Embedded Systems Spring 2019
Lab 1

Jonathan Colella

February 28, 2019

Purpose

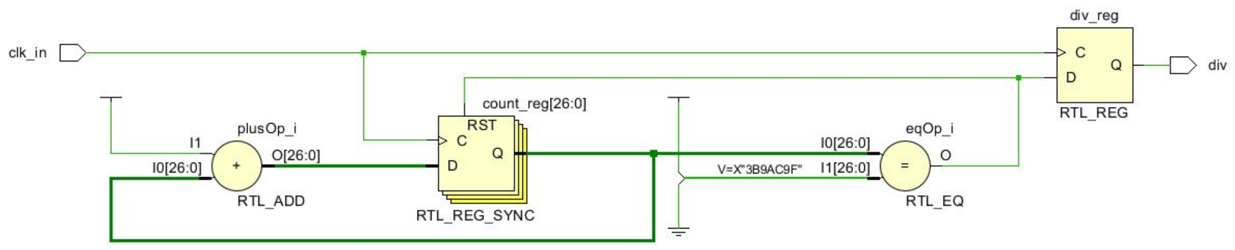
This lab introduces counters and debouncers in the context of VHDL modelling. I will create models for the various required components, then combine them into a single unit. The lab assignment will help develop my thinking when writing VHDL, and to gain experience in designing embedded systems. This will help serve as a foundation for later labs in this course.

1a) Clock Divider

Theory of Operation

This device takes in a 125MHz clock signal and outputs a 2Hz clock signal. It operates by counting every time a clock is input, and only outputs a clocked ‘1’ once for every 62,500,000 input clocks. It resets the counter each time this happens.

Elaboration Schematic



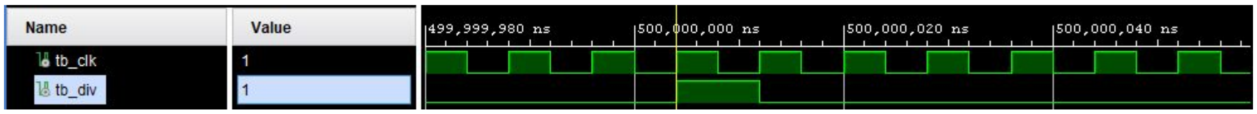
Design

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL; --this library makes incrementing the vector
easier

entity clock_div is
    Port ( clk_in : in STD_LOGIC;
           div : out STD_LOGIC);
end clock_div;

--divides a 125MHZ signal down to a 2Hz signal
architecture Behavioral of clock_div is
signal count: std_logic_vector (26 downto 0); --26 bits needed to count to
62,499,999
begin

process(clk_in) begin
    if (rising_edge(clk_in)) then
        if (count = "11101110011010110010011111") then --binary for 62,499,999
            count <= (others => '0'); --reset to 0 when we get to 62.5 million
            div <= '1'; --output is high
        else
        count <= count + 1; --increment by one otherwise with help of unsigned
        div <= '0'; --output is low
        end if;
    end if;
end process;

end Behavioral;
```

Test Bench

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

entity clock_div_tb is
end clock_div_tb;

architecture tb of clock_div_tb is

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

signal tb_clk : std_logic := '0';
signal tb_div : std_logic := '0';

begin

dut: clock_div port map (clk_in => tb_clk, div => tb_div);

process begin
        tb_clk <= '0'; --simulate 125MHz clock
        wait for 4 ns;
        tb_clk <= '1';
        wait for 4 ns;
end process;

end tb;
```

Simulation



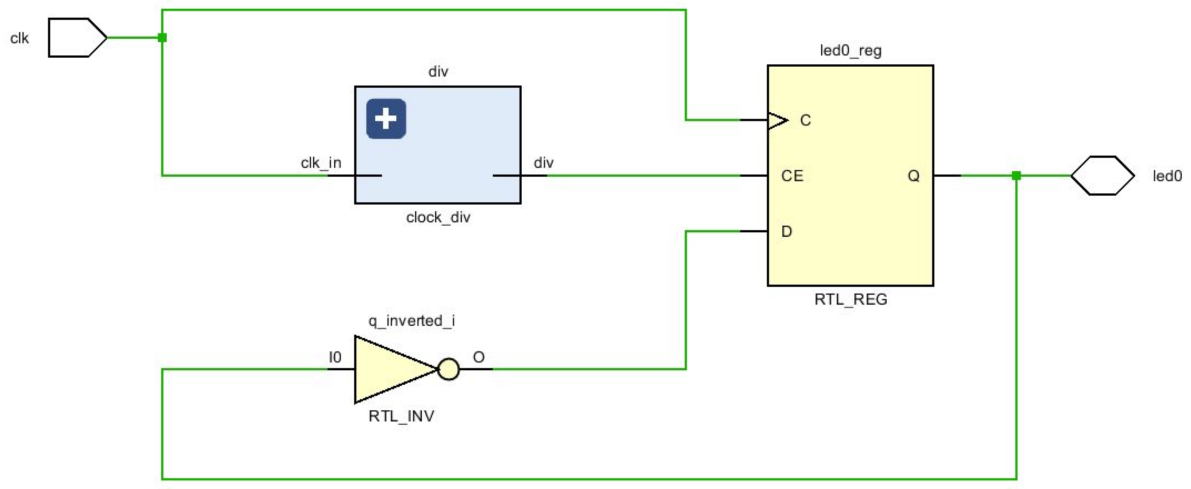
The output oscillates at 2Hz.

1b)   Top-Level Divider

Theory of Operation

This device is the top-level RTL schematic for the clock divider. It divides the clock down to 2 Hz and stores the current clock value in a register. This register allows the clock value to be held at either 1 or 0 as needed, as opposed to the clock_div module that only generates pulses.

Elaboration Schematic



Design

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

entity divider_top is
  Port ( clk : in STD_LOGIC;
       led0 : inout STD_LOGIC);
end divider_top;

architecture rtl_ckt of divider_top is
--components
component clock_div
    port(clk_in : in std_logic;
         div : out std_logic);
```

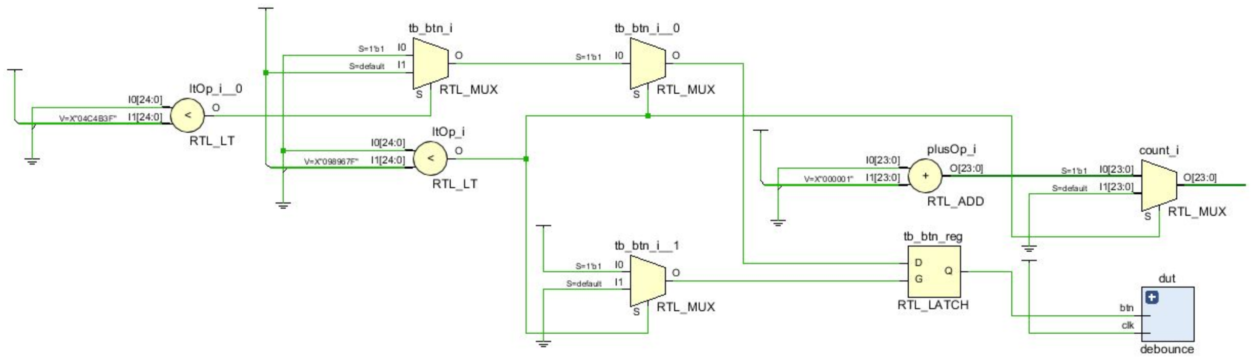
```
end component;

--signals
signal div_output : std_logic;
signal q_inverted : std_logic;

Begin
div: clock_div
port map ( clk_in => clk,
           div => div_output);

led_reg: process(clk)
Begin
    if (rising_edge(clk)) then
        if (div_output = '1') then
            led0 <= q_inverted;
        end if;
    end if;
end process;

q_inverted <= NOT(led0);

end rtl_ckt;
```

2) Debounce Circuit

Theory of Operation

This circuit digitally removes the bouncing effect from mechanical switches. When the button is pressed, it oscillates for some time between 1 and 0. This circuit waits for the button to be pressed without any nounces for 20 milliseconds before it drives the output high.

Elaboration Schematic

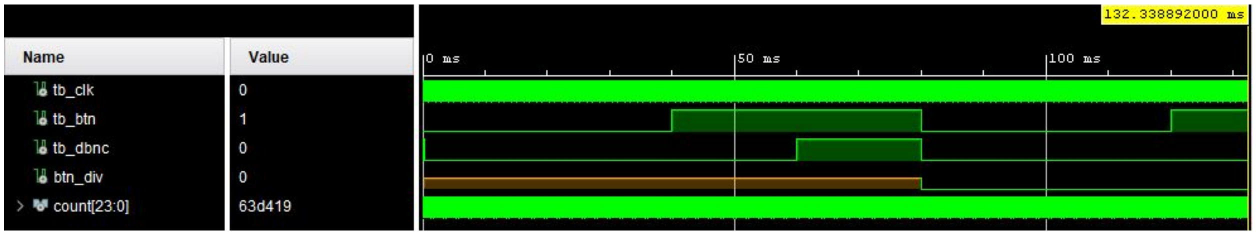


Design

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.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 shift_register : std_logic_vector (1 downto 0) := "00";
--initialize to zero
    signal register_output : std_logic;
    signal count : std_logic_vector (21 downto 0) := (others => '0'); --22
bits to get 2,500,000 clocks counted
begin

    process (clk)
    begin
        if (rising_edge(clk)) then --ripple the signal up through the shift
register
```

```
            --register_output <= shift_register(1); The shift register was not
working
            --shift_register(1) <= shift_register(0);
            --shift_register(0) <= btn; --input from the button

        end if;
    end process;

    process (clk)
    begin
        if (rising_edge(clk)) then
            if(btn = '1') then
                if (count < 2499999) then
                    count <= count + 1;
                    dbnc <= '0';
                else
                dbnc <= '1';
                end if;
            else
            count <= (others => '0');
            dbnc <= '0';
            end if;
        end if;
    end process;
```

Simulation

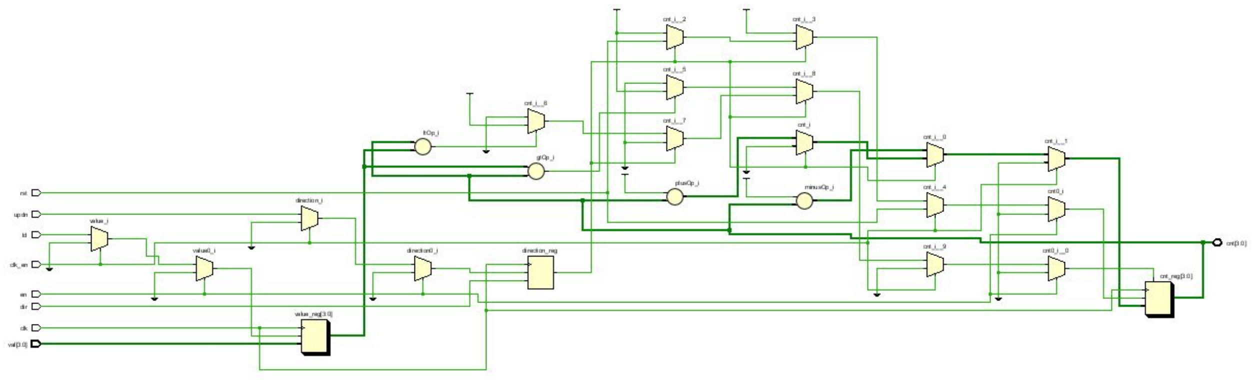


3) Fancy Counter

Theory of Operation

This component is a 4 bit counter with a small set of features. It can count in either direction and be reset. The entire device is clocked in a synchronous manner and has enable lines to activate or disable the logic.

Elaborated Schematic



Design

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

entity 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 : inout STD_LOGIC_VECTOR (3 downto 0));
end fancy_counter;

architecture Behavioral of fancy_counter is
signal direction : std_logic := '0';
signal value : std_logic_vector (3 downto 0)  := (others => '0');
begin

    process (clk)
    begin
        if (en = '1') then
```

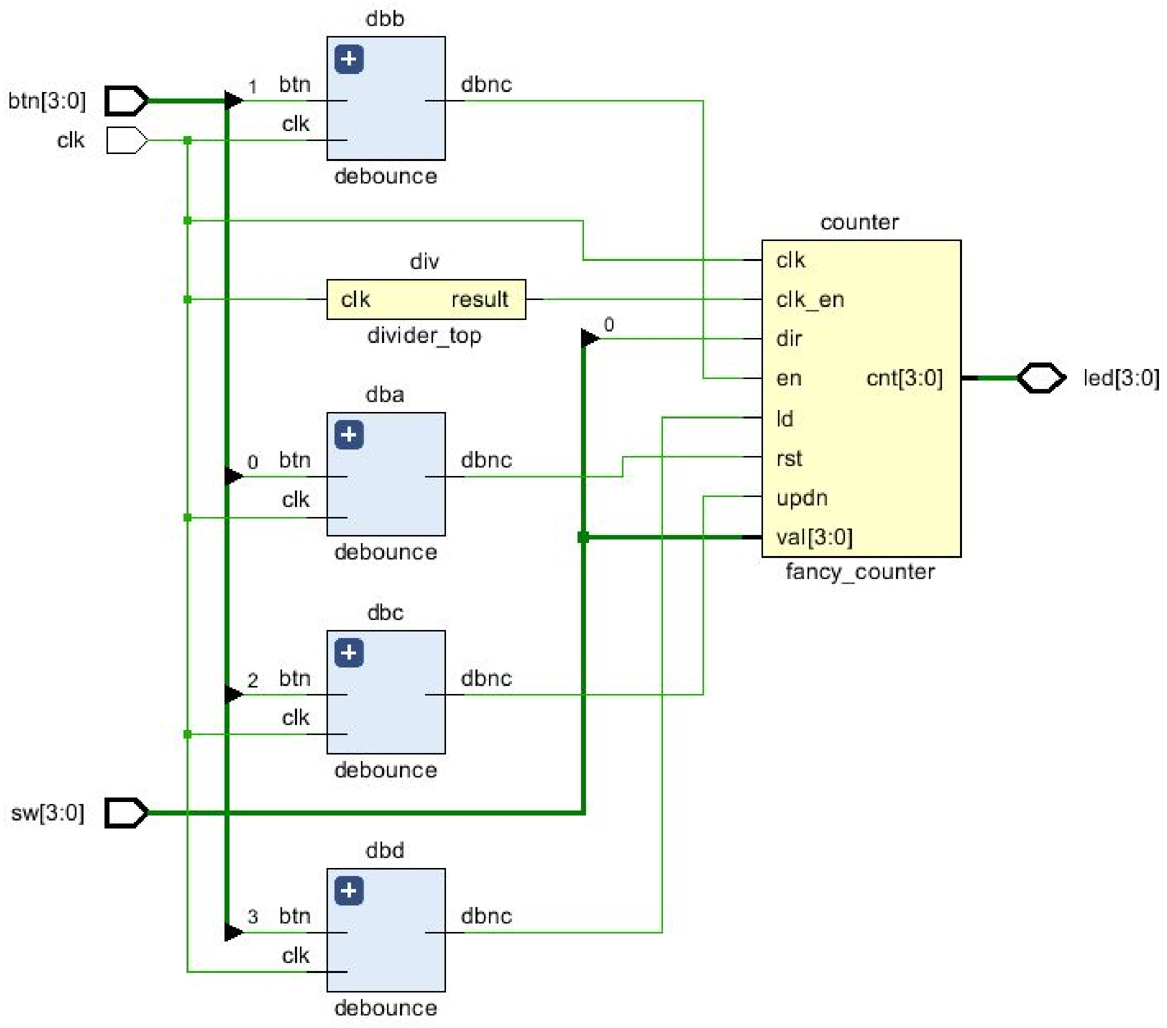
```
            if (rising_edge(clk)) then

                if (rst = '1') then
                cnt <= "0000";
                end if;

                if (clk_en = '1') then

                    if (updn = '1') then
                    direction <= dir;
                    end if;

                    if (ld = '1') then
                    value <= val;
                    end if;

                    if (direction = '1') then
                        if (cnt < value) then
                        cnt <= cnt +1;
                        else
                        cnt <= "0000";
                        end if;
                    end if;

                    if (direction = '0') then
                        if (cnt > value) then
                        cnt <= cnt -1;
                        else
                        cnt <= "0000";
                    end if;

                end if;
            end if;
        end if;
        end if;
    end process;

end Behavioral;
```

4) Full Counter

Theory of Operation

This device is a combination of all of the other devices made in the lab represented in VHDL, written in a structural style. It combines 4 unique de-bounced button inputs, a clock divider, and a fancy counter together into a single unit.

Elaborated Schematic

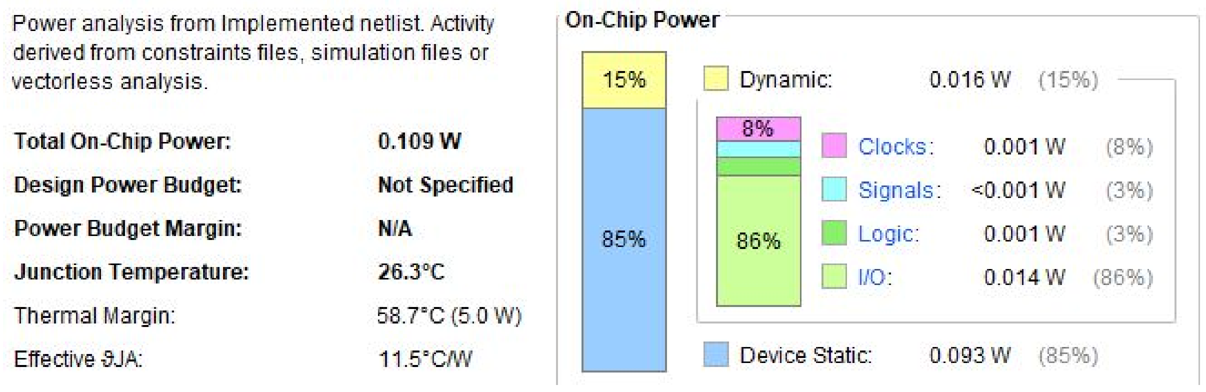


Design

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

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

architecture rtl_ckt of counter_top is
--components
component divider_top is
    port( clk : in STD_LOGIC;
          result : inout STD_LOGIC);
end component;

component debounce is
    port( clk : in STD_LOGIC;
          btn : in STD_LOGIC;
          dbnc : 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 : inout STD_LOGIC_VECTOR (3 downto 0));
end component;

--signals
signal dbnc_0 : std_logic;
signal dbnc_1 : std_logic;
signal dbnc_2 : std_logic;
signal dbnc_3 : std_logic;
signal div_clk : std_logic;

begin

dba: debounce
port map (clk => clk, btn => btn(0), dbnc => dbnc_0);

dbb: debounce
port map (clk => clk, btn => btn(1), dbnc => dbnc_1);

dbc: debounce
port map (clk => clk, btn => btn(2), dbnc => dbnc_2);

dbd: debounce
```

```
port map (clk => clk, btn => btn(3), dbnc => dbnc_3);

div: divider_top
port map (clk => clk, result => div_clk);

counter: fancy_counter
port map (clk => clk, clk_en => div_clk, dir => sw(0), en => dbnc_1, ld =>
dbnc_3, rst => dbnc_0, updn => dbnc_2, val => sw, cnt => led);

end rtl_ckt;
```

Synthesis Results

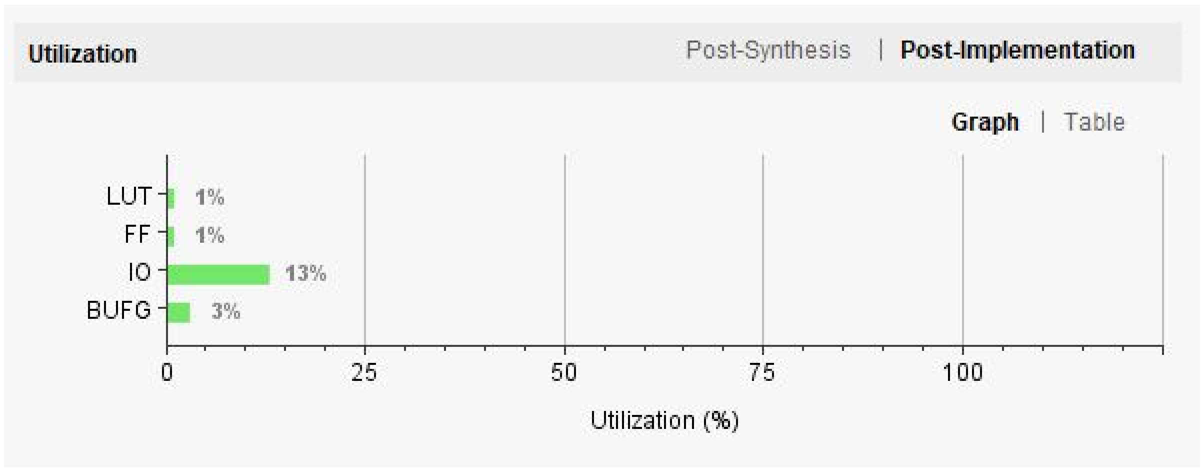


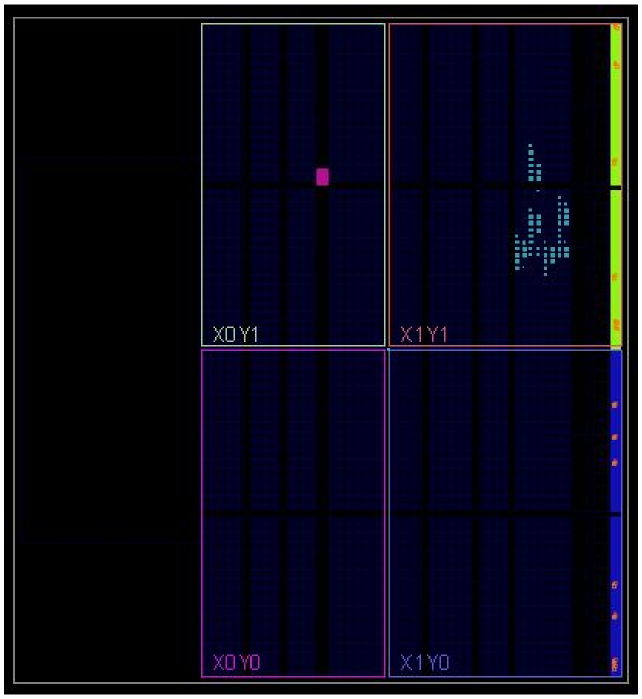
Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.109 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**26.3°C**
Thermal Margin:	58.7°C (5.0 W)
Effective ϑJA:	11.5°C/W

On-Chip Power

	Dynamic:	0.016 W	(15%)
	Clocks:	0.001 W	(8%)
	Signals:	<0.001 W	(3%)
	Logic:	0.001 W	(3%)
	I/O:	0.014 W	(86%)
	Device Static:	0.093 W	(85%)





Discussion

1.1

The division factor is 125,000,000 / 2 = 62,500,000.

1.2

26 bits are needed to count to 62,500,000.

2.1

When buttons on the Zybo are pressed, they have a value of ‘1’.

2.2

In the case of the design I used, yes, since the response to button presses is not symmetrical.

2.3

We need 2,500,000 ticks to get 20 ms of delay.

2.4

22 bits are required to count to 2,500,000.

Observations

This lab was definitely the most complex lab I've completed during my time here at Rutgers so far. I ran into scheduling issues in completing the lab on time, and have learned to pay more attention to the work assigned in the course, as it is challenging. That being said, I enjoyed completing this lab overall. VHDL is refreshing to study and I am interested in creating working models and circuits that can be tested in the real world. I learned more about building models, RTL, and assembling structures together in VHDL.

Follow-Up

I understood fully all but one component of this lab. All of the models were self-explanatory and I felt I had a good enough knowledge of VHDL to accomplish the design criteria. The one exception to this is the shift register inside the debouncing circuit. As far as I could tell my implementation of it was correct, but it always broke my debouncer. I ultimately left it out of the project, and despite being there only as a comment, the rest of the project appears to work correctly.