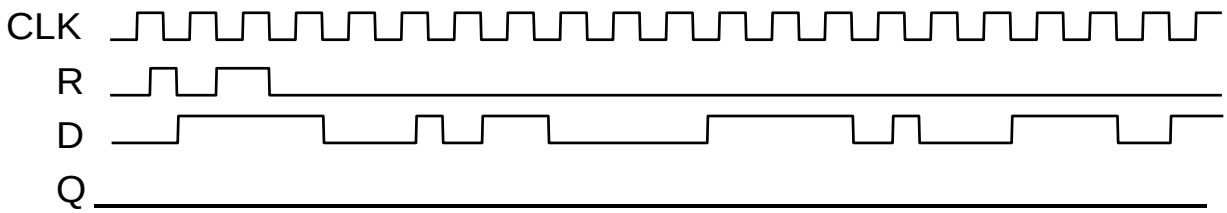


Figure 3: Timing diagram of a DFF operation.

**You are finished!**
Make sure you bring the pre-lab answers to your section. A TA will check this off for completion marks. **It would be beneficial for you to make yourself familiar with the lab instructions handout.**

# 6 Printable Timing Diagrams

You may wish to print this page for answering the timing questions.

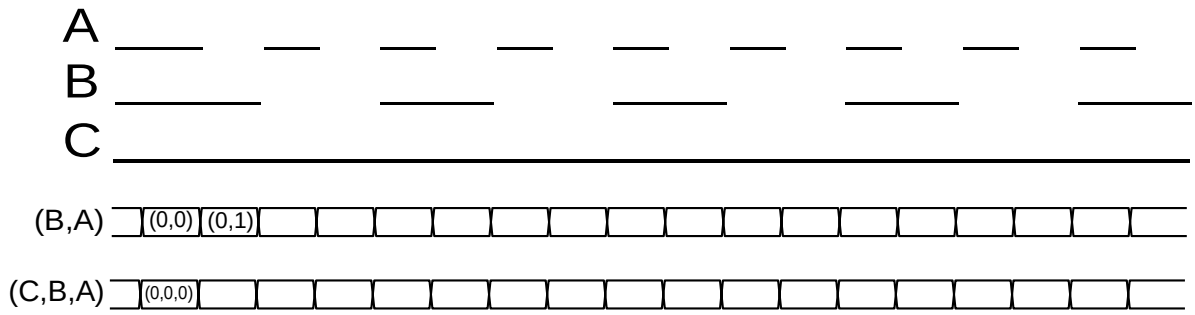


Figure 4: Timing diagram for Section 4

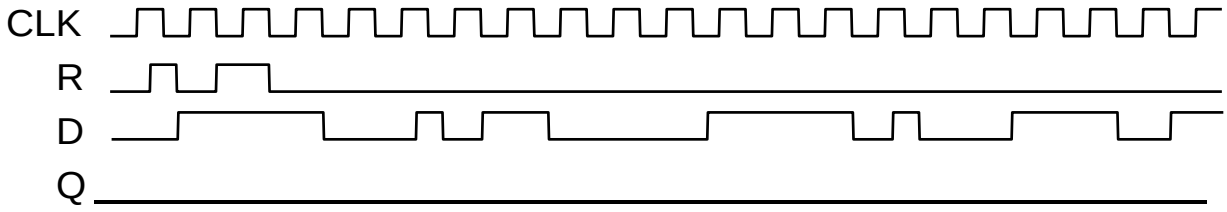


Figure 5: Timing diagram for Section 5.