Embedded Systems

Lab Report #1

Lawrence Lenoci

Wesley Lui

10/09/18

**Purpose:**

The main purpose of this lab is to explore one of the core components of the digital circuits we discuss in class and lab: the counter. Through the use of counters in this lab we intend to slow a high frequency clock down a few orders of magnitude and then proceed to apply it to a bidirectional counter. Additionally we will be using a counter to solve a “core problem” which involves getting a user's manual input to a digital circuit through mechanical buttons. To finish it off we are going to put all our designs together to produce a binary counter that can be initialized and modified through user input via the ZYBO switches and buttons.

**Part 1 - My Clock is Just Right**

Theory of Operation:

The main theory behind a clock divider is based around the use of a counter. It starts by creating a circuit that contains a single counter that counts 1 bit at a time which we initialized to 0. As we proceed we analyze our 125MHz clock and for every rising edge of the clock we increment the counter. Upon the counter signal reaching our predetermined count in this case, n = 31250000, we use the code to set our output “div” to 1. So in this situation our output signal only outputs high every “ith” pulse of our input clock in which case out “i” is our division ratio. Through this code we can pass a clock to other designs “clock nable pins” o that we can achieve a design running at a desired clock. So for this case we have taken our 125MHz clock and used a counter to count to 31250000 at which point we output a high clock pulse to get our desired 2Hz clock.

Design:

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.numeric\_std.ALL;

entity clock\_div is -- define entity

port (

clk,reset: in std\_logic; -- define inputs clk and reset

div: out std\_logic); -- define output div

end clock\_div;

architecture behavior of clock\_div is -- create architecture

signal cnt: integer:=1; --create signal for count

signal clock\_out : std\_logic := '0'; --create signal for output clock

begin

process(clk,reset) -- Process with if statements to increment counter

begin

if(reset='1') then

cnt<=1;

clock\_out<='0';

elsif(rising\_edge(clk)) then

cnt <=cnt+1;

if (cnt = 31250000) then -- count-31250000 (125MHz/Desired Clock)/2

clock\_out <= NOT clock\_out;

cnt <= 1;

end if;

end if;

div <= clock\_out;

end process;

end behavior;

Test

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

ENTITY clock\_div\_tb IS

END clock\_div\_tb;

ARCHITECTURE behavior OF clock\_div\_tb IS

COMPONENT clock\_div

PORT(

clk : IN std\_logic;

reset : IN std\_logic;

div : OUT std\_logic

);

END COMPONENT;

--Inputs

signal clk : std\_logic := '0';

signal reset : std\_logic := '0';

--Outputs

signal div : std\_logic;

-- Clock period definitions

constant clk\_period : time := 8 ns;

BEGIN

-- Instantiate the Unit Under Test (UUT)

uut: clock\_div PORT MAP (

clk => clk,

reset => reset,

div => div

);

-- Clock process definitions

clk\_process :process

begin

clk <= '0';

wait for clk\_period/2;

clk <= '1';

wait for clk\_period/2;

end process;

-- Stimulus process

stim\_proc: process

begin

wait for 100 ns;

reset <= '1';

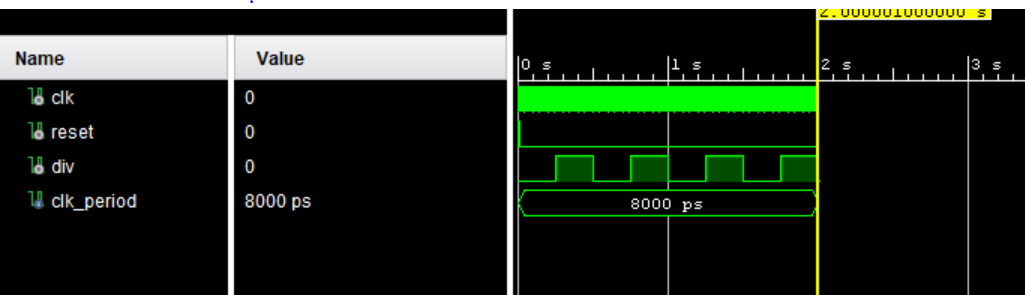
wait for 100 ns;

reset <= '0';

wait;

end process;

END;



Implementation:

Vivado Elaboration Schematic:

Vivado Synthesis Schematic:

**Part 2 - Bouncy Buttons**

Theory of Operation:

The main purpose of this part of the lab is to try and tame the “fickle” nature of a mechanical button. They can be an issue for digital designer when it comes to executing designs if they're not handled properly. The main issue with these buttons is that they are “mechanical.” This means when you press one there are contacts and a spring to hold the button in one position or another. For example the spring holds the button out and when pressed in and released the spring inside cause oscillation of the contacts which results in an unsteady signal with spikes of high/low. So for this part we intent to write a piece of code that will utilize counters to take into account how long a button is pressed and how long it reads high/low signals. From this we set a boundary where if, for example, it reads a high for 20ms it can assume the button has been pressed and will output high then as the switch bounces the counter never gets to the predetermined count required for high so it ignores the spikes in the signal from the button because they were not “long” enough for it to let the code output high.

**Part 3 Actually using a counter to count**

Theory of operation:

For this part of lab our intended goal is to create a “normal bidirectional counter” that utilizes “extra control signals.” In other words we are creating a device that has multiple inputs all of which result in a counter that is capable of counting up to “4 bits” or 1111 from 0000 or vise versa. Also we want to be able to initialize a value to start counting up or down from if desired.

**Part 4 Bringing it all together**

Theory of Operation:

For the final part of the lab we are bringing together everything we have done previously in one design, literally. We are implementing a code that will basically create a bunch of “components” that contain the code from the three prior sections and put them together. The result of this from what I can see in the schematic will be a binary counter. It is going to use the ZYBO board switches to initiate a value if desired and will use the buttons to start, reset and load the values and the counter. The output of the design will utilize 4 LED’s to represent the individual bits of a 4 bit counter. It will be able to count both up and down and roll over or underflow if it maximizes or hits a minimum. But for the most part in simple terms the design will either take a starting value in terms of the bits ‘0000,’ or initialize the count at 0 and go from there blinking in the same way a binary clock does in our case at 2Hz from our clock divider.

**Discussion:**

**My Clock is Just Right**

Questions:

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

To get a resulting clock of 2Hz we needed to divide out 125 MHz clock by 31250000

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

To store a counter that can handle the value 3125000 we need 25 bits.

Observations:

In this I observed that it is important to note that for example when trying to get a 2Hz clock from 125MHz you need to not only divide the number 2Hz to acquire the number you need to use in the counter but divide it in half since you are looking for the rising edge of the clock in your counter. Dividing by simply 2Hz in this case results in an output divided clock of 4Hz which is obviously not 2Hz. This was one of my initial mistakes when writing my code for this part of the lab.

Questions/Follow-Up:

Upon completion of this part of the lab there really isn’t anything I would say I am confused about as I understand pretty well how the counter is working and the loops I used to actually produce the desired result.

**Bouncy Buttons**

Questions:

2.1 What is the value of the button when it is pressed for the ZyBo?

The button outputs high when it is pressed or in other words 1 or VCC = 3V.

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

Yes, the structure is the same, however we have to be counting after a falling edge rather than a rising edge on the output of the button to decide if the button is actually pressed or just “bouncing.”

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’ ?)

Well in this case since 20ms converts to a clock of 50Hz to be considered a steady 1. This would be 125000 ticks of the counter.

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

21 Bits are required for this specific counter.

Observations:

**Actually using a counter to count**

Observations

**Bring it all together**

Observations: