Report on LAB 1

(Topics in Adv Comp Eng: Embedded Systems Hardware EE 493)

NAME: Prince K. Bose
NET ID: pkb44
RUID: 186008149

DATE: 2/28/19

Purpose

The aim of this experiment is to create different blocks/ systems, like clock divider, debounce and fancy counter. Individually test them, by making individual test benches. And ultimately, combining every system onto a single system using Top level design.

This is opposite to the block diagram approach discussed in class, where we start at a very high-level black box. Gradually we keep defining a newer block that does a specific task, adding up to the entirety of the circuit functioning.

This Lab focuses on the bottom up approach of designing, where we take care of individual tasks first, and then we combine them to make an entire circuit.

The use of this fancy counter is to either up count to a specific "value" decided by the two way switches (an on board peripheral of the Zybo) or down count froma specific "value", again decided by the switches on the board.

The en pin decides whether the circuit runs or not.

The direction of count (up/down) is decided by the sw0 (LSB) of the value. In case, sw0 is '1' the counter should count UP, upon loading the value to the direction register, when 'updn' signal is high.

This circuit needs to print the counter value onto the LEDs representing the binary form of the "value".

Example Run Case: If en is '1', 'value' is 3, dir is '1' and rst pin is '0', then the flow of the counter becomes:

0000 → 0001 → 0010 → 0011 → 0000

S.No.	LED0	LED1	LED2	LED3
1	0	0	0	0
2	0	0	0	1
3	0	0	1	0
4	0	0	1	1

There are 4 main components to the circuit :
1. Clock Divider
2. Debounce Switch
3. Fancy Counter
4. Top Level Design

Each of these components are discussed in the following sections.

1. CLOCK DIVIDER

The Zybo board has an inbuilt clock frequency of 125Mhz which has a time period of 8 nanoseconds. As we know, a clock signal is the heart of all FPGA systems and all subsystems on a synchronous circuit rely heavily on having the exact same clock signal as it helps the subsystems synchronize and communicate efficiently.

It is impossible for the human eye to notice an LED blinking at the rate of 125Mhz. Imagine an LED remaining ON for 4 ns and OFF for 4 ns. That means 250000000 switches between ON and OFF state per second. You definitely cannot notice that without a Super Slomo camera.

In order to tackle this problem we have circuits called clock divider circuits, that essentially derive lower frequency clocks from higher frequency Master clocks. This is done using counters. We keep adding a 1 to a count register till a specific value, then we switch the output clock. This helps us derive a clock that has a greater clock cycle duration than the original clock.

In my clock divider code, at every rising clock edge, I count up to 31250000, before resetting the counter to 0 and the switching the output clock. This gives me a desired output frequency of 2 Hz. The time period of the output clock is 500 ms with a 50% duty cycle (250ms ON, 250 ms OFF).

Design

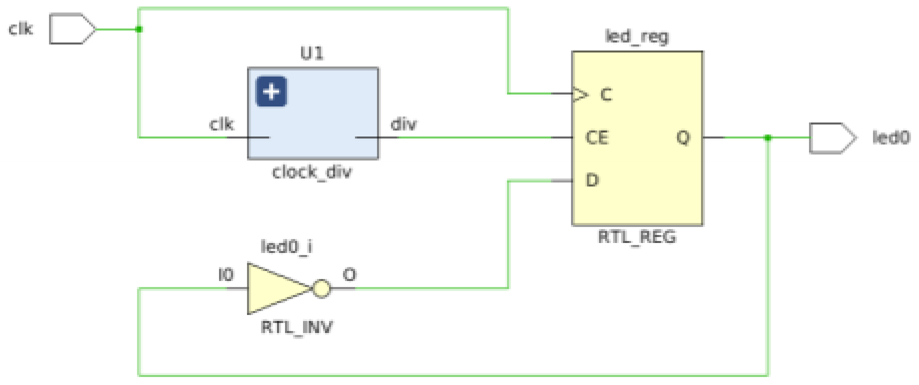


Fig 1. Desired Schematic of Clock Divider

The desired circuit consists of an input as clk and an output div. The divider_top consists of a clock divider and a D Flip Flop that switches an output LED at the rate of 2Hz.

VHDL Code

```
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 clk_div of clock_div is
  signal count : std_logic_vector (25 downto 0) := (others => '0');
  signal temp : std_logic := '0';
begin
  process(clk)
  begin
    if rising_edge(clk) then
        if(unsigned(count) < 62500000/2) then
        count <= std_logic_vector( unsigned(count) + 1 );
        else
        count <= (others => '0');
        temp <= NOT(temp);
        end if;
    end if;
  end process;
div <= temp;
end clk_div;
```

VHDL Testbench

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.Numeric_std.all;

entity tb_clock_div is
end tb_clock_div;

architecture clk_tb of tb_clock_div is
  component clock_div
  port (
    clk : in std_logic;
    div : out std_logic
  );
  end component;
  signal clk : std_logic := '0';
  signal div : std_logic := '0';
begin
  dut: clock_div port map(clk => clk,
                          div => div);
  process
  begin
      clk <= '1';
      wait for 4 ns;
      clk <= '0';
      wait for 4 ns;
  end process;
end clk_tb;
```

Output Schematic

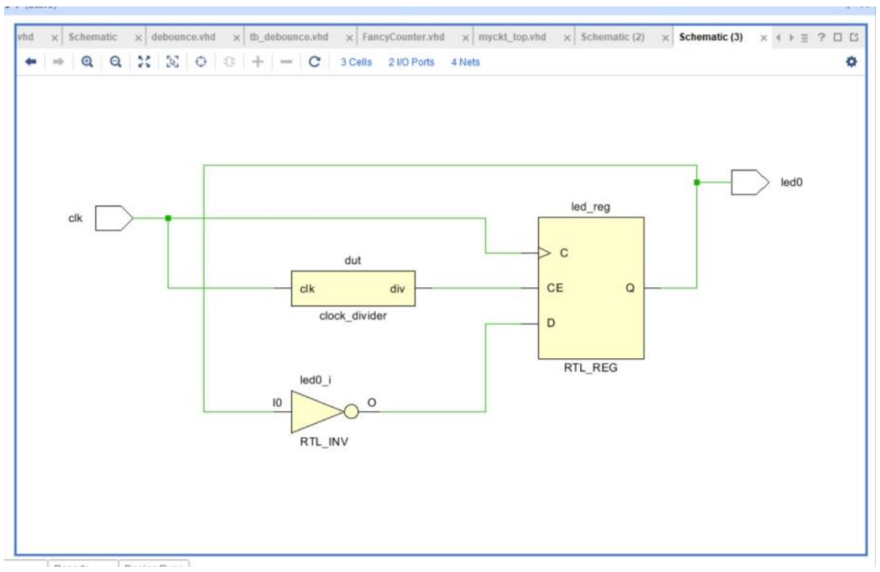


Fig 2. Obtained Schematic of Clock Divider

The obtained schematic matches the desired schematic.

Simulation Results

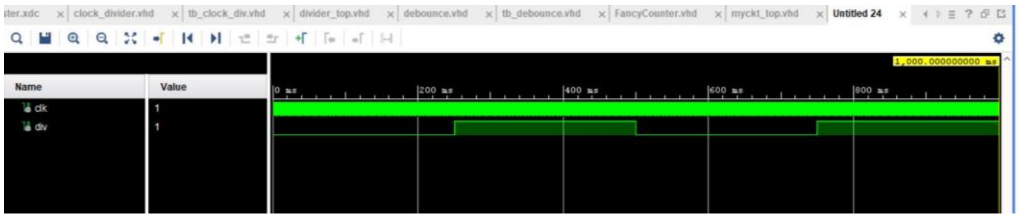


Fig 3. Simulation – Clock Divider

Here clk shows the input clock of 125 MHz and div represents the digitally generated output clock of 2 Hz.

2. DEBOUNCE

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.

This means, in between a swap from ‘0’ to ‘1’ or LOW to HIGH or ‘1’ to ‘0’ or HIGH to LOW, the switch goes through a few meta stable stages in between. During these meta stable stages, the output value may fluctuate.

It looks something like this :

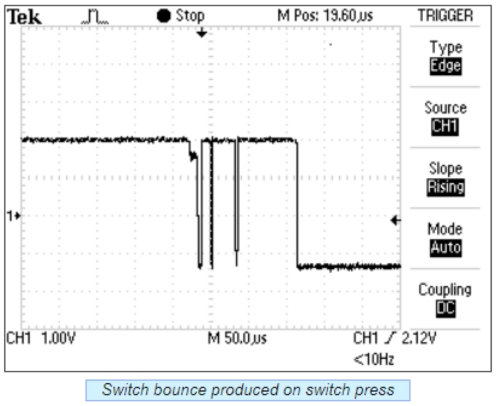


Fig 4. Switch Debouncing

DESIGN

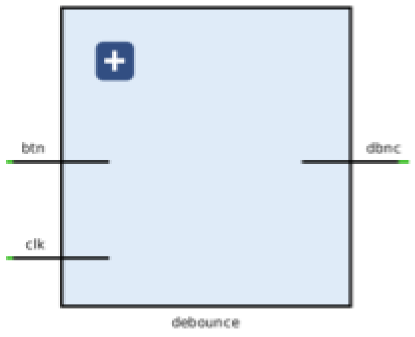


Fig. 5 Block Diagram of a Debounce Circuit

When a state changes from ‘0’ to ‘1’, the circuit waits for 20 ms for the switch to come to a stable state, before finally changing the output of the debounce at ‘1’.

In order to do this, we set the count value to 2499999.

VHDL Code

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


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

architecture Behavioral of debounce is
signal sh : std_logic_vector(1 downto 0) := "00";
signal cnt_value : std_logic_vector(22 downto 0):=(others => '0');
begin
process(clk,btn)
begin

if (rising_edge(clk)) then
   sh(1) <= sh(0);
   sh(0) <= btn;
   if (unsigned(cnt_value) < 2499999) then
      dbnc <= '0';
      if (sh(1) = '1') then
         cnt_value <= std_logic_vector(unsigned(cnt_value)+1);
      else
         cnt_value <= (others => '0');
      end if;
   else
      dbnc <= '1';
      if(btn = '0') then
         dbnc <= '0';
         cnt_value <= (others => '0');
      end if;
   end if;
end if;
end process;
end Behavioral;
```

VHDL Testbench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity tb_debounce is
end tb_debounce;

architecture Behavioral of tb_debounce is
component debounce
   Port (clk: in std_logic;
      btn: in std_logic;
      dbnc: out std_logic );
end component;
signal tb_clk: std_logic := '0';
signal tb_btn: std_logic := '0';
signal tb_dbnc: std_logic := '0';
begin
devut: debounce port map(   clk => tb_clk,
btn => tb_btn,
dbnc => tb_dbnc);
clock:  process
begin
   tb_clk <= '1';
   wait for 4 ns;
   tb_clk <= '0';
   wait for 4 ns;
end process;
button: process
begin
tb_btn <= '0';
wait for 200us;
tb_btn <= '1';
wait for 60ms;
tb_btn <= '1';
wait for 2ms;
tb_btn <= '1';
wait for 1ms;
tb_btn <= '0';
tb_btn <= '1';
wait for 22ms;
tb_btn <= '0';
wait for 1ms;
tb_btn <= '1';
wait for 50 ms;
end process;
end Behavioral;
```

Simulation Results

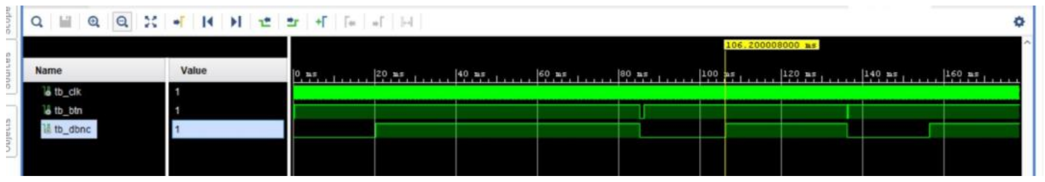


Fig 6. Simulation Waveform for Debounce

As seen in the simulation waveform above, the circuit waits for 20ms till the output is stable at ‘1’. After 20s, the output is pulled up to ‘1’. As soon a ‘0’ is received, the output is pulled down to ‘0’.

3. Fancy Counter

A fancy counter is nothing but a simple 4 bit bidirectional counter with some extra control signals.

Required Schematic

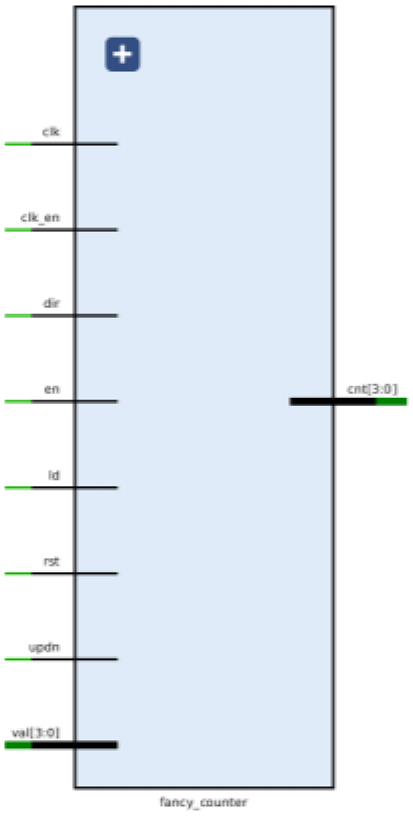


Fig 7. Fancy Counter Schematic

- 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
- On the clock rising edge, 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.

VHDL Code

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

entity FancyCounter 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 FancyCounter;

architecture Behavioral of FancyCounter is

signal count: std_logic_vector(3 downto 0) := (others => '0');
signal uploadValue: std_logic_vector(3 downto 0) := (others => '0');
signal direction: std_logic := '0';

begin
process(clk,clk_en)
begin
   if (rst = '1') then
         count <= (others => '0');
         cnt <= count;
   end if;
   if (en ='1') then

         if(clk_en = '1') then
            if (ld = '1') then
               uploadValue <= val;
            end if;
            if (updn = '1') then
               direction <= dir;
            end if;
            if(direction = '1') then
               if(unsigned(count) < unsigned(uploadValue)) then
                  count <= std_logic_vector(unsigned(count) + 1);
                  cnt <= count;
                else
                  count <= (others => '0');
                  cnt <= count;
                end if;
            else
               if(unsigned(count) > "0000") then
                  count <= std_logic_vector(unsigned(count) - 1);
                  cnt <= count;
               else
                  count <= uploadValue;
                  cnt <= count;
               end if;
            end if;
         end if;
      end if;
end process;
end Behavioral;
```

<u>VHDL Testbench</u>

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

entity fc_tb is
end fc_tb;

architecture Behavioral of fc_tb is
component FancyCounter
     Port (clk, clk_en, dir, en, ld, rst, updn: in std_logic;
          val: in std_logic_vector(3 downto 0);
          cnt: out std_logic_vector(3 downto 0));
end component;
signal clk, clk_en, en, dir, ld, rst, updn : std_logic := '0';
signal val: std_logic_vector(3 downto 0) := (others => '0');
signal cnt: std_logic_vector(3 downto 0) := (others => '0');
begin
dut: FancyCounter port map(clk => clk,
                     clk_en => clk_en,
                     en => en,
                     dir => dir,
                     ld => ld,
                     rst => rst,
                     updn => updn,
                     val => val,
                     cnt => cnt);
   process
```

```
        begin
        --8ns clock cycle = 125MHZ

      wait for 4 ns;
      clk <= '1';
      wait for 4 ns;
      clk <= '0';

  end process;
 process
        begin

  wait for 5ms;
        en <= '0';
        clk_en <= '1';
        dir <= '1';
        ld <= '1';
        val <= "1001";
        updn <= '1';
        rst <= '0';
        wait for 10ms;
        en <= '1';
        clk_en <= '0';
        ld <= '1';
        val <= "1001";
        updn <= '1';
        --rst should set the output to '0'
        rst <= '1';
        wait for 5ms;
        rst <= '0';
        wait for 1ms;
        en <= '1';
        clk_en <= '1';
        dir <= '1';
        ld <= '1';
        val <= "1001";
        updn <= '1';
        wait for 8ns;
        clk_en <= '0';
        wait for 8ns;
        clk_en <= '1';
        wait for 8ns;
        clk_en <= '0';
        wait for 8ns;
        clk_en <= '1';
        wait for 8ns;
```

```
clk_en <= '0';
wait for 8ns;
clk_en <= '1';
wait for 8ns;
clk_en <= '0';
wait for 8ns;
clk_en <= '1';
wait for 8ns;
clk_en <= '0';
wait for 8ns;
clk_en <= '1';
wait for 8ns;
clk_en <= '0';
wait for 8ns;
clk_en <= '1';
wait for 8ns;
clk_en <= '0';

wait for 20ms;
en <= '1';
clk_en <= '1';
dir <= '0';
ld <= '0';
val <= "1001";
updn <= '1';
wait for 8ns;
clk_en <= '0';
wait for 8ns;
clk_en <= '1';
wait for 8ns;
clk_en <= '0';
wait for 8ns;
clk_en <= '1';
wait for 8ns;
clk_en <= '0';
wait for 8ns;
clk_en <= '1';
wait for 8ns;
clk_en <= '0';
wait for 8ns;
clk_en <= '1';
wait for 8ns;
clk_en <= '0';
wait for 8ns;
clk_en <= '1';
wait for 8ns;
```

```
        clk_en <= '0';
        wait for 8ns;
        clk_en <= '1';
        wait for 8ns;
        clk_en <= '0';
        wait for 10ms;
        rst <= '1';
        clk_en <= '0';
        dir <= '0';
        ld <= '0';
        val <= "0000";
        updn <= '0';
        wait;
end process;
end Behavioral;
```

Simulation Results

As seen in the following timing diagram,

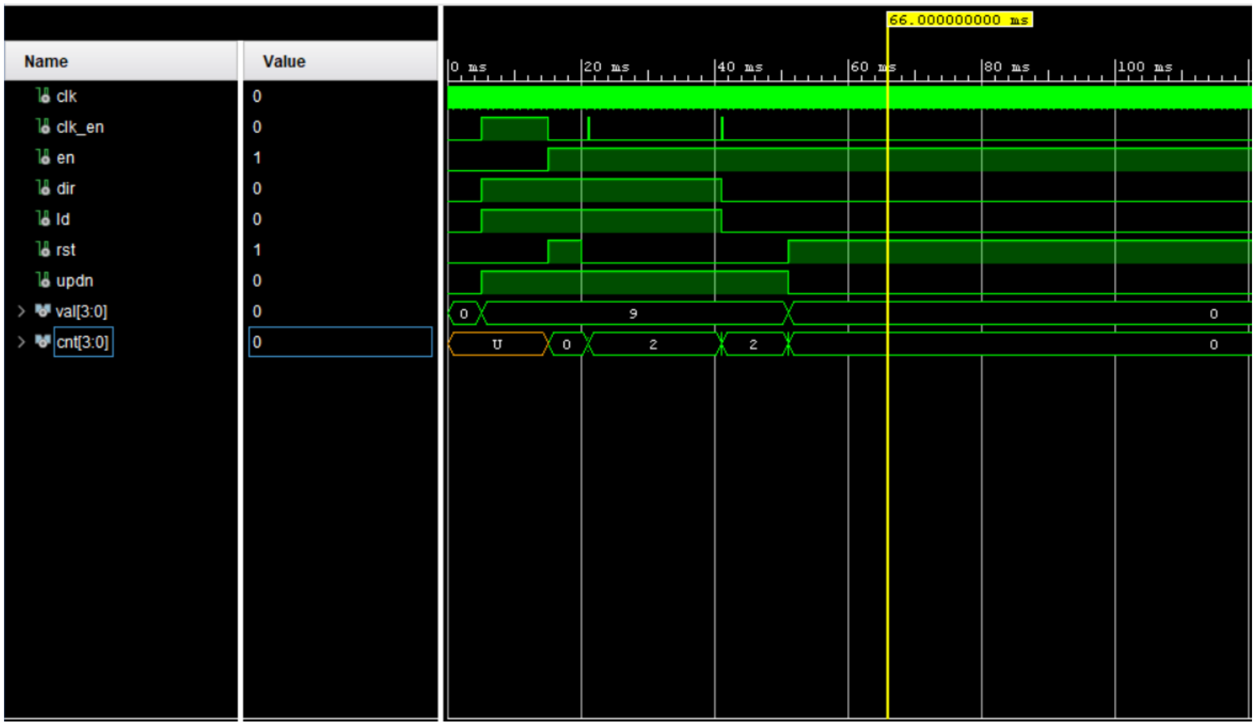


Fig 8. Simulation Waveform

4. Top Level Synthesis

The top level synth only combines all available and pre made devices/ systems and creates instances of each component.
The intermediate wiring is done using locally created signals.

VHDL Code

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


entity myckt_top is
    Port ( btn : in STD_LOGIC_VECTOR (3 downto 0);
           sw : in STD_LOGIC_VECTOR (3 downto 0);
           led : out STD_LOGIC_VECTOR (3 downto 0);
           clk : in STD_LOGIC);
end myckt_top;

architecture Behavioral of myckt_top is

component clock_div
port (clk : in std_logic;
     div : out std_logic
);
end component;

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

component FancyCounter
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;
```

```
signal dev1_out,dev2_out,dev3_out,dev4_out,temp_div : std_logic:= '0';

begin

dev1: debounce
port map( btn => btn(0),
        clk => clk,
        dbnc => dev1_out);

dev2: debounce
port map( btn => btn(1),
        clk => clk,
        dbnc => dev2_out);

dev3: debounce
port map( btn => btn(2),
        clk => clk,
        dbnc => dev3_out);

dev4: debounce
port map( btn => btn(3),
        clk => clk,
        dbnc => dev4_out);

dut_clk: clock_div
port map( clk => clk,
        div => temp_div);

dut_fc: FancyCounter
port map( clk => clk,
        clk_en => temp_div,
        dir => sw(0),
        en => dev2_out,
        ld => dev4_out,
        rst => dev1_out,
        updn => dev3_out,
        val => sw,
        cnt => led);

end Behavioral;
```

Full Top Level Elaboration Schematic

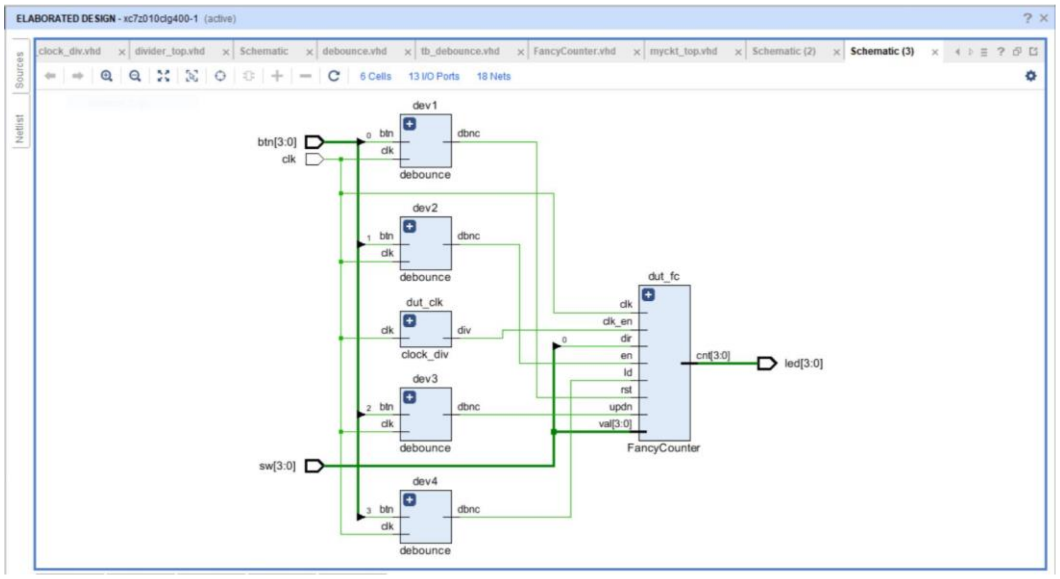


Fig 9. Full Schematic

Synthesis Schematic

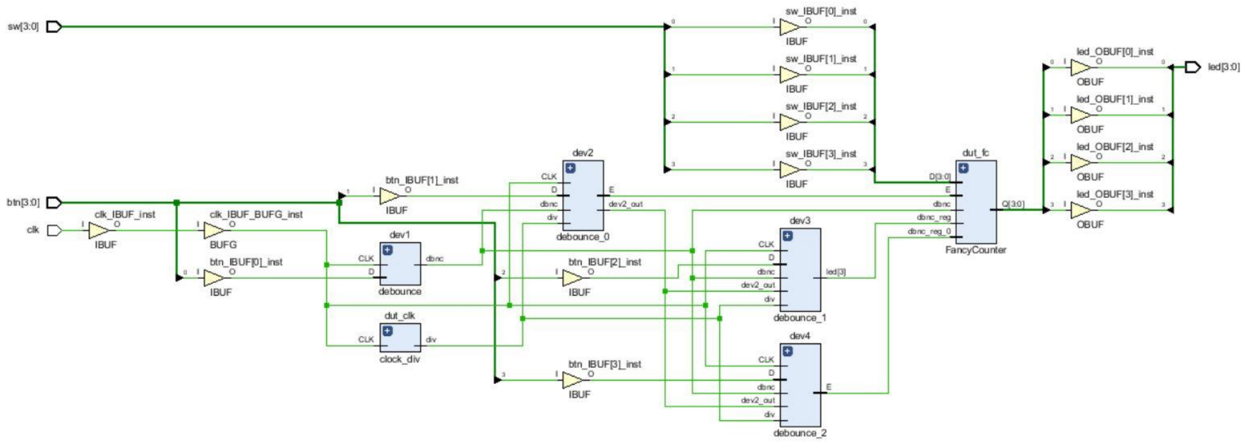
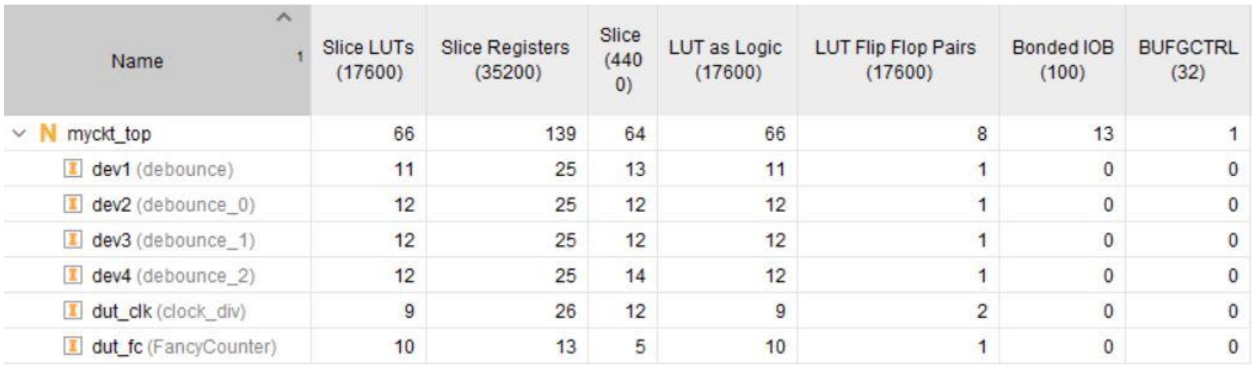


Fig 10. Synthesis Schematic of myckt_top

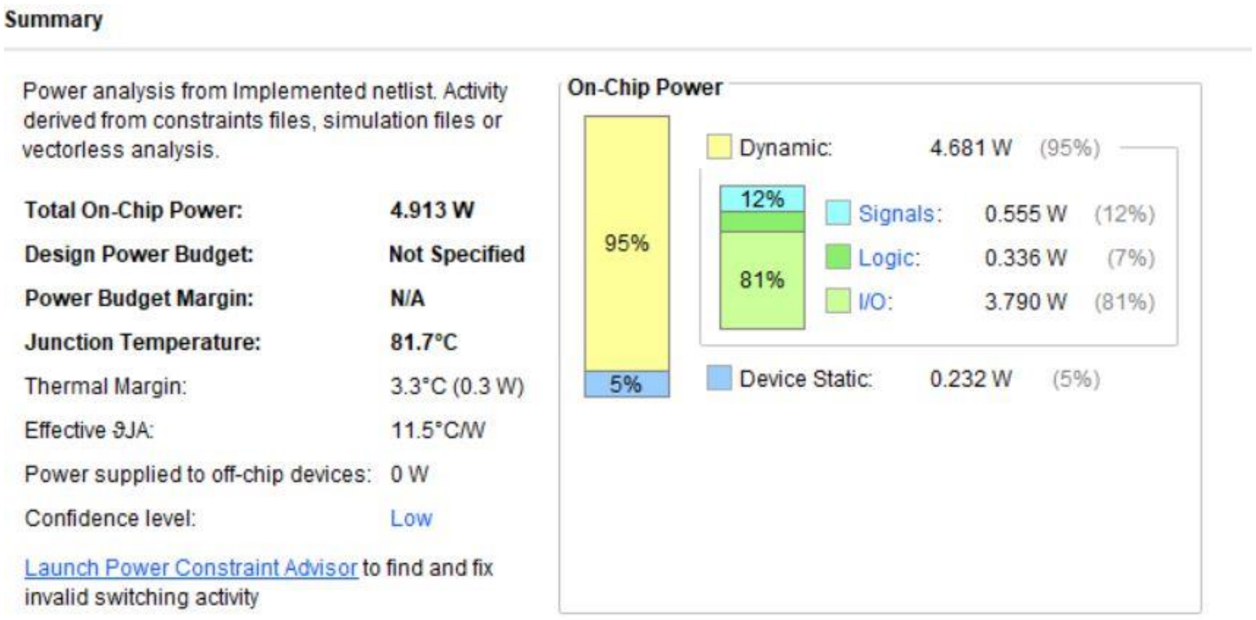
Utilization Report



Name	Slice LUTs (17600)	Slice Registers (35200)	Slice (4400)	LUT as Logic (17600)	LUT Flip Flop Pairs (17600)	Bonded IOB (100)	BUFGCTRL (32)
myckt_top	66	139	64	66	8	13	1
dev1 (debounce)	11	25	13	11	1	0	0
dev2 (debounce_0)	12	25	12	12	1	0	0
dev3 (debounce_1)	12	25	12	12	1	0	0
dev4 (debounce_2)	12	25	14	12	1	0	0
dut_clk (clock_div)	9	26	12	9	2	0	0
dut_fc (FancyCounter)	10	13	5	10	1	0	0

Fig 11. Resource Utilization Report

Power Consumption Report



Summary

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

Total On-Chip Power:	**4.913 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**81.7°C**
Thermal Margin:	3.3°C (0.3 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:	4.681 W	(95%)
🟦	Signals:	0.555 W	(12%)
🟩	Logic:	0.336 W	(7%)
🟩	I/O:	3.790 W	(81%)
🟦	Device Static:	0.232 W	(5%)

Fig 12. Power Consumption

Discussion

Questions for Task 1

1.1 : We divide input frequency 125000000 by 2 to get an output of 2 Hz.

1.2 : 26 bits can store the value of 125000000/4 or 62500000/2.

Questions for Task 2

2.1: The analog voltage of the button goes upto V_{IH}.

2.2: If it was some other value, we would have to change the count value as the time taken to reach a different voltage level would be different.

2.3: The number of ticks required to reach 20 ms is 2499999.

2.4: The number of bits that can store 2499999 are 22.

Observations

1. In the XDC file, while using multiple switches/leds/buttons we use '[' to denote each bit. While in the VHDL source files, we use '('.
2. A conditional in the code is converted into a MUX.
3. Any register that stores a value is represented as a D Flip Flop.
4. All inputs and outputs have buffers.
5. I/O utilizes max power (81% for my design).
6. Total power consumption for the circuit is approximately 5W.
7. A total of 66 LUTs out of a 17600 are used for this design.

Questions/ Follow Up:

Concepts Understood:
1. Switches, Buttons, LEDs
2. Debouncing
3. Clock Division

Concepts Unsure of:
1. Why do we not debounce a switch when it is pulled down?
2. Is there any way to upscale a frequency?
3. How do the frequency boosts in Intel CPU's work? Is overclocking digital?