Embedded Systems Spring 2019
Lab 1

Joseph Shenouda

February 28th 2019

Purpose

The purpose of this lab was to work with the Xilinix 7 series based Zybo board and use it understand how to manipulate the clock on an FPGA. In this lab we first learned how to divide the clock down to be interpreted by human eyes from 125MHz to 2Hz. Then we learned how to debounce our buttons to get around imperfections in the mechanics that could create unintended signals. Then we created a sophisticated counter with resets load and bidirectional counting. Finally we put everything together in a structural design and loaded it onto the Zybo for testing.

1 My Clock Just Right

In this part of the lab we were tasked with taking in the 125MHz clock generated by the Zybo board and bringing it down to 2Hz frequency so that we could observe an LED being turned on an off by the clock. The circuit was intended to behave such that upon programming the Zybo we would see led0 light up twice every second.

Question 1.1 How much do we need to divide our input by to get from 125MHz to 2Hz?
To get from 125MHz to 2Hz we needed to divide our input by 62500000.

Question 1.2 How many bit are required to store a counter that can count up to the value found in Q1.1?
The amount of bits needed to represent the number 62500000 is 26 bits.

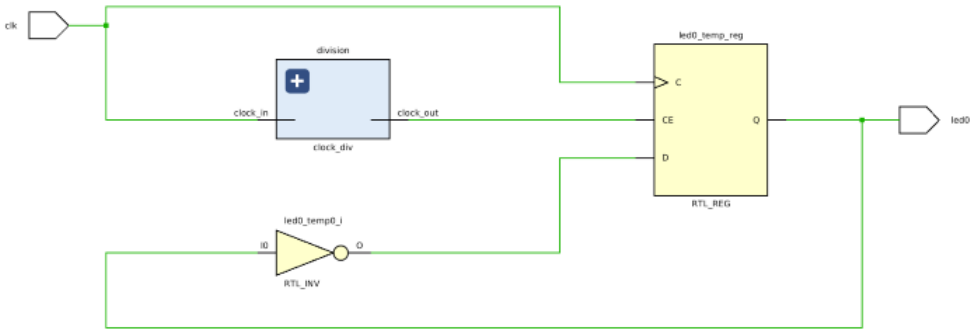

Figure 1: RTL Schematic for Part 1

VHDL Design

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;


entity clock_div is
    Port( signal clock_in : in std_logic;
          signal clock_out : out std_logic);
end clock_div;

architecture Behavioral of clock_div is
signal count : std_logic_vector(25 downto 0) := (others => '0');

begin

process(clock_in)
    begin
        if rising_edge(clock_in) then
            count <= std_logic_vector(unsigned(count)+1);
            if count = "11101110011010110010100000" then
                clock_out <= '1';
                count <= (others => '0');
            else
                clock_out <= '0';
            end if;
        end if;
 end process;
end Behavioral;
```

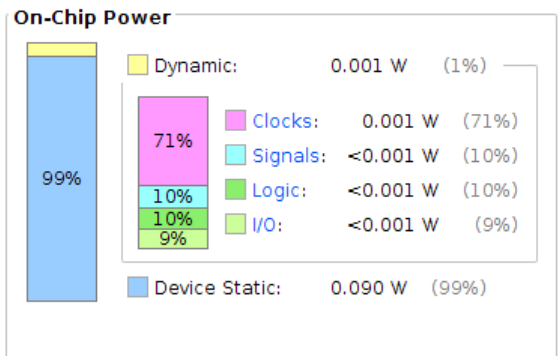

Figure 2: Power Graph for Part 1

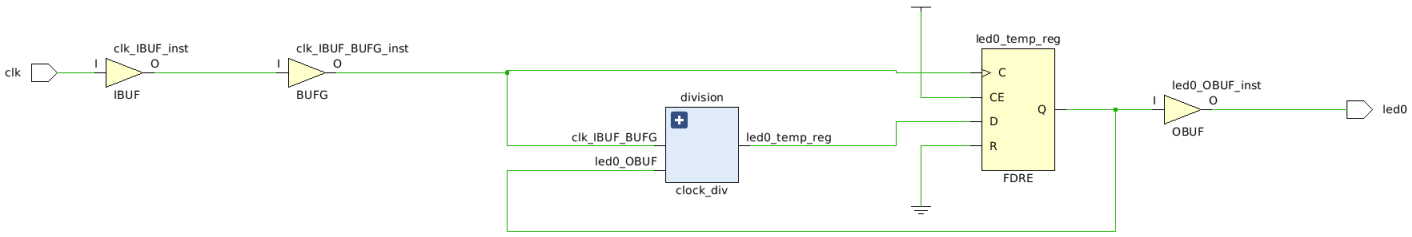

Figure 3: Synthesis Schematic for Part 1

Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	10	17600	0.06
FF	28	35200	0.08
IO	2	100	2.00
BUFG	1	32	3.13

Figure 4: Utilization Table for Part 1

XDC File

```
## This file is a general .xdc for the ZYBO Rev B board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used signals according to the project


##Clock signal
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L11P_T1_SRCC_35 Sch=sysc
create_clock -add -name sys_clk_pin -period 8.00 -waveform {0 4} [get_ports { clk }];


##Switches
#set_property -dict { PACKAGE_PIN G15   IOSTANDARD LVCMOS33 } [get_ports { sw[0] }]; #IO_L19N_T3_VREF_35 Sch=
#set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports { sw[1] }];  #IO_L24P_T3_34 Sch=SW1
#set_property -dict { PACKAGE_PIN W13   IOSTANDARD LVCMOS33 } [get_ports { sw[2] }]; #IO_L4N_T0_34 Sch=SW2
#set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports { sw[3] }]; #IO_L9P_T1_DQS_34 Sch=SW


##LEDs
set_property -dict { PACKAGE_PIN M14   IOSTANDARD LVCMOS33 } [get_ports { led0 }]; #IO_L23P_T3_35 Sch=LED0
#set_property -dict { PACKAGE_PIN M15   IOSTANDARD LVCMOS33 } [get_ports { led[1] }]; #IO_L23N_T3_35 Sch=LED1
#set_property -dict { PACKAGE_PIN G14   IOSTANDARD LVCMOS33 } [get_ports { led[2] }]; #IO_0_35=Sch=LED2
#set_property -dict { PACKAGE_PIN D18   IOSTANDARD LVCMOS33 } [get_ports { led[3] }]; #IO_L3N_T0_DQS_AD1N_35 

##Buttons
#set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { btn[0] }]; #IO_L20N_T3_34 Sch=BTN0
#set_property -dict { PACKAGE_PIN P16   IOSTANDARD LVCMOS33 } [get_ports { btn[1] }]; #IO_L24N_T3_34 Sch=BTN1
#set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { btn[2] }]; #IO_L18P_T2_34 Sch=BTN2
#set_property -dict { PACKAGE_PIN Y16   IOSTANDARD LVCMOS33 } [get_ports { btn[3] }]; #IO_L7P_T1_34 Sch=BTN3
```

In this constraint file the only thing we kept uncommented was the clock, and the first led (led0)

2 Bouncy Buttons

In the second part of the lab we attempted to solve the problem of mechanical buttons because then pressed they often give us signals that were unintended because of the spring that causes them to jump up and down momentarily. So to eliminate this problem we utilized a 2 bit shift register and a counter to count how a specific number of 1s received before confirming that the button was indeed pushed.

Question 2.1: What is the value of a button when it is pressed on the Zybo?
When the button is pressed the value of the button is a digital 1 based on the schematic.

Question 2.3: If we want our debounce to be 20ms, and our system clock is 125MHz, how many ticks do we need a steady '1' to be read for it to count as a '1'?
We need 2500000 ticks for a steady '1' because our system clock samples 125M times per second so if we want a debounce of 20ms we multiply these two values to get 2500000.

Question 2.4: How many bits are required for a counter that can go that high?
We need 22 bits to represent 2500000.

VHDL Design

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;


entity debounce is
    Port( signal clk : in std_logic;
          signal btn : in std_logic;
          signal dbnc : out std_logic);
end debounce;

architecture Behavioral of debounce is

signal counter : std_logic_vector(21 downto 0) := (others => '0');
signal shift_register :  std_logic_vector(1 downto 0) := (others => '0');
signal temp_shift : std_logic;

begin

process(clk)
begin
    if(rising_edge(clk)) then
        temp_shift <= shift_register(0);
        shift_register(0) <= btn;
        shift_register(1) <= temp_shift;
        if shift_register(1) = '1' then
            counter <= std_logic_vector(unsigned(counter)+1);
            if unsigned(counter) = 2500000 then
                dbnc <= '1';
            end if;
        else
            counter <= (others => '0');
            dbnc <= '0';
        end if;
    end if;
end process;
end Behavioral;
```

VHDL Testbench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity debounce_tb is
end debounce_tb;

architecture Behavioral of debounce_tb is

signal button : std_logic;
signal clock : std_logic;
signal debounce_out : std_logic;

component debounce is
port ( clk : in std_logic;
       btn : in std_logic;
       dbnc : out std_logic);
end component;


begin
debouncer : debounce port map (clk => clock, btn => button, dbnc => debounce_out);


process
    begin
        clock <= '1';
        wait for 4ns;
        clock <= '0';
        wait for 4ns;
end process;

process
    begin
        button <= '1';
        wait for 10ms;
        button <= '0';
        wait for 10ms;
        button <= '1';
        wait for 40ms;
end process;

end Behavioral;
```

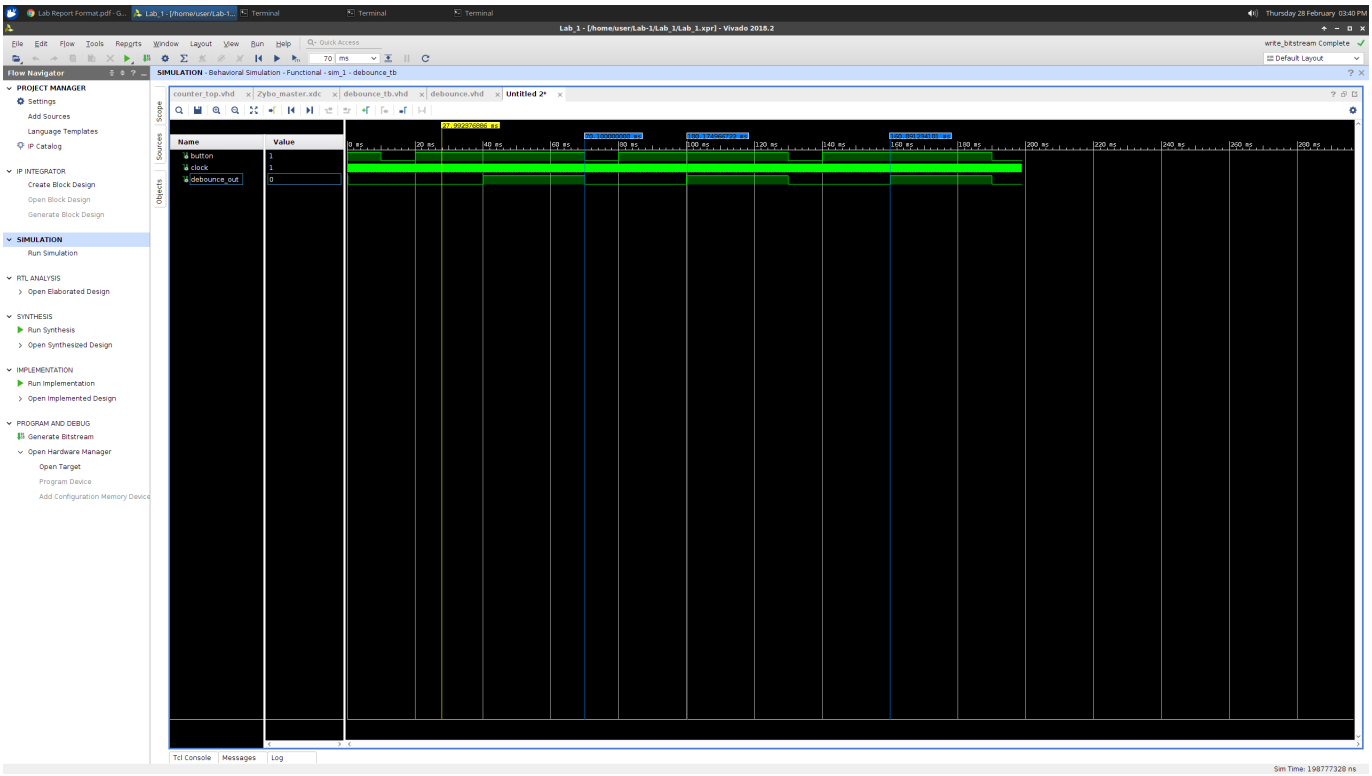

Figure 5: Simulation Screenshot for debouncer: From this screenshot we can see that the output of the debounce only produces a 1 when the button input is '1' for at least 20ms.

3 Actually Using a Counter to Count

Theory

For this part of the lab we created a counter that could do many different operations. Firstly it was able to count bidirectionally meaning it could count up and down. Secondly the counter had many inputs like reset and load to load values into the counter and reset to reset the counter back to its original position. The counter only counts up to the value in the value register then loops back to 0000. Similarly the counter only counts down to 0000 and then resets to the position in the value register.

VHDL Design

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;


entity fancy_counter is
    Port ( signal clk : in std_logic;
           signal clk_en : in std_logic;
           signal dir : in std_logic;
           signal en : in std_logic;
           signal ld : in std_logic;
           signal rst : in std_logic;
           signal updn : in std_logic;
           signal val : in std_logic_vector(3 downto 0);
           signal cnt : out std_logic_vector(3 downto 0) := (others => '0'));
end fancy_counter;

architecture Behavioral of fancy_counter is

signal valueRegistered : std_logic_vector(3 downto 0) := "1111";
signal dirPrev : std_logic;
```

```
signal counter : std_logic_vector(3 downto 0):="0000";


begin


process(clk)
begin
dirPrev <= dir;
if(rising_edge(clk) ) then
-- enable must be one for anything to happen
    if en = '1' then
    --The only operation that can occur when enable set to 1 is reset
        if rst = '1' then
            counter <= (others => '0');
        end if;
        --Perform operations when clk enable is 1
        if clk_en = '1' then
            if ld ='1' then
                valueRegistered <= val;
             end if;
             if updn = '1' then
                if dir = '1' then
                    if counter /= valueRegistered then
                        counter <=  std_logic_vector(unsigned(counter)+1);
                    else
                        counter <= "0000";
                    end if;
                    dirPrev <= dir;
                else
                    if counter /= "0000" then
                        counter <= std_logic_vector(unsigned(counter)-1);
                    else
                        counter <= valueRegistered;
                    end if;
                    dirPrev <= dir;
                end if;
             else
                if dirPrev = '1' then
                      if counter /= valueRegistered then
                        counter <= std_logic_vector(unsigned(counter)+1);
                      else
                        counter <= "0000";
                      end if;
                else
                     if counter /= "0000" then
                        counter <= std_logic_vector(unsigned(counter)-1);
                     else
                        counter <= valueRegistered;
                     end if;
                end if;
             end if;
         end if;
      end if;
end if;
end process;

cnt <= counter;

end Behavioral;
```

4 Bringing it all together

Theory

In this part of the lab we intend to bring together all of the little pieces we have been building up. The intent is to utilize the fancy counter with the debouncer to solve the mechanical issues of button presses and with the clock divider so that we can see the the counter changing with a frequency of 2Hz.

VHDL Design

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity counter_top is
    Port( signal clk : in std_logic;
          signal btn : in std_logic_vector(3 downto 0);
          signal sw : in std_logic_vector(3 downto 0);
          signal led : out std_logic_vector(3 downto 0)
          );
end counter_top;

architecture Structural of counter_top is

signal dbnc0 : std_logic;
signal dbnc1 : std_logic;
signal dbnc2 : std_logic;
signal dbnc3 : std_logic;

signal div_out : std_logic;



component debounce is
    port( clk : in std_logic;
          btn : in std_logic;
          dbnc : out std_logic
          );
end component;

component clock_div is
    port( clock_in : in std_logic;
          clock_out : out std_logic
          );
end component;

component fancy_counter is
    port(  clk : in std_logic;
           clk_en : in std_logic;
           dir : in std_logic;
           en : in std_logic;
           ld : in std_logic;
           rst : in std_logic;
           updn : in std_logic;
           val : in std_logic_vector(3 downto 0);
           cnt : out std_logic_vector(3 downto 0));
end component;


begin
```

```
u1 : debounce port map(clk => clk, btn => btn(0), dbnc => dbnc0);
u2 : debounce port map(clk => clk, btn => btn(1), dbnc => dbnc1);
u3 : debounce port map(clk => clk, btn => btn(2), dbnc => dbnc2);
u4 : debounce port map(clk => clk, btn => btn(3), dbnc => dbnc3);

u5 : clock_div port map(clock_in => clk , clock_out => div_out);

u6 : fancy_counter
    port map(clk => clk ,
             clk_en => div_out,
             dir => sw(0),
             en => dbnc1,
             ld => dbnc3,
             rst => dbnc0,
             updn => dbnc2,
             val => sw,
             cnt => led);

end Structural;
```

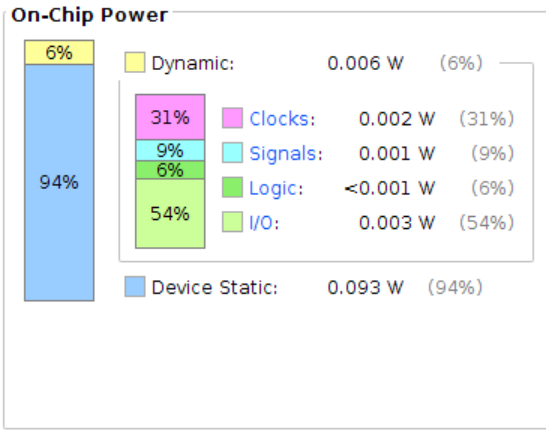

Figure 6: Power Graph Part 4

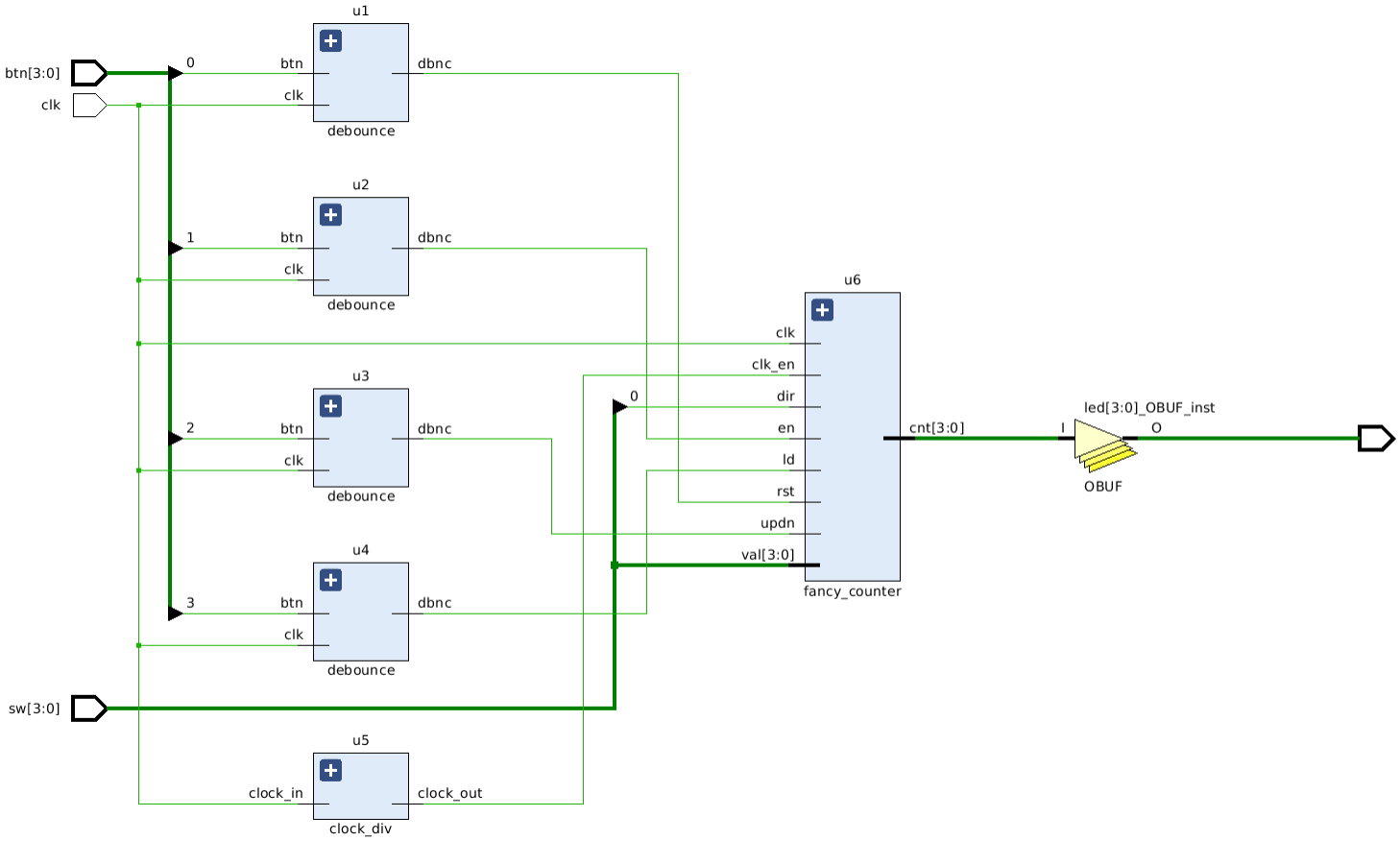

Figure 7: RTL Schematic for Part 4

Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	51	17600	0.29
FF	140	35200	0.40
IO	13	100	13.00
BUFG	1	32	3.13

Figure 8: Utilization Table for Part 4

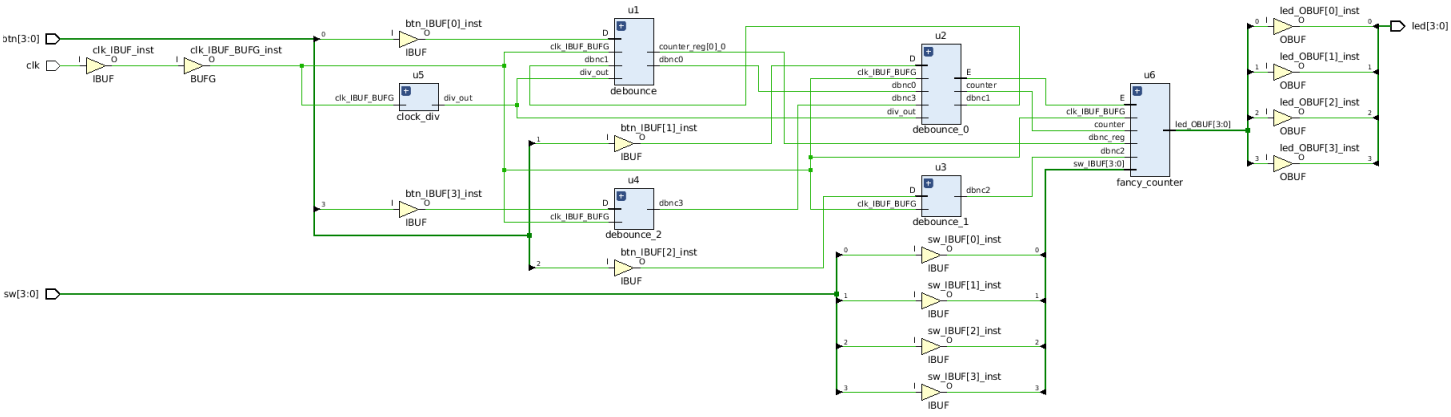

Figure 9: Synthesis Schematic for Part 4

Discussion

In this lab I learned a lot about how the clock in the FPGA works as well as how to use the different numeric types in my design for counting and other processing. I also learned how to use the clock that came in the board an manipulate it towards whatever process I need without directly messing with the clock tree structure in the FPGA. I also learned how to properly use current outputs of your design to determine the next output by using temporary variables.