RUTGERS
School of Engineering

Embedded Systems Design

14 : 332 : 493 : 03 / 16 : 332 : 579 : 05

Lab 1: "Clocks, Counters & Buttons"

1 Introduction

In this experiment, we will formally introduce one of the core building blocks of digital circuits: the humble counter. By exploiting the versatility of the counter, a high frequency input clock will be slowed down by several orders of magnitude and then used to drive a bidirectional counter. Additionally, a counter will be used to solve one of the core problems involved with getting manual user input in digital circuits.

How Does a Counter Work?

Inspect the design located at: http://www.edaplayground.com/x/ieF
A simulation picture is provided on canvas.
Read this article: https://www.doulos.com/knowhow/vhdl/vhdl-vector-arithmetic-using-numeric_std
Watch this video on clocks in FPGAs: https://www.youtube.com/watch?v=htwlb-DuEK8

Notice that no structure is specified in the counter design. The data is described simply as signals and operations on signals. This is called "behavioral" modeling and is the most common and powerful way of designing digital circuits. An experienced digital designer usually knows how their behavioral code will translate into hardware before they write it (conditionals become multiplexers, signals operated on at clock edges become flops, etc).

In the physical world you could use a chip like: http://www.ti.com/lit/ds/symlink/sn74f163a.pdf

The alternative strategy would be to use toggle flip flops designed with discrete master-slave latch pairs and instantiate 4 flops. This is a lot of work and defeats the purpose of using VHDL to design at a higher level of abstraction. That type of "structural" design is used for top level designs and test benches, not individual components usually.

1 My Clock is Just Right

1.1 Background

The Digilent Zybo contains a single crystal oscillator that runs at a frequency of 50 MHz. This goes into an Ethernet PHY which generates a 125 MHz clock to feed into the FPGA fabric.

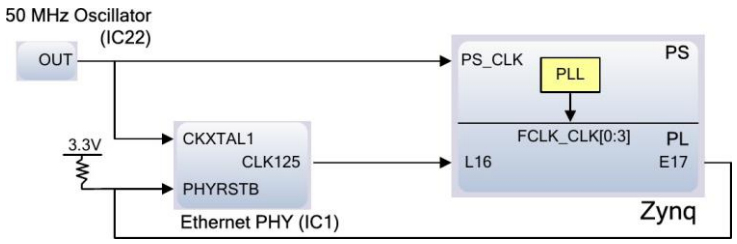

Figure 1.1: Zybo Clock System Schematic [2]

In this first section, we are going to divide that clock down to a much more human-friendly frequency in order to observe its changes by redirecting the then-divided clock to an LED.

In order to be more human-friendly, we are going to divide the frequency of the clock from 125 MHz down to 2 Hz (a long way indeed!). In order to do so, we are going to utilize a simple counter circuit.

Question 1.1: How much do we need to divide our input by to get from 125 MHz to 2 Hz?

In order to divide a clock using a counter, we create a simple circuit. It contains a single binary up counter. First, we initialize an n-bit signal to **0**. Then, for every rising edge of the input clock, we increment that signal. Once the signal reaches the division ratio, we reset the signal to **0**. Concurrently, when the signal is equal to the division ratio we set an output signal to **1**. Using this output signal as a clock enable, we only pass through every "ith" pulse, where "i" is the division ratio. In this way, we can pass along that clock enable to other designs' clock enable pins on their flops to achieve a design that runs at the divided frequency.

Question 1.2: How many bits are required to store a counter that can count up to the value obtained in Q1.1?

1.2 Tasks

- Create a design called "clock_div" in VHDL that takes as input a 125 MHz clock signal and divides it down to generate an enable signal every 2 Hz. Utilize a looping testbench to drive 125 million clock cycles on the input of the module and verify that the testbench output shows two full clock cycles. See top level block diagram in Figure 1.2. Modify the Zybo_Master.xdc file to un-comment

the clk and led[0] pins. Please note that square brackets in an XDC file are used to index a vector. Since only one led is currently being used, it is advised that you remove the brackets from led[0].

Design Hint: If designed correctly, the output pulse generated by clock_div should have width equal to that of one 125 MHz clock period.

- Create a top-level design called "divider_top". Instantiate clock_div and use its output (div) to control a signal inversion process. When div is high and the clock is on it's rising edge, invert a signal whose output drives led0. If done correctly, the resulting elaboration schematic of divider_top should be similar to that of figure 1.3, where div drives the enable port of a DFF, whose input is the inversion of its output. Visually confirm the 2 Hz operation speed by loading it onto the Zybo.

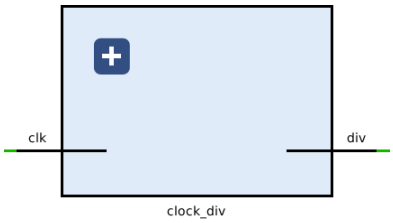

Figure 1.2: clk_div block diagram

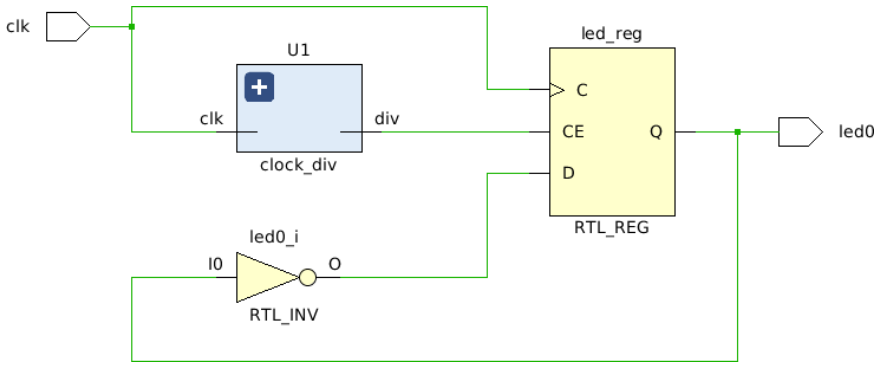

Figure 1.3: divider_top block diagram

1.3 Expert Bonus Challenge (+ 0.5 pts extra credit)

By taking advantage of built in arithmetic libraries and generics, create a clock_div entity that can be used to divide any input frequency to any possible output frequency achievable through integer division.

Hints:

- You will need to calculate the log2 of the division integer in order to find the bit width of your count signal.
- You will probably need to use conversion functions and make your own custom log2 function. See here for reference: https://nandland.com/function-3/
- The entity will have two generics (input frequency and division integer).

2 Bouncy Buttons

2.1 Background

Push buttons are fickle creatures. Due to being mechanical in nature, they are often the bane of existence for digital designers when not dealt with correctly. The problem lies in that mechanical nature. When the button is pushed or released the spring inside causes the contacts to behave like a damped oscillator, which creates spikes in the signal.

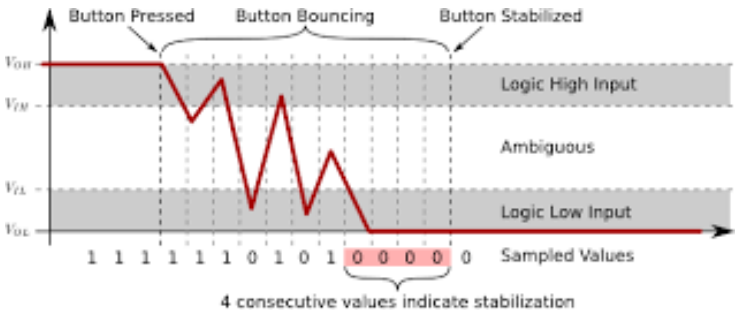

Figure 1.4: Sample Button Bounce Waveform [1]

In order to avoid this bouncy nature we have two options as engineers: analog or digital de-bouncing. In the analog world, this would be done by using an RC charge/discharge circuit whose time constant is larger than the amount of bounce time. However, we don't have the luxury of analog circuitry so we are going to use a digital method.

In order to digitally de-bounce the input, we use a simple counter circuit that has an enable and an inverted reset. We sample the button value with respect to a system clock (our 125 MHz clock from the Ethernet PHY) into a 2 bit long shift register [in order to avoid meta-stable state in between $V\,IH_{min}$ and $V\,IL_{max}$] (std logic vector where each clock bit 0 gets new value and bit 1 gets value at bit 0) and see if the value of the end of the shift register (bit 1 of the vector) is a **1**. If it is, our counter increases by one. Any time our sampled value differs from a **1**, we reset our counter. Once the counter hits its maximum value (the number of successive samples we deem enough), the output of the debounce circuit displays a **1** and the counter will no longer increment. At any time the counter has not hit its maximum value, the output of the debounce circuit displays a **0**.

Note: Due to how the circuit adds based on samples of '1' until a certain number is reached, this circuit is very similar to the aforementioned RC analog version.

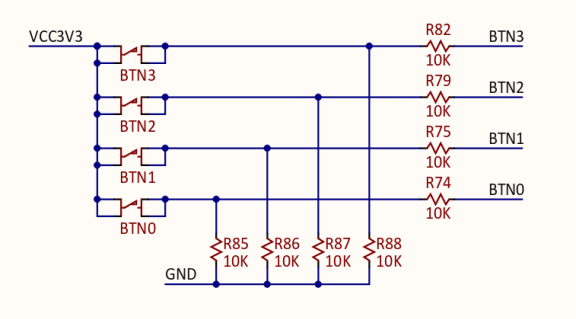

Figure 1.5: Zybo Push-button Circuit Diagram [3]

Question 2.1: What is the value of the button when it is pressed for the Zybo?

Question 2.2 (optional): If it were the other value when pressed, would we have to alter our debounce design? Why or why not?

Question 2.3: If we want our debounce time to be 20 ms, and our system clock is 125 MHz, how many ticks do we need a steady '1' to be read for it to count as a '1'?)

Question 2.4: How many bits are required for a counter that can go that high?

2.2 Task

Create a design called debounce in VHDL that takes as input a single bit signal from a push button and de-bounces it in order to avoid undesirable mechanical behavior. Create a simple testbench with a clock, "button", and output that shows the de-bouncing behavior of the circuit. Use your answers from the last section as a reference for your implementation.

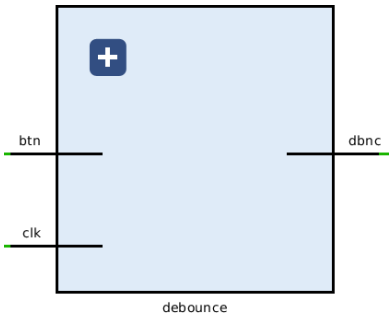

Figure 1.6: debounce block diagram

Signal Name	clk	btn	dbnc
Direction	input	input	output
Connected To	oscillator	button	nothing yet

Table 1.1: debounce signal summary

3 Actually Using a Counter to Count

3.1 Background

A counter needs little introduction, especially since we are now familiar with some of its other uses. In this section, we are going to create a normal bidirectional counter with a couple extra control signals.

3.2 Task

Design a fully synchronous 4-bit counter called fancy_counter (Figure 1.7) with the following behavior:

- On the clock rising edge, unless en is **1**, nothing will change in the circuit.
- Even if en is **1**, if clk_en is **0** nothing can change the circuit except rst
- When rst is asserted the cnt value will become **0**.
- It can count either up or down depending on the value of a "direction" register, which is updated at the clock rising edge with the value present at dir when updn is **1**.
- On the clock rising edge, if ld is **1**, the value present at val will be loaded into the "value" register.
- If counting up, it will count until the number in a 4-bit "value" register has been reached, at which point it will roll over to **0000**. If counting down, it will go from **0000** to value when it underflows.

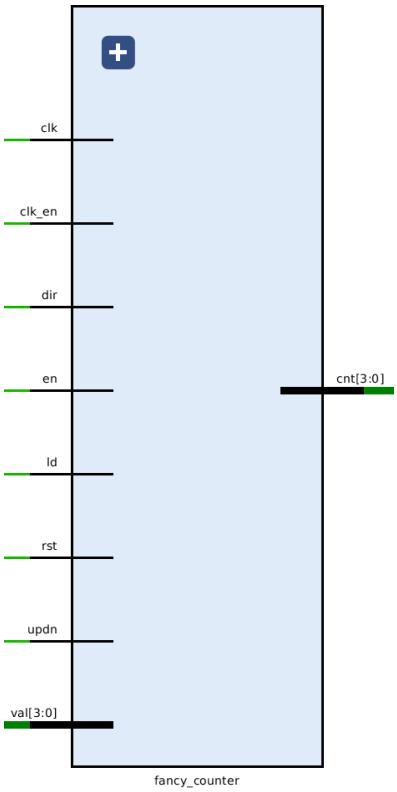

Figure 1.7: fancy_counter block diagram

Hint/Clarification: 'ld' loads in the value seen on the 'val' pin, while 'updn' loads in the value present on the 'dir' pin. They can act independently of one another.

3.3 Expert Bonus Challenge (+ 0.5 pts extra credit)

Create a testbench that demonstrates the direction change, reset, value update, and enable control of fancy counter. Clearly show each behavior specified.

4 Bringing it All Together

4.1 Background

Now that we have all the components assembled and tested, we are going to combine them in a structural top level design to do something useful. Because all of the components have been individually tested, we can say with reasonable confidence that the system will behave as expected.

4.2 Task

Create a top level design following the diagram shown in the figure below called counter_top. Modify the Zybo_Master.xdc file to enable the appropriate pins. Load the design onto the Zybo and demonstrate the correct performance of the final design.

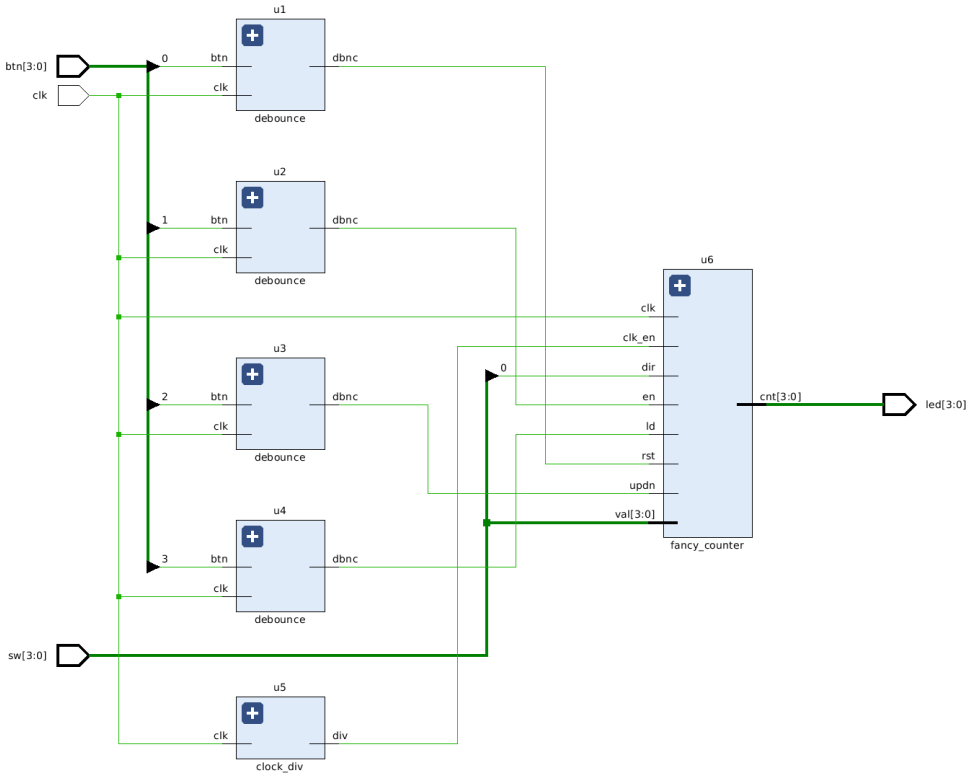

Figure 1.8: counter_top block diagram

5 Credits

Sources

[1] *Tap-Tempo Metronome - Button Bounce*. url: https://www.wayneandlayne.com/files/metronome/images/button_bounce.png.

[2] *ZyBo Reference Manual*. Feb. 27, 2017. url: https://reference.digilentinc.com/reference/programmable-logic/zybo/reference-manual

[3] *ZyBo Schematic*. May 7, 2015. url: https://reference.digilentinc.com/_media/zybo:zybo_sch.pdf.

Acknowledgments

Phil Southard, Milton Diaz
Part Time Lecturers

Gregory Leonberg
Author, Fall 2017

Steve DiNicolantonio
Updates/Modifications, Fall 2018