Learning Objectives

1. Learning about clocks and clock dividers
2. Using the Clocking Wizard to instantiate a clock divider IP core
3. Displaying a binary number on two 7-segment displays at the same time

Work You Need To Do

1. This lab will not have a Homework part. It is all going to be done during the lab session.
2. Carefully go through this handout and make sure you read and follow all the instructions.
3. Perform an SVN up to load the FPGALab2 directory. The contents of the directory are listed below.
   - src directory: Contains provided source files. Also, use for uploading your source code.
4. To get credit for this lab, you need to show an instructor or TA that the design is functioning properly on the board. If you do this lab on your own, you need to come to an Honors office hour and show your results.

What You Should Submit

Make sure you submit these files before you leave lab:

1. All the Verilog modules: DecimalDigitDecoder.v, top_level.v, clock_divider.v, and the Clock Divider IP Core, to be submitted in the src directory.
2. The modified Basys3_Master.xdc file
3. The partners.txt file if you worked in pairs
Introduction

Our goal in this lab is to extend our binary to decimal decoder from Lab 1 to use two 7-segment displays at the same time. As you recall from Lab 1, since the Basys3 uses common cathodes (i.e., all the cathodes corresponding to the same segment, in the four displays, are connected to the same pin), we need a scanning circuit in order to utilize more than one display at the same time. The circuit should drive the anode signals and corresponding cathode patterns of each digit in a repeating, continuous succession, at an update rate that is faster than what the human eye can detect. In order for each of the digits to appear bright and continuously illuminated, all desired digits should be driven once every 1 to 16ms, for a refresh frequency of 1 KHz to 60 Hz. If the update or “refresh” rate is slowed to around 45 Hz, most people will begin to see the display flicker. If it is faster than 1 KHz, it will be too fast, and you won’t be able to distinguish the digits. Since the Basys3 is equipped with a 100 MHz clock, we will need to design a clock divider to generate a slower clock from it. We will start by talking about clocks, and explaining what clock dividers are, and how they are designed. To do that, we will use a combination of a Xilinx-provided Intellectual Property (IP) clock divider, and a manually-written, Verilog, clock divider. Once we have our desired clock, we will extend the design that we did in Lab 1 to output the tens digit on the second 7-segment display (instead of the LED). The following handout will guide you through the steps required to achieve our goal.

Part 1 Designing a Clock Divider

Intro to Clocks and Clock Dividers

Sequential circuits are a very important subset of digital circuits. They include Finite State Machines (FSMs), memories, and other circuits that form the building blocks of all modern digital systems. Since most sequential circuits are synchronous, a “clock” is required to synchronize the different parts of the circuit and run it at a certain desired frequency. Clocks are simple signals that oscillate between “High” and “Low”. In the common case, clocks are usually square waves, generated by an oscillator or clock generator, and have a certain period, referred to as “clock cycle”, which is the time between two successive “rising edges” or “falling edges”. They can have different duty cycles, which represents the percentage of time the clock is high during its period; most commonly, clocks have 50% duty cycles (they are high during half the period and low during the other half). Figure 1 shows a clock with a 50% duty cycle. Since clocks are usually generated off-chip, we have no control over the different characteristics of the generated clock, mainly frequency, and duty cycle. We can, however, manipulate the clock in our circuit to generate different versions of the clock with different frequencies and phase shifts. Assuming that a clock $C_1$ has a rising edge at time $t_0$, then, a clock $C_2$ that has it’s phase shifted relative to $C$, would have it’s rising edge at a time

![Figure 1. Clock “Anatomy”](image-url)
\( t_0 < t < t_0 + T \), where \( T \) represents the period of the main clock. Phase shifts are measured in degrees, such that a clock that has been shifted by \( \frac{T}{4} \) is said to have a 90° phase shift. Figure 2 shows 4 clocks that are phase shifted. \( C_1 \) has a period \( P \) and represents the main clock. \( D \) represents the phase shift between the clocks and is equal to \( \frac{P}{4} \). Compared to \( C_1 \), \( C_2 \) is shifted by 90°, \( C_3 \) is shifted by 180°, and \( C_4 \) is shifted by 270°.

![Figure 2. Phase Shifts](image)

Another important manipulation is changing frequency. This is very useful when we require a clock that is slower than the clock provided to us by the clock generator. The process of taking a faster clock, and generating a slower clock out it, is called clock division. Figure 3 shows an example of a divided clock. The main clock of 40MHz can be used to generate two slower clocks, one running at 20MHz, and one running at 10MHz. In fact, given any clock, you can generate any slower clock as long as it’s period is an even multiple of the original clock’s period. For example, by looking at Figure 3, one can see that if we represent the period of the first clock by \( T \), it follows that the period of the second clock is \( T_1 = 2T \), and that of the third clock is \( T_2 = 2T_1 = 4T \). Conversely, if \( f \) is the frequency of the first clock, then we have \( f_1 = \frac{f}{2} \) and \( f_2 = \frac{f}{4} \). Therefore, based on this observation, one can deduce that we can achieve a slower clock with a specific frequency \( f' \), by counting how many cycles of the main clock we need, to form a full cycle of the slower clock. The count that we need to perform is proportional to \( \frac{f}{f'} \).

![Figure 3. Clock Division](image)
Dividing the Basys3 Clock

The Basys3 board that we use has a built-in 100MHz clock connected to Pin W5. Since we need to use the clock for the scanning circuit of our 7-segment display (described in the Introduction), we will require the clock to have a frequency of 500Hz. In order to do so, we will design a two-tier clock divider. We will start by using the Vivado Clocking Wizard to automatically generate a 100MHz to 5MHz clock divider. Then, we will design our own Verilog clock divider to further divide the clock from 5MHz to 500Hz. The reason we can’t only use the Vivado Clocking Wizard to generate our desired clock is because the wizard doesn’t allow us to set the output frequency to less than 4.687MHz. The clocking wizard can be used to generate many different clocks with different phase shifts and frequencies, however we will only use it in our labs for the purpose of clock division.

1-1 Generating the Vivado Clock Divider IP

The following steps will guide you through the process of generating a 100MHz to 5MHz clock divider IP core using the Vivado Clocking Wizard.

1-1.1 In the Project Manager, click on the IP Catalog. Navigate to the Clocking subfolder under the FPGA Feature and Design folder, then double click on Clocking Wizard. When the wizard opens, you will notice that there are five tabs and a Component Name text box; set the name to clk_wiz.

1-1.2 The first tab, titled Clocking Options, has parameters related to input clock and clocking features, the input frequency value and range. Since the actual clock source is 100 MHz, we will keep the value to default.

1-1.3 The second tab, titled Output Clocks, has parameters which are related to the output clocks and desired frequencies. Change the Requested Output Frequency to 1.000 MHz and notice that it shows the frequency in red indicating something is wrong. Move the mouse over it and you will see a pop-up indicating that the actual frequency range for this device is 4.687 MHz to 800 MHz. Change it to 5.000 MHz, and disable the RESET input and LOCKED output.

1-1.4 The third tab, titled MMCM Settings, shows the calculated settings. Do not change anything in this tab.

1-1.5 The fourth tab, titled Port Renaming, allows you to change the port names. Change the input port to clk_in and the output port to clk_out.

1-1.6 Click OK and then click Generate to generate the files used for synthesis, implementation, and simulation.
1-2 Designing a Verilog Clock Divider

In the following section we will design our own Verilog clock divider. Since you haven’t written sequential Verilog before, we have provided you with a template of the clock divider code in Listing 1 below. Note that the counter value is left blank on purpose!

Listing 1. Clock Divider Verilog Module

```
module clock_divider(
    input clkin,
    output clkout);

reg [31:0] counter = 1;
reg temp_clk = 0;
always @(posedge clkin)
begin
    if (counter == . . . )
    begin
        counter <= 1;
        temp_clk <= ~temp_clk;
    end
    else
    counter <= counter+1;
end
assign clkout = temp_clk;
endmodule
```

We will start by creating a Vivado module for a 12Hz to 2Hz clock divider. We will simulate this, and once we are sure that it is working correctly we will modify it to be a 5MHz to 500Hz clock divider. The following steps will guide you through this process.

1-2.1 Create a new Design source in Vivado. Name it `clock_divider.v` and set it’s input and output to be `clkin` and `clkout` respectively. Fill in the code for the clock divider, and make sure you calculate the proper value for the counter. Remember that we want a 12Hz to 2Hz divider.

1-2.2 Add the `divider_tb.v` test bench and simulate it. View the waveform and verify that you have created a 2Hz clock.

1-2.3 Screenshot the simulation for SVN.

1-2.4 Modify the counter in the divider so that it becomes a 5MHz to 500Hz divider. Tip: Show your result to an instructor before you move on.
Part 2 Creating the Binary to Decimal Decoder

In this part, we will modify the BCD to 7-Segment Display Decoder that we implemented in Part 1. We should modify the `DecimalDigitDecoder` module to output a 4-bit BCD representation for the one's decimal place of the number (instead of the 1-bit that we were outputing to drive the LED). This 4-bit output will then be fed into a `BCDToLED` decoder to generate the corresponding bits required to display the number on the 7-segment display. We then need to use the clock that we generated to alternate between the two rightmost displays to display the ones and tens digit on them.

2-1 Modifying the Binary to Decimal Decoder

We will now add the Verilog files from Lab 1.

2-1.1 Add the `BCDToLED`, `DecimalDigitDecoder`, and `top_level` modules from Lab 1.

2-1.2 Modify the `DecimalDigitDecoder` module such that \( z \) is now 4-bits instead of 1-bit.

2-1.3 Simulate the modified `DecimalDigitDecoder` module using the `DecimalDigitDecoder_tb.v` file provided in your FPGALab2/src directory.

   Tip: Show an instructor your simulation. Make sure both output ports are expanded and a suitable radix is chosen for all inputs/outputs.

2-2 Modifying the Top Level Module

2-2.1 Modify the `top_level` module to include both clock dividers and perform the necessary modifications to connect both the one's and ten's digits to the 7-segment displays and enable the corresponding display accordingly. Remember, we are using the clock to alternate between the two displays. One way of doing this is by enabling one display when the clock is high, and enabling the other when the clock is low. It also has to take the Basys 3 clock as an input. *(Note: Remember, the clock is connected to pin W5).*

2-2.2 Add the `Basys3_Master.xdc` and modify it accordingly.

2-2.3 Synthesize the design.

2-2.4 Implement the design.

2-2.5 Generate the bitstream, download it to the Basys3 board, and check the functionality.

   Have an Honors instructor check that your implemented design is functioning correctly to get credit for this lab.

Please don’t forget to upload everything to SVN!