Lab 1 – Clocks, Counters, and Buttons

Embedded systems
Lab Report 1
Anthony Lau: 153001126
February 28, 2019

Purpose:

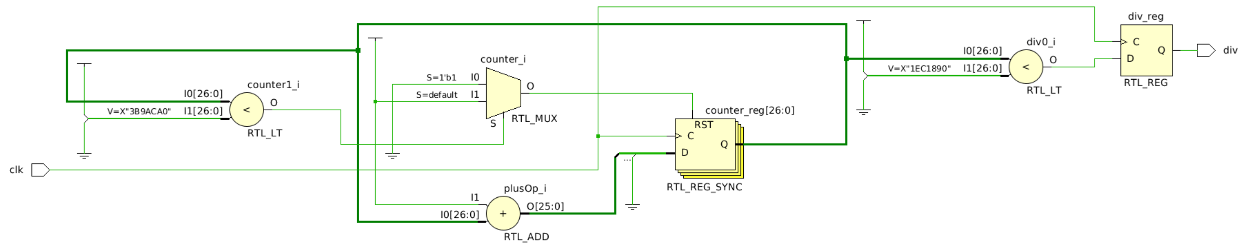
One of the most basic and core building blocks of digital circuits is the counter. In order to create a working counter, we need to deal with the high frequency input clock that a FPGA has. We need to slow it down in order to drive the counter that we desire. Additionally, manual input from a user will be able to set the circuit. When creating this counter, behavioral modeling will be implemented to achieve our desired results.

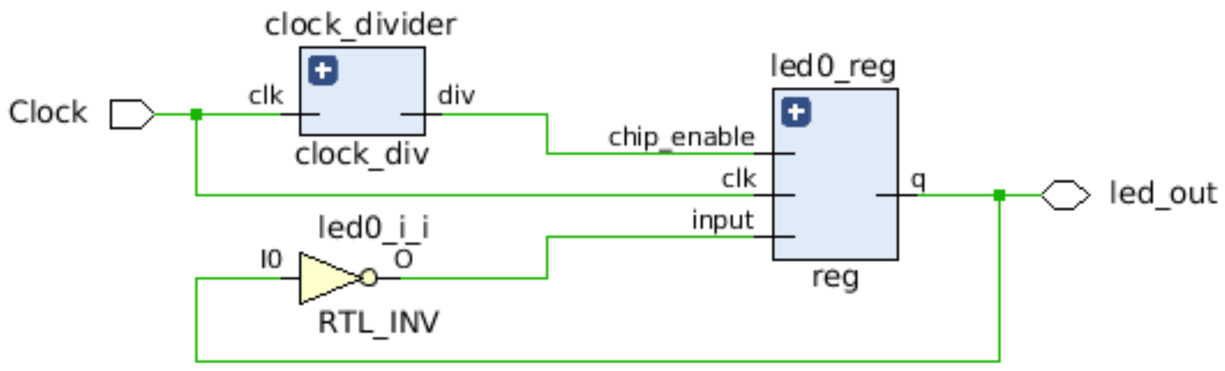
1. My Clock Is Just Right

Theory:

In order to implement a clock divider, we need a counter that counts up to our specified division ratio on the rising edge of the clock. Once the counter has reached that value, we reset it to 0 and send an output signal of 1. With this, we can use this output signal as a slower clock to any input we desire.

Schematic Diagram:





Design:

Clock Divider Code

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

entity clock_div is
    Port (clk: in std_logic;
          div: out std_logic);
end clock_div;

architecture Behavioral of clock_div is

signal counter  : std_logic_vector(26 downto 0) := (others => '0');
begin
    process(clk)
        begin
            if rising_edge(clk) then
                if (unsigned(counter) < 62500000) then
                    counter <= std_logic_vector(unsigned(counter) + 1) ;
                    else
                        counter <= (others => '0');
                 end if;

                 if (unsigned(counter) = 32250000) then
                    div <= '1';
                 else
                    div <= '0';
                 end if;
            end if;
    end process;
end Behavioral;
```

Divider Top Code

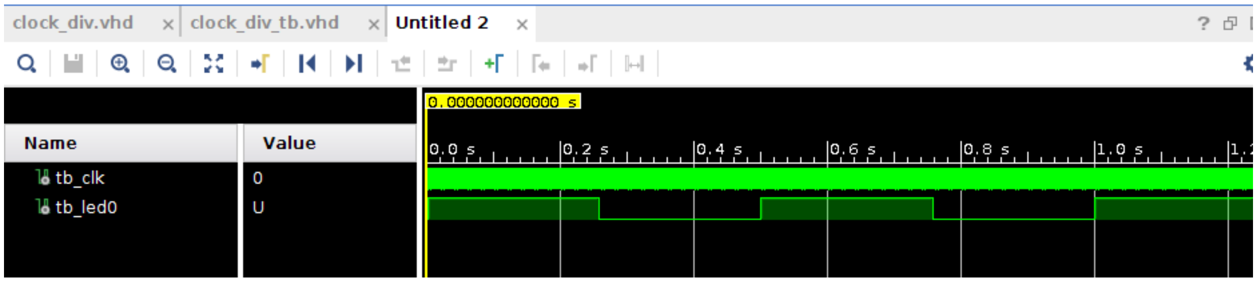
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity reg is
port(clk, chip_enable, input: in std_logic;
     q: out std_logic);
end reg;

architecture my_reg of reg is
begin
    process(clk)
    begin
        if(rising_edge(clk)) then
            if(chip_enable = '1') then
                q <= input;
            else
```

```vhdl
                q <= '0';
            end if;
        end if;
     end process;
end my_reg;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity divider_top is
port(Clock: in std_logic;
     led_out: inout std_logic);
end divider_top;

architecture led_reg of divider_top is
---------------intermediate signals----------------------------
signal new_clk: std_logic;
signal led0_i: std_logic;
--------------clock_div Component------------------------------
component clock_div
Port (clk: in std_logic;
      div: out std_logic);
end component;

component reg
port(clk, chip_enable, input: in std_logic;
     q: out std_logic);
end component;

begin
led0_i <= not led_out;
clock_divider: clock_div
    port map(clk => Clock,
             div => new_clk);

led0_reg: reg
port map(clk => clock,
         chip_enable => new_clk,
         input => led0_i,
         q => led_out);
end led_reg;
```

Test:

```vhdl
library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;

entity clock_div_tb is
end clock_div_tb;

architecture testbench of clock_div_tb is

```

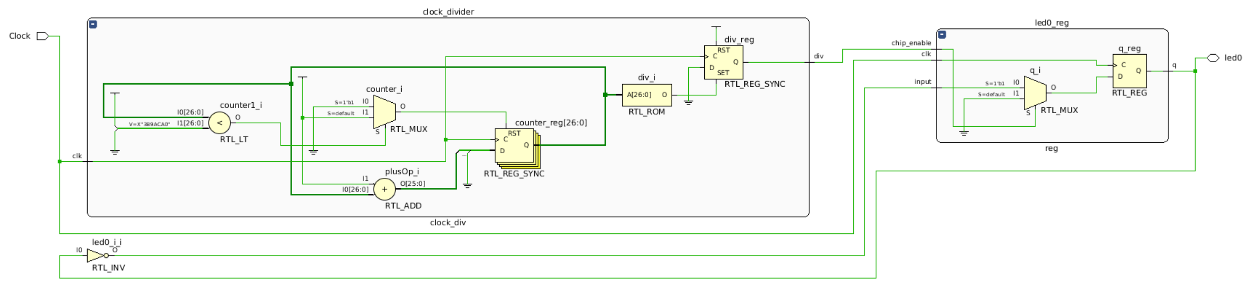
```vhdl
    signal tb_clk : std_logic := '0';
    signal tb_led0 : std_logic;

    component clock_div is
        port(

            clk   : in std_logic;        -- 125 Mhz clock

            div : out std_logic        -- led, '1' = on

        );
    end component;

begin

    -- simulate a 125 Mhz clock
    clk_gen_proc: process
    begin

        wait for 4 ns;
        tb_clk <= '1';

        wait for 4 ns;
        tb_clk <= '0';

    end process clk_gen_proc;


    dut : clock_div
    port map (

        clk  => tb_clk,
        div => tb_led0

    );


end testbench;
```



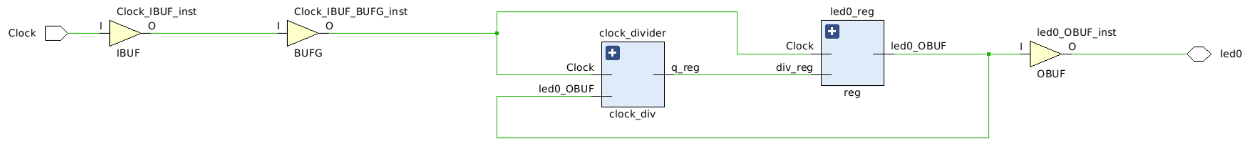
Name	Value
tb_clk	0
tb_led0	U

Implementation:

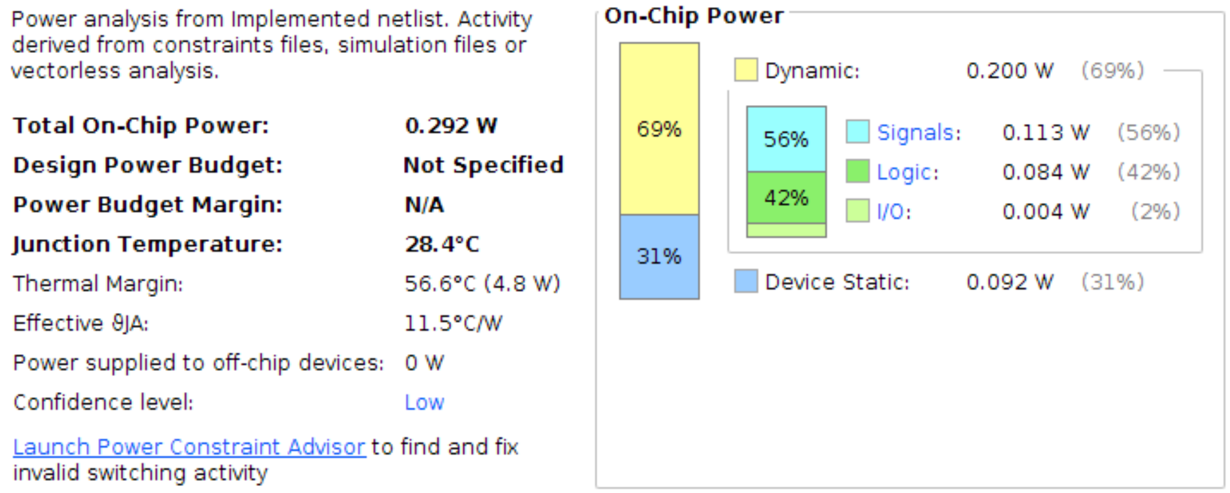
Elaboration Schematic



Synthesis Schematic



Project Summary



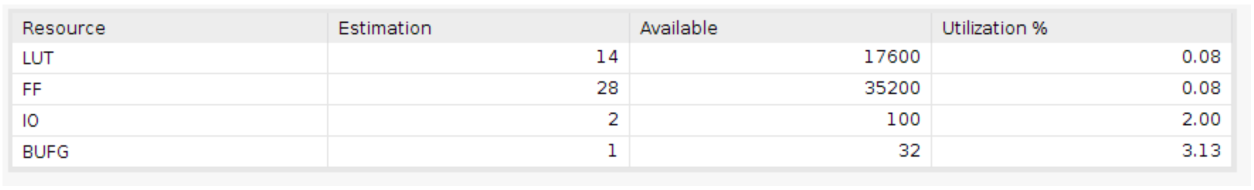
Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.292 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**28.4°C**
Thermal Margin:	56.6°C (4.8 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:		0.200 W	(69%)
	Signals:	0.113 W	(56%)
	Logic:	0.084 W	(42%)
	I/O:	0.004 W	(2%)
Device Static:		0.092 W	(31%)



Resource	Estimation	Available	Utilization %
LUT	14	17600	0.08
FF	28	35200	0.08
IO	2	100	2.00
BUFG	1	32	3.13

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=sysclk
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=SW0
#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=SW3


##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


##LEDs
set_property -dict { PACKAGE_PIN M14   IOSTANDARD LVCMOS33 } [get_ports {
led_out }]; #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 Sch=LED3
```

In this XDC file, I only needed to uncomment the clock for the input and one led for the output.

2. Bouncy Buttons

Theory:

Since the push buttons are mechanical, with springs inside them, the spring causes the contacts to act as a damped oscillator. This means that the signal that is received will not be constant. In order to fixed this issue, we use a 2 bit shift register to make sure that we get a constant ‘1’ signal from the button. We have a counter that counts up and bit 0 of the shift register gets what the button is outputting. Then bit 1 of the shift register gets bit 0. Once bit 1 has been ‘1’ for a specified number of ticks, then we output a ‘1’ from the debounce circuit.

Schematic:

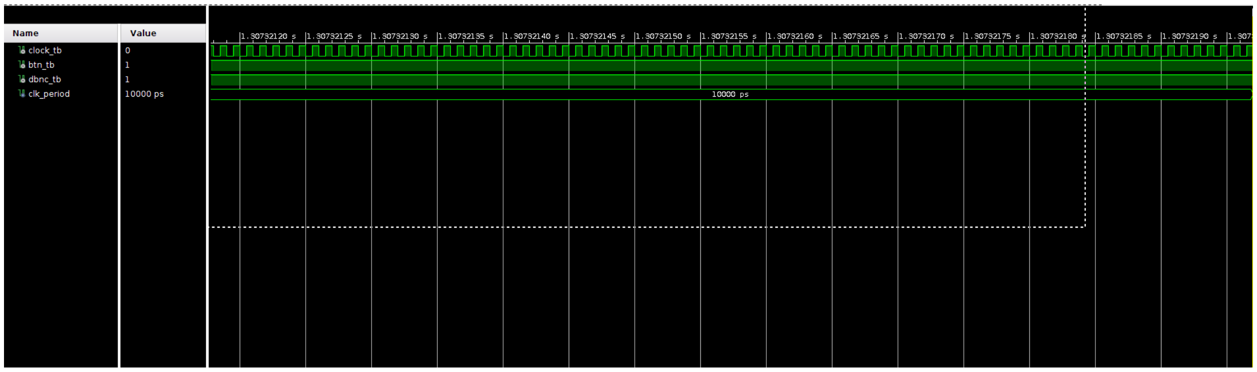
Design:

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity debounce is
Port (clk, btn: in std_logic;
      dbnc: out std_logic);
end debounce;

architecture Behavioral of debounce is

signal counter: std_logic_vector(21 downto 0);
signal count_set: std_logic;
signal shift_register: std_logic_vector(1 downto 0);

begin
    process(clk)
    begin
    count_set <= shift_register(1) xor shift_register(0);
        if(rising_edge(clk)) then
            shift_register(0) <= btn;
            shift_register(1) <= shift_register(0);
                if(count_set = '1') then
                    counter <= (others => '0');
                elsif(counter(21) = '0') then
                    counter <= std_logic_vector(unsigned(counter) + 1);
                else
                    dbnc <= shift_register(1);
                end if;
        end if;
     end process;
end Behavioral;
```

Test:

```vhdl
Library IEEE;
use IEEE.std_logic_1164.all,IEEE.numeric_std.all;

ENTITY debounce_tb IS
END debounce_tb;

architecture testbench of debounce_tb is
```

```vhdl
Component debounce
port(clk :in std_logic;
    btn :in std_logic;
    dbnc :out std_logic);
end Component;

signal clock_tb : std_logic := '0';
signal btn_tb : std_logic;
signal dbnc_tb : std_logic;

constant clk_period : time := 10 ns;

begin
DUT: debounce
    PORT MAP (clk => clock_tb,
              btn => btn_tb,
              dbnc => dbnc_tb);

generate_clk_process :process
    begin
        clock_tb <= '0';
        wait for clk_period/2;
        clock_tb <= '1';
        wait for clk_period/2;


end process;


simulate_process: process
    begin
        wait for 100ns;
        btn_tb<='1';
        wait;
    end process;
end;
```



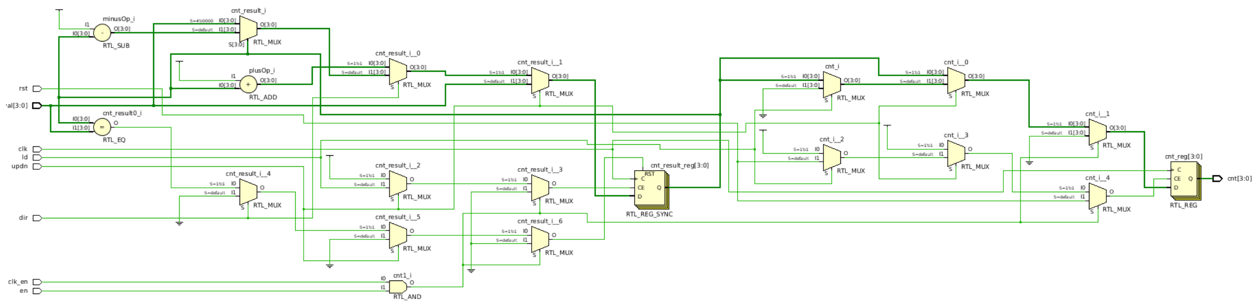
```
clock_tb     0
btn_tb       1
dbnc_tb      1
clk_period   10000 ps
```

3. Actually Using a Counter to Count

Theory:

The counter that we want to build here receives the clock from the board. However, it will only do something when its clk_en and en are ‘1’, which are connected to our clock divider and a button respectively. We need to be able to load a 4 bit value when a ld signal is ‘1’, which we will set from the switches. There needs to be the ability to count up or down depending on what our direction bit is set to. When the counter hits the value the set value when counting up, the counter will loop back to ‘0000’. When the counter hits ‘0000’ when counting down, counter will reset to the set value and count back down.

Schematic Diagram:



Design:

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity fancy_counter is
Port (clk: in std_logic;
      clk_en: in std_logic;
      dir: in std_logic;
      ld, en: 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 fancy_counter;

architecture Behavioral of fancy_counter is
signal cnt_result: std_logic_vector(3 downto 0) := (others => '0');
signal hold: std_logic_vector(3 downto 0) := (others => '0');
begin
```

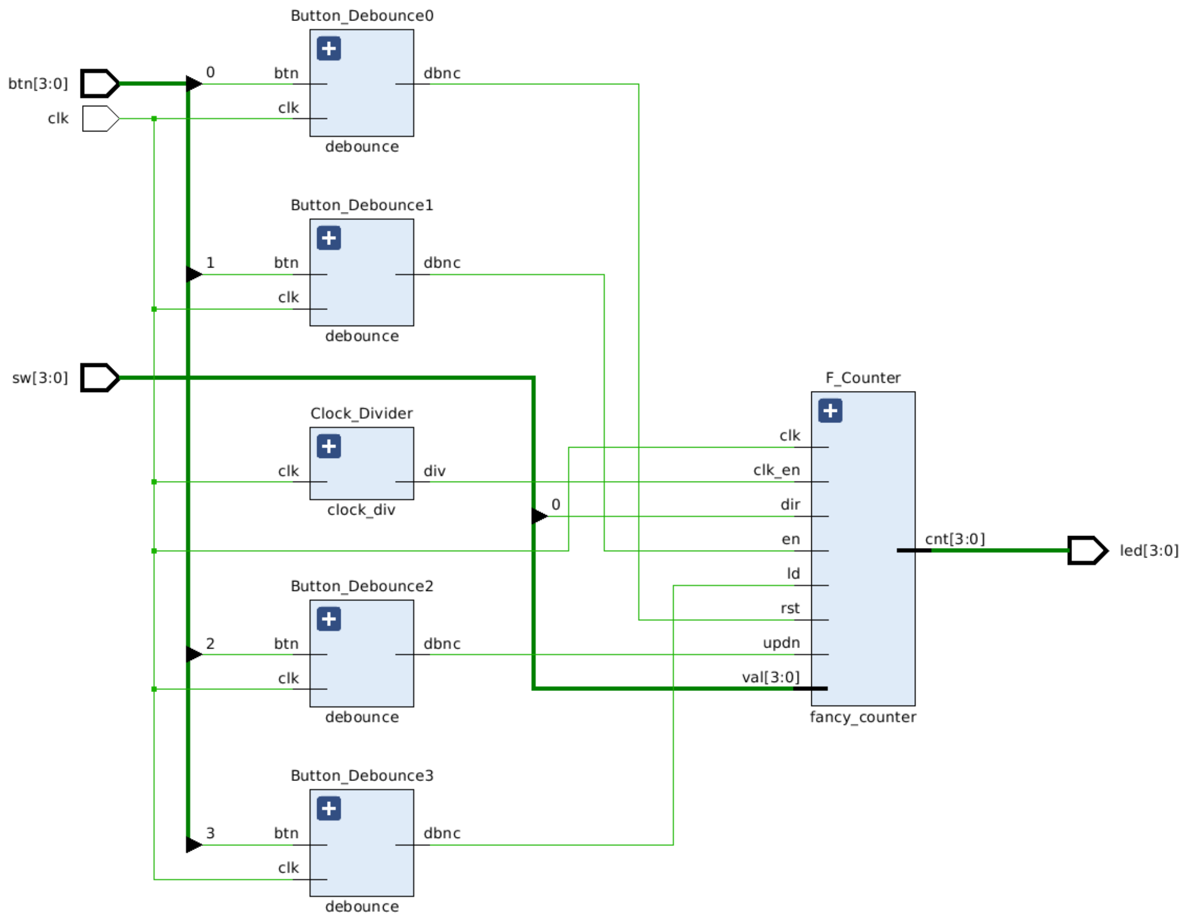
```
    process(clk)
    begin
        if(rising_edge(clk)) then
            if(rst = '1') then
                cnt <= (others => '0');
            end if;
            if(clk_en = '1' and en = '1') then
                if(ld = '1') then
                    cnt_result <= val;
                    hold <= cnt_result;
                    cnt <= cnt_result;
                end if;
                if(updn = '1') then
                    if(dir = '1') then
                        cnt_result <=
std_logic_vector(unsigned(cnt_result) + 1);
                        cnt <= cnt_result;
                        if(cnt_result = hold) then
                            cnt_result <= "0000";
                        end if;
                    else
                        cnt_result <=
std_logic_vector(unsigned(cnt_result) - 1);
                        cnt <= cnt_result;
                        if(cnt_result = "0000") then
                            cnt_result <= hold;
                            cnt <= cnt_result;
                        end if;
                    end if;
             end if;
            end if;
        end if;
    end process;

end Behavioral;
```

4. Putting it All Together

Theory:

Now that we have all the necessary components, we should be able to combine everything into one top level design. We will control the value of val(3:0) with the switches. All the buttons on the board will go through the debounce circuit to ensure that the we receive an output of ‘1’ through them. The buttons will control the enable, reset, load and up down functionality. Finally, the output will go to the leds.

Schematic Diagram:



Design:

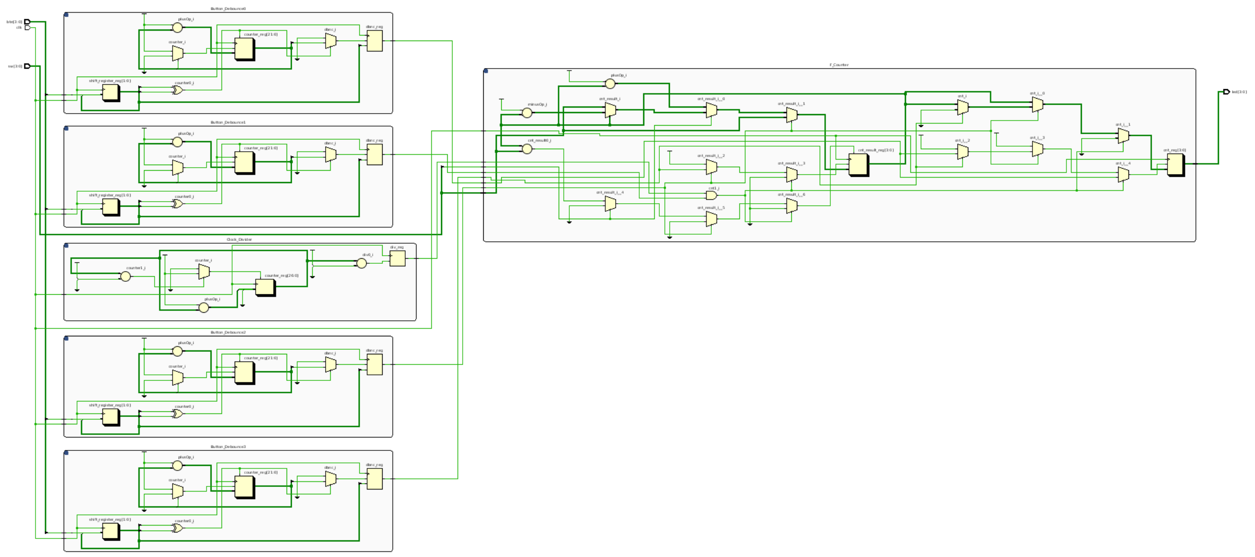
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

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

architecture counter of counter_top is

```

```
signal div_result: std_logic;
signal dbnc: std_logic_vector(3 downto 0) := (others => '0');

--------------------Fancy Counter Component-------------------------
component fancy_counter
Port (clk: in std_logic;
      clk_en: in std_logic;
      dir: in std_logic;
      ld, en: 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;

--------------------Debounce Component--------------------------------
component debounce
Port (clk, btn: in std_logic;
      dbnc: out std_logic);
end component;

--------------------Clock Divider Component---------------------------
component clock_div is
    Port (clk: in std_logic;
          div: out std_logic);
end component;

begin

Clock_Divider: clock_div
    port map(clk => clk,
             div => div_result);

Button_Debounce0: debounce
    port map(clk => clk,
             btn => btn(0),
             dbnc=> dbnc(0));

Button_Debounce1: debounce
    port map(clk => clk,
             btn => btn(1),
             dbnc=> dbnc(1));

Button_Debounce2: debounce
    port map(clk => clk,
             btn => btn(2),
             dbnc=> dbnc(2));

Button_Debounce3: debounce
                 port map(clk => clk,
                          btn => btn(3),
                          dbnc=> dbnc(3));
```

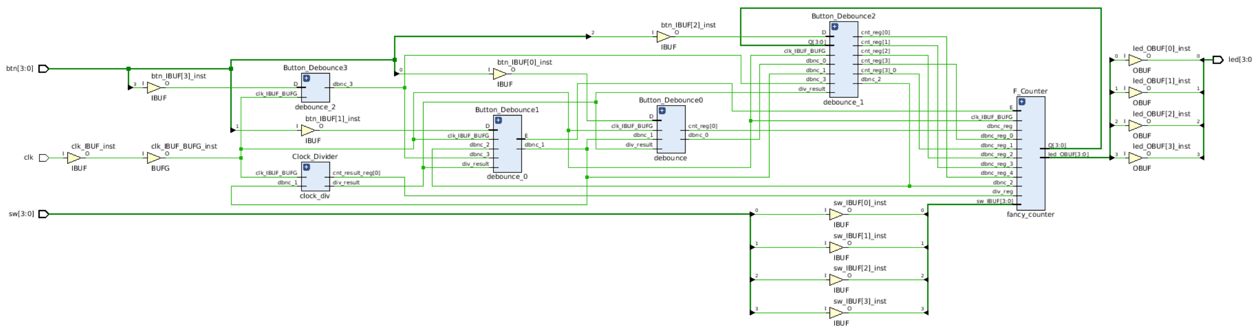
```vhdl

F_Counter: fancy_counter
    port map(clk => clk,
             clk_en => div_result,
             dir => sw(0),
             en => dbnc(1),
             ld => dbnc(3),
             rst => dbnc(0),
             updn => dbnc(2),
             val(0) => sw(0),
             val(1) => sw(1),
             val(2) => sw(2),
             val(3) => sw(3),
             cnt(0) => led(0),
             cnt(1) => led(1),
             cnt(2) => led(2),
             cnt(3) => led(3));

end counter;
```

Implementation:

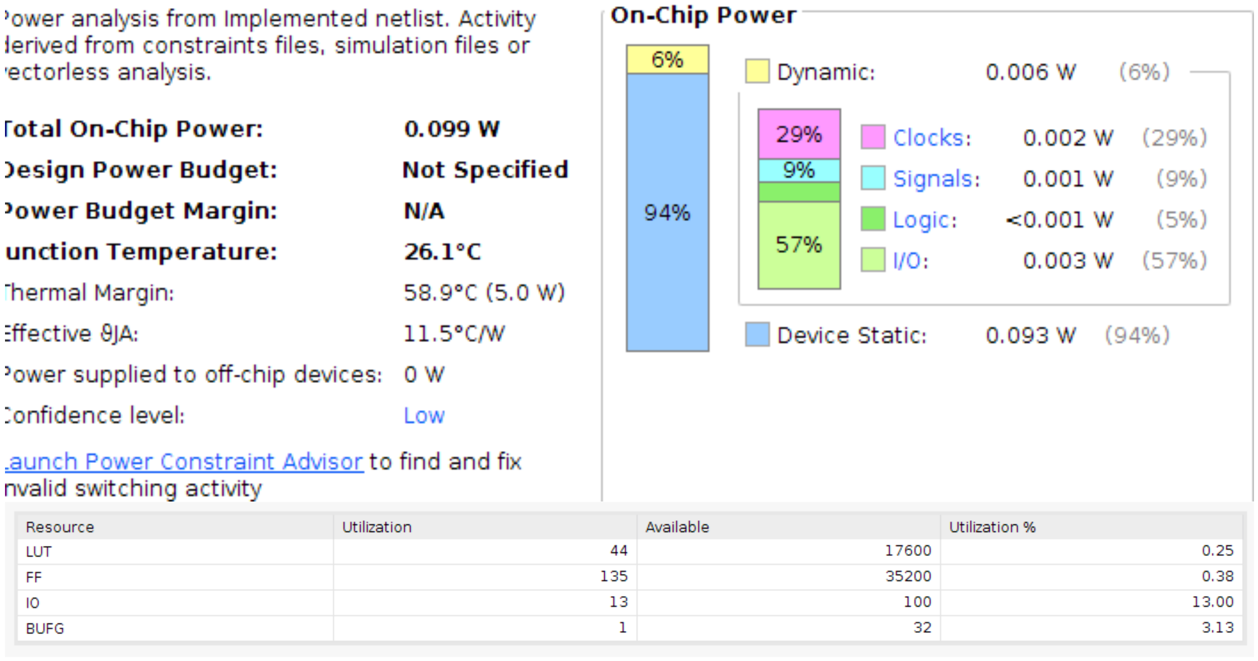
Elaboration Schematic



Synthesis Schematic



Project Summary(Util table, on chip power)



ower analysis from Implemented netlist. Activity
erived from constraints files, simulation files or
ectorless analysis.

otal On-Chip Power:	**0.099 W**
esign Power Budget:	**Not Specified**
ower Budget Margin:	**N/A**
unction Temperature:	**26.1°C**
hermal Margin:	58.9°C (5.0 W)
ffective ϑJA:	11.5°C/W
ower supplied to off-chip devices:	0 W
onfidence level:	Low

aunch Power Constraint Advisor to find and fix
nvalid switching activity

On-Chip Power

	Dynamic:	0.006 W	(6%)
	Clocks:	0.002 W	(29%)
	Signals:	0.001 W	(9%)
	Logic:	<0.001 W	(5%)
	I/O:	0.003 W	(57%)
	Device Static:	0.093 W	(94%)

Resource	Utilization	Available	Utilization %
LUT	44	17600	0.25
FF	135	35200	0.38
IO	13	100	13.00
BUFG	1	32	3.13

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=sysclk
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=SW0
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=SW3


##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


##LEDs
set_property -dict { PACKAGE_PIN M14   IOSTANDARD LVCMOS33 } [get_ports { led[0] }];
#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 Sch=LED3
```

In the XDC file, I needed to uncomment all the buttons, switches, leds, and the clock. Additionally, I had to make sure my port names matched my top level design.

Discussion:

1.1: In order to go from 125 MHz to 2 Hz we need to divide it by 4.

1.2: We need 25 bits in order to represent the value from 1.1

2.1: The value for the buttons when it is pressed for the Zybo will oscillate between 0 and 1 (due to the mechanical nature of the button) until it settles at '1'.

2.3: For a debounce time of 20ms at 125MHz, we need 2,500,000 ticks.

2.4: We need 22 bits for 2,500,000

What I learned from this lab is the need to debounce a signal from the buttons. If we don't do this, then the output will not be stable when we run it. Additionally, I had some trouble when getting the counter to run correctly on the board. When it was running, it was counting up or down too fast. That was because in my clock divider, the output signal would read '1' for a while at the same time as the on the rising edge of the clock from the board. What we needed was a pulse, so I had to change a less than sign to an equals sign. I understand how to create a clock divider and a 2 bit debouncer for the most part. Overall, this first lab it was not too bad, but there was a lot to pick up.