

Charles Owen
Clo64
189002061

Embedded Systems Lab Report 1

Submitted February 28, 2019

Table of Contents

Purpose	3
Clock Divider	4
Schematic.....	5
VHDL Code	6
Test Bench Code.....	8
Simulation Results.....	10
Implementation	10
XDC File.....	12
Clock Divider Part Two	13
VHDL Code	14
Test Bench Code.....	16
Implementation	16
XDC File.....	18
Debouncer	19
VHDL Code	20
Test Bench Code.....	22
Simulation Results.....	24
Implementation	24
Counter	25
VHDL Code	26
Test Bench Code.....	30
Implementation	30
Fancy Counter	31
VHDL Code	32
Schematic.....	35
Test Bench Code – Bonus.....	36
Implementation	39
XDC Code	40
Discussion	41
Lab Manual Questions.....	41
Section 1	41
Section 2	41
Observations/Discoveries.....	42
Questions/Follow Up	42

Purpose

Lab 1 requested the completion of multiple discrete components written in VHDL. Those multiple components would be designed and tested on their own; then, assembled into a functioning higher level system comprised of the smaller components.

The components to be designed included: a clock divider, a switch debouncer and an up-down capable counter.

The clock divider's design brief called for a device capable of taking a 125 MHz clock signal from the ZYBO board and dividing it down to a 2 Hz output signal. Correct construction of the component was then to be confirmed via a test bench driving an LED output. Once test bench confirmation was achieved the component was loaded onto the ZYBO board for physical execution.

The design brief for the switch debouncer called for a digital solution to the inherent mechanical instability of physical contacts within a switch. The potentially unstable nature of a switch necessitates the use of a sampling device to take repeated measure of the input signal and ensure the inbound signal is intentional data, and not noise. The brief requested a debouncer that samples an input signal for 20ms before sending the signal to its output port.

The third component to be constructed was a 4-bit up-down capable counter with various inputs and a bundle of LED outputs. The counter did not require a test bench.

Finally, all components were to be integrated into one top level design to stand as a fully functioning counter with a down sampled operational clock rate and debounced input buttons.

Clock Divider

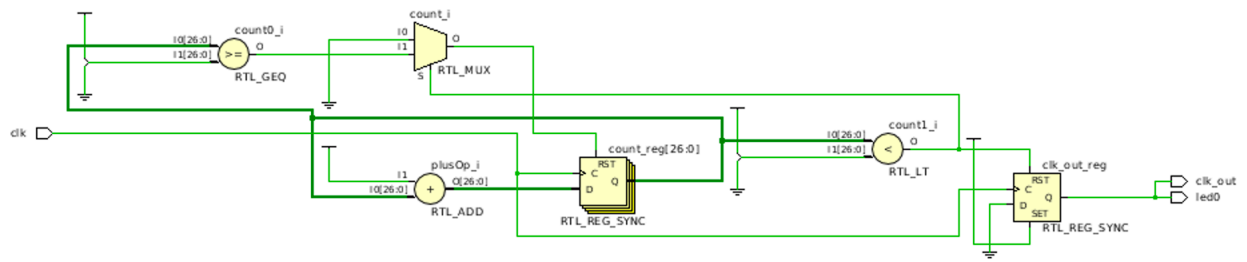
The purpose of the clock divider in this lab was two-fold. First, as a stand-alone component the divider was meant to serve as a demonstration of a quintessential building block of digital design. Second, the divider would serve as a clock-enable input for our more complex counter at the end of the lab.

The divider was to be designed to take a 125 MHz input signal and down-sample it to a 2 Hz output signal. To accomplish this task a behavioral model needed to be constructed capable of incrementing an internal value, a counter, every rising edge of the input clock. The circuit would then count cycles until a delay sufficient to produce a 2 Hz signal had been reached and output a high signal.

The input 125 MHz signal is created by a signal with period 8ns. This signal is equivalent to 125,000,000 hertz, meaning there are 125,000,000 clock cycles in one second. Our target clock cycles of 2 clock cycles per second and a period of 1 second could be achieved by counting half the period of the 125 MHz signal before outputting a high signal from the clock divider. The high signal should persist for an additional $(125 \text{ Mhz} / 2)$ clock signals before going “low” again. The signal should then repeat indefinitely.

I expect the clock divider circuit to operate to design brief specifications.

Schematic



VHDL Code

```
-- Charles Owen
-- Embedded Systems
-- Lab 1
-- Clock Divider

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

entity clock_div is
  Port (
    clk    : in std_logic; -- device common clock
    clk_out : out std_logic; -- downsampled/divided output
    led0    : out std_logic -- additional output for checking accuracy of downsample signal
  );

end clock_div;

architecture Behavioral of clock_div is

  signal count : std_logic_vector(26 downto 0) := (others => '0'); -- 26 bit bundle to accomodate
  clock cycle count

begin

  Divider_Process: process (clk) begin

    if rising_edge(clk) then

      count <= std_logic_vector(unsigned(count) + 1); -- count +1 every clock cycle

      if (unsigned(count) < 62499999) then -- if count has not reached half period of 2hz,
signal stays low

        clk_out <= '0';
        led0 <= '0';
```

```
else

    clk_out <= '1';
    led0 <= '1';

    if (unsigned(count) >= 125000000) then -- if count exceeds half period of 2hz, signal
goes high                                -- greater than sign included to catch possible count errors
        count <= (others => '0');
        led0 <= '1';

    end if;

end if;

end if;

end process;

end Behavioral;
```

Test Bench Code

```
-- Charles Owen
-- Embedded Systems
-- Lab 1
-- Clock Divider Test Bench
```

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

```
entity clock_test is
end clock_test;
```

```
architecture Behavioral of clock_test is
```

```
signal tb_clock : std_logic := '0'; -- test bench clock signal
signal clk_out : std_logic := '0'; -- test bench down sampled clock output
signal tb_led0 : std_logic := '0'; -- led output for visual confirmation of down sample signal
```

```
component clock_div
```

```
Port (
    clk : in std_logic;
    clk_out : out std_logic;
    led0 : out std_logic
);
```

```
end component;
```

```
begin
```

```
clock_process: process begin
```

```
    wait for 4ns; -- This is the first half period of the test bench clock cycle
    tb_clock <= '0'; -- clock signal low
```



```
    wait for 4ns; -- Second half of the test bench clock period
    tb_clock <= '1';

end process;

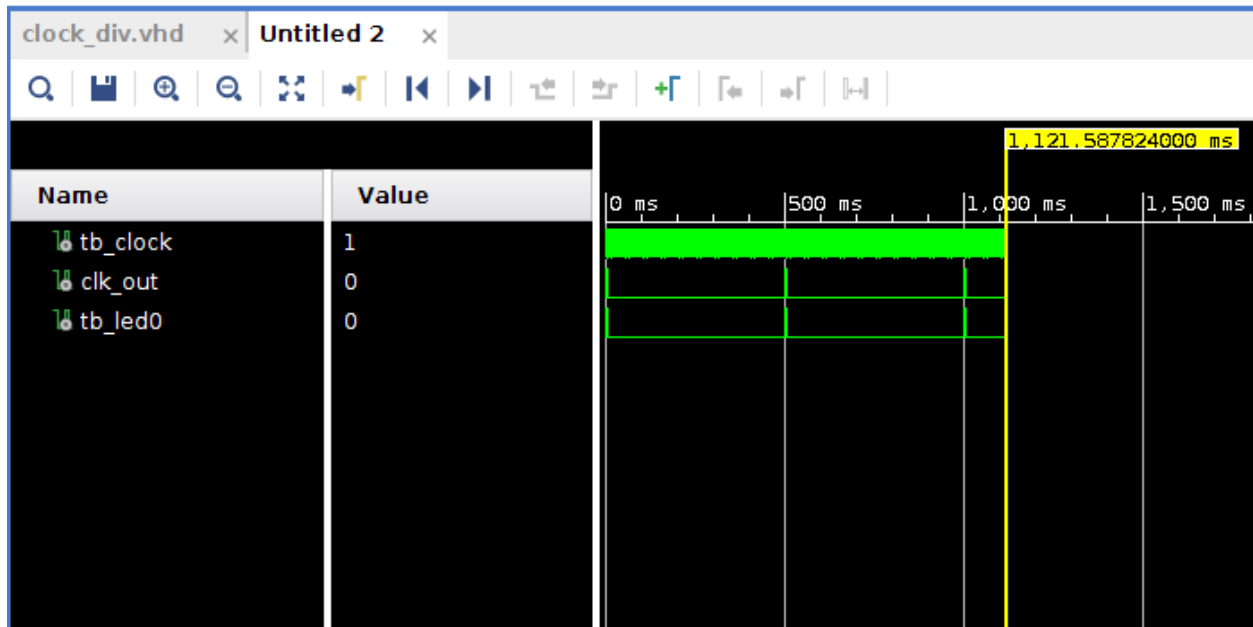
dut: clock_div -- device under test declaration
  port map(

    clk => tb_clock, -- porting general clock to tb_clock
    clk_out => clk_out,
    led0 => tb_led0

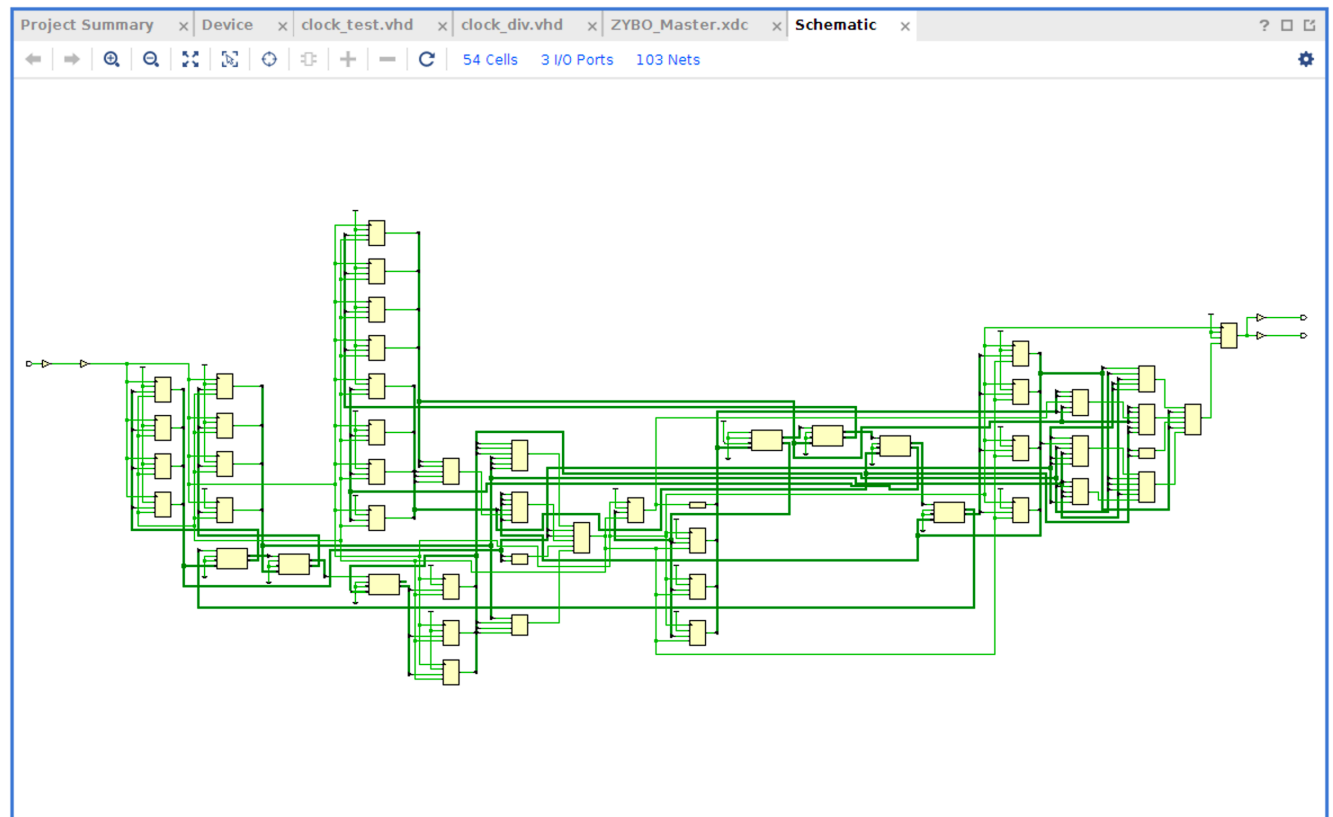
  );

end Behavioral;
```

Simulation Results



Implementation



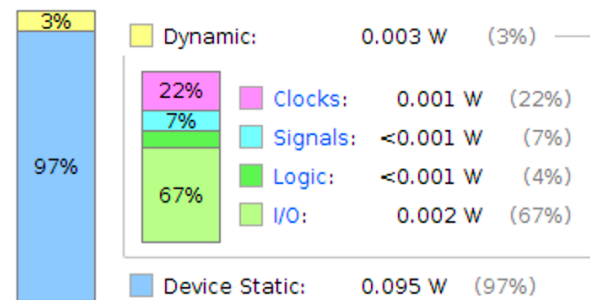
Summary

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

Total On-Chip Power: 0.098 W
Design Power Budget: Not Specified
Power Budget Margin: N/A
Junction Temperature: 26.1°C
 Thermal Margin: 58.9°C (5.0 W)
 Effective θ_{JA} : 11.5°C/W
 Power supplied to off-chip devices: 0 W
 Confidence level: Medium

[Launch Power Constraint Advisor](#) to find and fix invalid switching activity

On-Chip Power



XDC File

To implement the clock divider onto the Zybo board the led0 and clk had to be uncommented. This is to provide a clock input and led output for our circuit.

##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 { 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 Sch=LED3
```

Clock Divider Part Two

The second part of the clock divider design requested that it be connected as the clock enable on a D flip flop with inverted data coming from its output.

This implementation was a simple variant on the first design requiring minimal additional effort to create the D flip flop with VHDL.

The purpose of the top-level clock divider design was to functionally drive an external device with another discreet component. The theory was that the 2 Hz output signal of the clock divider would serve as a clock-enable signal for a D flop flop, ensuring that actuation of the flip flop would only occur at the 2 Hz clock rate.

I expect that the circuit will operate as specified by the design brief.

VHDL Code

```
--Charles Owen
-- Embedded Systems
-- Lab 1
-- Clock Divider Top Level

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

entity divider_top is
  Port(
    clk   : in std_logic;
    led0   : out std_logic -- output to drive the LED
  );

end divider_top;

architecture Behavioral of divider_top is

  component clock_div is -- Declare use of the clock divider as component
    Port (
      clk   : in std_logic;
      clk_out : out std_logic
      --led0   : out std_logic

    );

  end component;

  signal clk_divider : std_logic; -- signal to take clock divider port
  signal q           : std_logic := '0'; -- signal for d flop flop output feedback

begin

  led0 <= q; -- establishing link from led0 output to q feedback signal
```

```
div1 : clock_div -- instantiate a clock divider
port map (clk => clk, -- map the clock port
         clk_out => clk_divider -- map the downsampled output
        );

process (clk)

begin

if rising_edge(clk) then

    if (clk_divider = '1') then -- is the clk_divider is high, i.e. enabled

        q <= not(q); -- invert the signal and use as data

    end if;
end if;

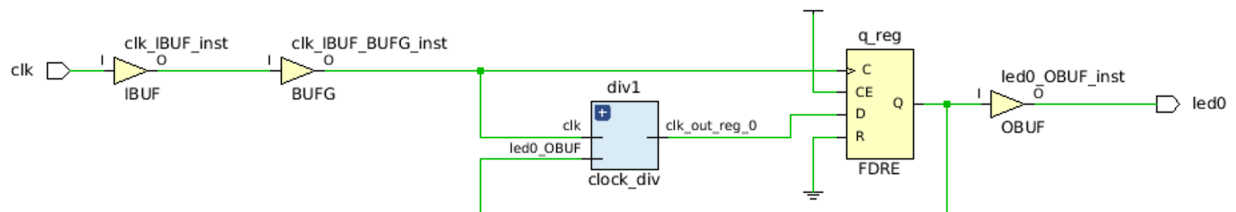
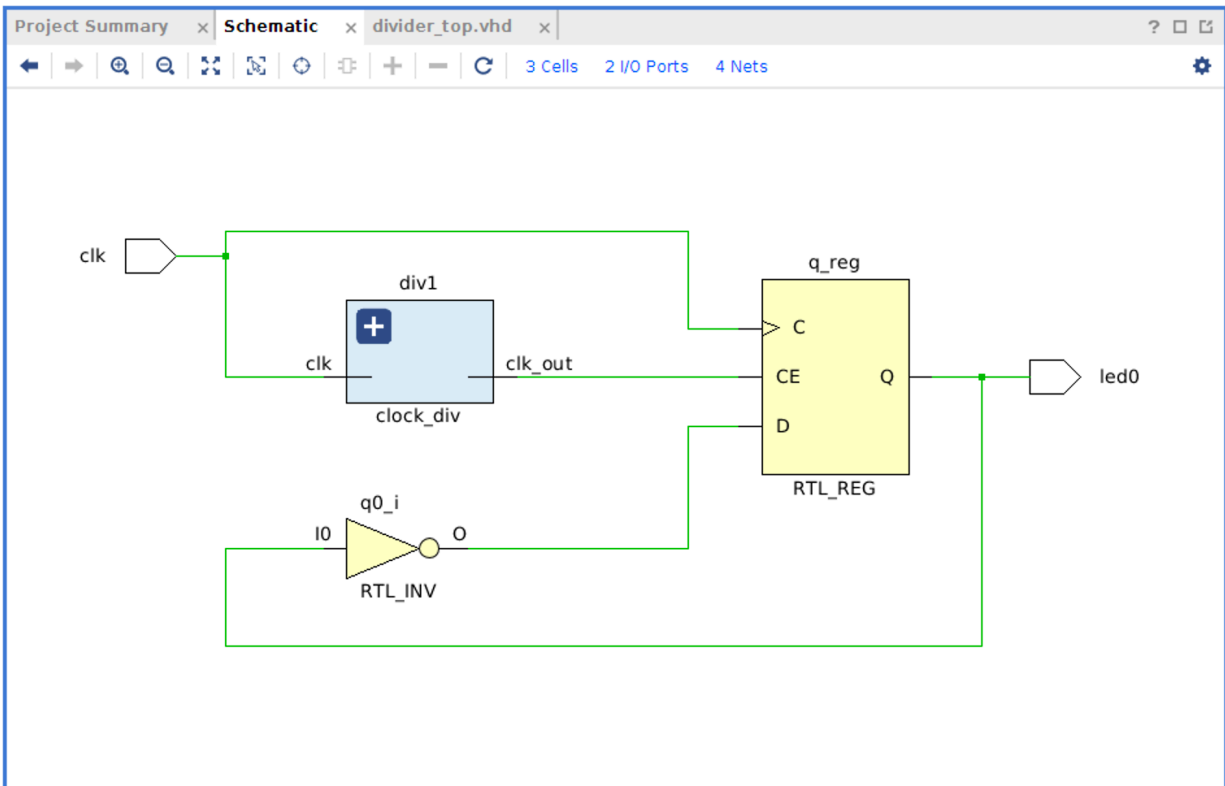
end process;

end Behavioral;
```

Test Bench Code


Test Bench not required for Clock Divider Part 2

Implementation



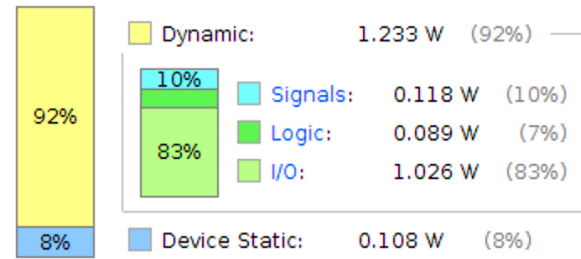
Summary

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

Total On-Chip Power: 1.341 W
Design Power Budget: Not Specified
Power Budget Margin: N/A
Junction Temperature: 40.5°C 
 Thermal Margin: 44.5°C (3.8 W)
 Effective θ_{JA} : 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



XDC File

The XDC file for Clock Divider part 2 required the same clock and led input to be uncommented. This allowed for the Zybo board to input the 125 MHz clock signal and provided an led output for the circuit.

See XDC file for clock divider above.

Debouncer

Part two of lab 1 required the construction of a switch debouncer. A debouncer is used to ensure an intended input signal is data and not mistaken noise. The nature of a mechanical switch allows possible errors of input to be transmitted to our circuit via mechanical errors. The job of our debouncer is to take the input and continuously sample the signal for a period. Once the period, our button press threshold, has been satisfied our debouncer should send an output signal to whatever device is waiting for a button press.

Our debouncer will sample its input signal for 20 ms before forwarding the button press to its output port. To ensure a sample of 20 ms, a count will be initiated on the rising edge of the the first button push.

The 20 ms delay will require 2500000 cycles to be counted before outputting the last sampled value to the button.

VHDL Code

```
-- Charles Owen
-- Embedded Systems
-- Lab 1
-- Debounce

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

entity debounce_two is
    Port(
        btn, clk    : in std_logic; -- button and clock input
        dbnc        : out std_logic -- output for our debounced signal
    );
end debounce_two;

architecture Behavioral of debounce_two is

    signal reg_A      : std_logic_vector( 1 downto 0); -- input register to sample inbound button
    press
    signal out_to_dbnc : std_logic := '0'; -- intermediate signal to leave register and interface wiht
    debounce output
    signal count       : std_logic_vector( 26 downto 0) := (others => '0'); -- counter for debounce
    time

begin

    dbnc <= out_to_dbnc;

    debounce_update: process (CLK) -- process to update register
    begin

        if rising_edge(CLK) then

            reg_A(0) <= btn; -- update register to the value of input button
            reg_A(1) <= reg_A(0); -- update second bit of register to first bit input

        end if;
    end process;

end process;
```

```

rising_button:  process (CLK) -- process to read second bit of register and count
begin          -- until 20 ms

    if rising_edge(CLK) then

        if (btn = '1') then -- if button inpt is 1, continue process, of 0, no need to count

            if (unsigned(count) < 2500000) then -- if less than 2500000 cycles continue

                out_to_dbnc <= '0'; -- output stays low

                if (reg_A(1) = '1') then -- if register bit 2 sample still high, count up one

                    count <= std_logic_vector(unsigned(count) + 1);

                else -- if register value goes low, reset count to 0

                    count <= (others => '0');

                end if;

            else

                out_to_dbnc <= reg_A(1); -- if above count threshold output value of
register 2nd bit
                                -- to debounce output
            end if;

            else -- else to go with if btn = 1, else keep output low and count is refreshed to
0

                out_to_dbnc <= '0';
                count <= (others => '0');

            end if;

        end if;

    end process;

end Behavioral;

```

Test Bench Code

```
-- Charles Owen
-- Embedded Systems
-- Lab 1
-- Debounce TB
```

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

```
entity debounce_TB is
end debounce_TB;
```

```
architecture Behavioral of debounce_TB is
```

```
signal tb_clock    : std_logic := '0'; -- intermediate clock signal for port mapping
signal tb_dbnc     : std_logic := '0'; -- intermediate debounce output signal for port mapping
signal tb_btn      : std_logic := '0'; -- intermediate button input signal for port mapping
```

```
component debounce_two is -- declare clock divider componenet
```

```
    Port(
        btn, clk    : in std_logic;
        dbnc        : out std_logic
```

```
    );
end component;
```

```
begin
```

```
clock_process: process begin -- simulate 125 MHz clock
```

```
    wait for 4ns; -- half period
    tb_clock <= '0';
```

```
    wait for 4ns; -- half period
    tb_clock <= '1';
```

```
end process;
```

```
button_process: process begin -- simulate button press
```

```
    wait for 100000000ns;
```

```
    tb_btn <= '0';
```

```
    wait for 100000000ns;
```

```
    tb_btn <= '1';
```

```
end process;
```

```
dut: debounce_two -- port map for the device under test, debouncer
```

```
    port map(
```

```
        clk => tb_clock,
```

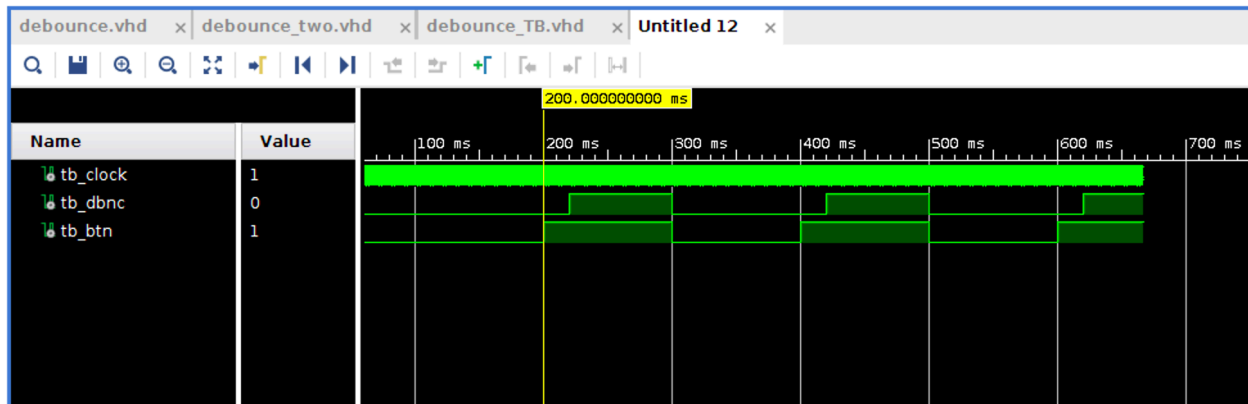
```
        btn => tb_btn,
```

```
        dbnc => tb_dbnc
```

```
    );
```

```
end Behavioral;
```

Simulation Results



Implementation

No implementation required for the debouncer

Counter

The last discrete component requested by the design brief is the counter. The counter can count either up or down to a user input value. The counter is required to have the following inputs:

clk
clk_en
dir
en
ld
rst
updn
val[0:3]

The counter will have one bundle of outputs cnt[3:0].

Operational constraints for the clock include the following:

Nothing in the circuit should change unless en is 1. If en is 1, nothing can change unless clk_en is also 1, except for rst. rst will reset the count back to 0000. The counter counts either up or down depending on the value in the direction register. The direction register is updated when updn is set to 1.

To achieve these constraints a series of nested if statements will be used as necessary. Additionally, simplification and clarity will be achieved by separating processes containing discrete signal assignments, ie. signal assignments used in one if statement only, into their own process.

VHDL Code

```

-- Charles Owen
-- Embedded Systems
-- Lab 1
-- Fancy Counter

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

entity fancy_counter_again is
    Port(
        clk, clk_en, dir  : in std_logic; -- clock .clock enable and direction inputs
        en, ld, rst       : in std_logic; -- enable, load, and reset
        updn              : in std_logic; -- updown input
        value_in          : in std_logic_vector(3 downto 0); -- value in is the new input count until
value from user
        cnt               : out std_logic_vector(3 downto 0) -- cnt is the output of our conter's value
    );
end fancy_counter_again;

architecture Behavioral of fancy_counter_again is

    signal direction_register : std_logic := '0'; -- register to hold direction up or down
    signal count              : std_logic_vector (3 downto 0) := (others => '0'); --temporary count signal
    signal value              : std_logic_vector (3 downto 0) := (others => '1'); -- temporary value signal

begin

    cnt <= count; -- send our intermediate count signal to the count output

    count_pro: process (CLK)
    begin

        if rising_edge(CLK) then -- now we catch our main clock rising edge

            if (EN = '1') then -- make sure enable is set to 1, otherwise no changes in cicrcuit

                if (rst = '1') then -- is reset is 1 we reset value of count

                    count <= "0000";

```

```

end if;

if (CLK_EN = '1') then -- if intermediate clock signal is 1, then we count

    if (direction_register = '0') then -- direction is 0 and

        if (count < value) then -- count is less than user input target value, we increment

            count <= std_logic_vector( unsigned(count) + 1 );

        else

            count <= "0000"; -- if count reaches our target value, rolls to 0000

        end if;

    end if;

    if (direction_register = '1') then

        if (value > "0000") then -- for counting down, if not at zero yet

            count <= std_logic_vector(unsigned(count) - 1);

            if (count = "0000") then

                count <= value; -- after target value roll to value

            end if;

        end if;

    end if;

end if;

end if;

end if;

```

```

end process;

ld_process: process (CLK) -- load gets its own process since no competing signal assignments
begin
    -- in other if statement

    if rising_edge(CLK) then

        if (CLK_EN = '1') then -- following three lines, ensure enable criteria met

            if (EN = '1') then

                if (ld = '1') then

                    value <= value_in; -- set value to input value from user

                end if;

            end if;

        end if;

    end if;

end process;

direc_proc: process (CLK) -- direction also gets its own process since no other if statement
begin
    -- competes with signal assignment

    if rising_edge(CLK) then

        if (CLK_EN = '1') then -- check enable criteria

            if (EN = '1') then

                if (updn = '1') then

                    direction_register <= dir; -- set direction input from user

                end if;

            end if;

        end if;

    end if;

end process;

```

end if;

end process;

end Behavioral;

Test Bench Code

No test bench required for counter component .

Implementation

No implementation required for counter component.

Fancy Counter

The final portion of the lab brief tasks us with bringing together all the previously constructed and tested elements into a final top level design. The functionality of the top-level design is such that all components will receive a common clock signal, the debounce components will be conduits through which the mechanical buttons on the Zybo board input data to inputs of the counter component, switches on the Zybo board will serve to feed a “count to” value to the counter and the clock divider component will feed the clock enable input on the counter a 2 Hz signal in sync with the common clock . Finally, the led lights on the ZYBO board will serve as visual output signals of the counter.

All components have been previously test benched, or implemented on the Zybo board. Therefore, the functionality should work as intended by the design brief.

VHDL Code

```
-- Charles Owen
-- Embedded Systems
-- Lab 1
-- The top level design, the design in charge, the ultimate primo numeral uno master plan!

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

entity fancy_top_level is

port(

    btn    : in std_logic_vector(3 downto 0); -- button inputs for top level
    clk    : in std_logic;                    -- clock input for top level
    sw     : in std_logic_vector(3 downto 0); -- switch inputs for top level
    led    : out std_logic_vector(3 downto 0) -- led outputs for top level

);

end fancy_top_level;

architecture Behavioral of fancy_top_level is

    component clock_div is -- declare clock_divider component
        Port (
            clk    : in std_logic;
            clk_out : out std_logic
        );
    end component;

    component debounce_two is -- debouncer component for each switch
        Port(
            btn, clk    : in std_logic;
            dbnc        : out std_logic
        );
    end component;
```


component fancy_counter_again is --- fancy counter component

```

    Port(
        clk, clk_en, dir  : in std_logic;
        en, ld, rst       : in std_logic;
        updn              : in std_logic;
        value_in          : in std_logic_vector(3 downto 0);
        cnt               : out std_logic_vector(3 downto 0)
    );
end component;
```

--intermediate signals, used for connecting components within top level

```

signal debounce_int  : std_logic_vector(3 downto 0);
signal clk_en_int    : std_logic;
signal dir_int       : std_logic;
signal en_int        : std_logic;
signal ld_int        : std_logic;
signal updn_int      : std_logic;
```

begin

— instantiate the four switch debouncers

```

button1: debounce_two
    port map(
        btn => btn(0),
        clk => clk,
        dbnc => debounce_int(0)
    );
```

```

button2: debounce_two
    port map(
        btn => btn(1),
        clk => clk,
        dbnc => debounce_int(1)
    );
```

```

button3: debounce_two
  port map(
    btn => btn(2),
    clk => clk,
    dbnc => debounce_int(2)

  );

```

```

button4: debounce_two
  port map(
    btn => btn(3),
    clk => clk,
    dbnc => debounce_int(3)

  );

```

-- instantiate the counter

```

counter: fancy_counter_again
  port map(
    rst => debounce_int(0),
    clk => clk,
    clk_en => clk_en_int,
    dir => sw(0),
    en => debounce_int(1),
    ld => debounce_int(3),
    updn => debounce_int(2),
    value_in => sw,
    cnt => led

  );

```

-- instantiate the clock divider

```

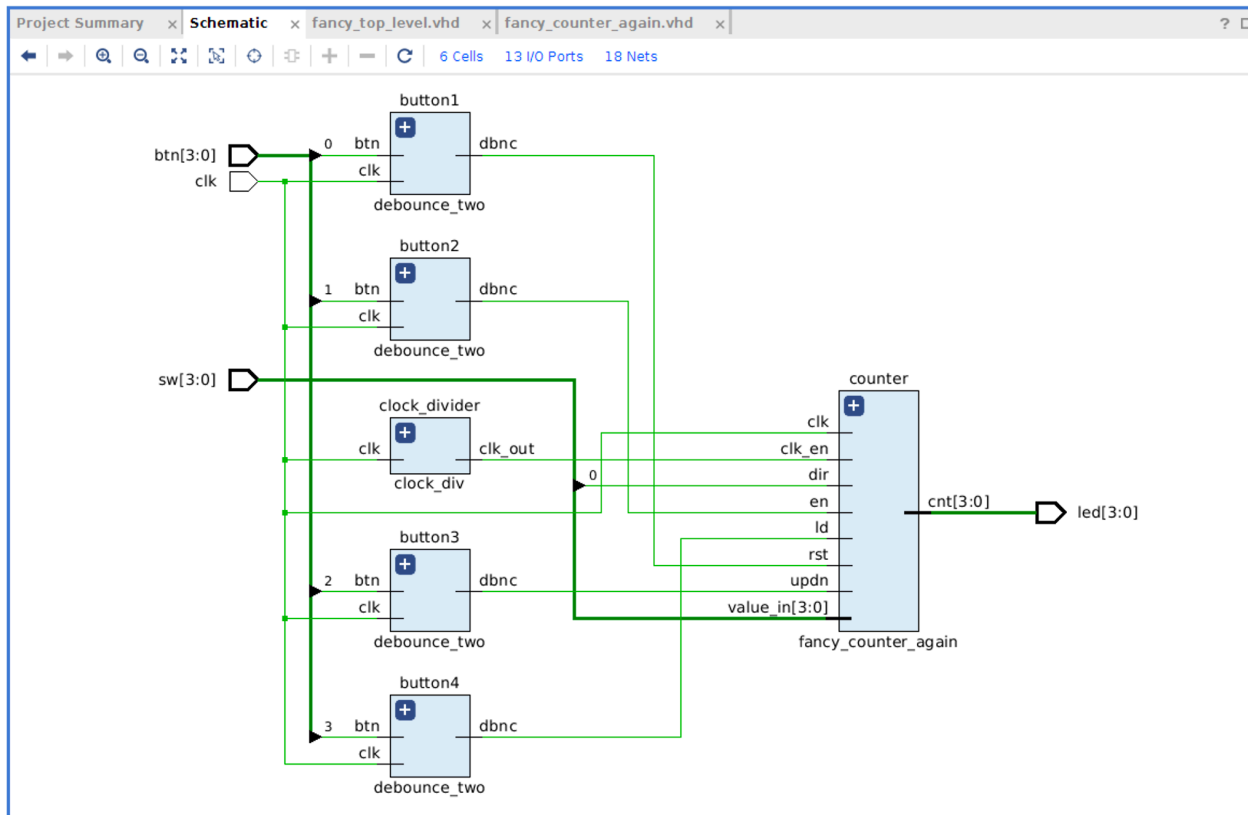
clock_divider: clock_div
  port map(
    clk => clk,
    clk_out => clk_en_int

  );

```

end Behavioral;

Schematic



Test Bench Code – Bonus

```
-- Charles Owen
-- Embedded Systems
-- Test Bench for the WHOLE ENCHILADA!
-- Cross your fingers, hide your kids, hide your wives
-- 'Bout to get crazy up in here'
```

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

```
entity fancy_top_TB is
```

```
end fancy_top_TB;
```

```
architecture Behavioral of fancy_top_TB is
```

```
signal clk_tb  : std_logic := '0';
signal sw_tb   : std_logic_vector(3 downto 0) := (others => '0');
signal btn_tb  : std_logic_vector(3 downto 0) := (others => '0');
signal led_tb  : std_logic_vector(3 downto 0) := (others => '0');
signal divider_light: std_logic;
```

```
component fancy_top_level is
```

```
port(
```

```
btn   : in std_logic_vector(3 downto 0);
clk   : in std_logic;
sw    : in std_logic_vector(3 downto 0);
led   : out std_logic_vector(3 downto 0);
divider_light : out std_logic
```

```
);
```

```
end component;
```

```
begin

clk_proc: process
    begin

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

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

    end process;

load_proc: process
    begin

        wait for 550000000ns;
        btn_tb(3) <= '1';

        wait for 21000000ns;
        btn_tb(3) <= '0';

        wait for 500000000ns;
        wait for 500000000ns;

    end process;

rst_proc: process
    begin

        wait for 650000000ns;
        btn_tb(0) <= '1';

        wait for 21000000ns;
        btn_tb(0) <= '0';

    end process;

-- buttons setting for enable on, direction 1,

--btn_tb(0) <= '0'; -- rst = 0
```

```
btn_tb(1) <= '1'; -- en = 1  
btn_tb(2) <= '0'; -- updn = 0
```

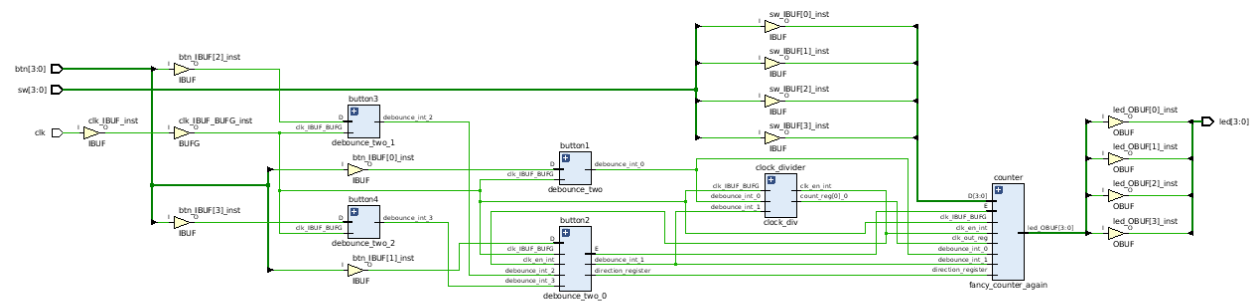
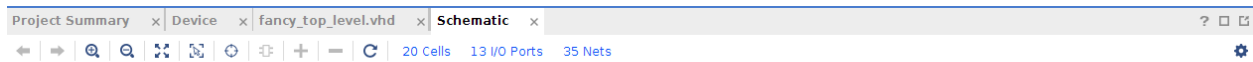
```
-- switches off for now  
sw_tb(0) <= '1';  
sw_tb(1) <= '0';  
sw_tb(2) <= '0';  
sw_tb(3) <= '0';
```

```
dut: fancy_top_level
```

```
  port map(  
  
    btn => btn_tb,  
    clk => clk_tb,  
    sw  => sw_tb,  
    led => led_tb,  
    divider_light => divider_light  
  
  );
```

```
end Behavioral;
```

Implementation



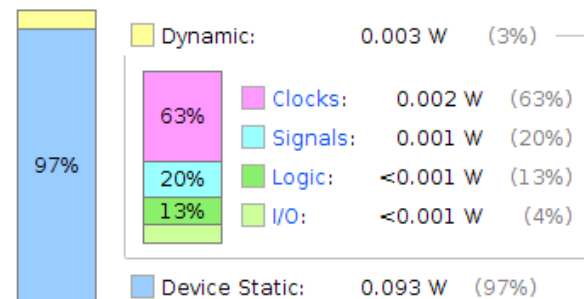
Summary

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

Total On-Chip Power:	0.096 W
Design Power Budget:	Not Specified
Power Budget Margin:	N/A
Junction Temperature:	26.1°C
Thermal Margin:	73.9°C (6.2 W)
Effective θ_{JA} :	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



XDC Code

Our XDC needs to be modified in such a way as to give us access to the Zybo board's LED lights, switch inputs and button inputs. In order to accomplish this we simply uncomment the appropriate lines and ensure the naming convention matches our those our components use.

##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
```


Discussion

Lab Manual Questions

Section 1

1.1: We need to divide our signal by 62.5 Mhz. Which means we just wait 62.5E6 cycles to express out output signal from the divider.

1.2: We will require 26 bits to store 62.5E6 count cycles.

Section 2

2.1: The value of the pressed button on the Zybo board is 1.

2.2: (Optional) If the value were opposite we would have to change our instructions to be sensitive to a 0 input and not a 1 input.

2.3: To achieve our 20 ms debounce wait time we'll need to count for 25E5 cycles.

2.4: We would require 22 bits to count to 25E5 cycles

Observations/Discoveries

This lab, after it was all over, was as rewarding as it was challenging. The challenge was a big one. I went from knowing virtually no VHDL, save small component synthesis for homework, to constructing and combining multiple discrete components into one functioning system. The process took every last drop of time allotted for the project.

The biggest challenges for me came in the form of poor quality of description in the lab's design brief. I struggled to understand exactly what kind of functionality was being requested of the clock divider. The lab brief seemed to indicate that a 50% duty cycle clock divider should be produced since there was no mention of altering the characteristics of the input wave, only that it should be divided. This led to a cascade of unnecessary complexity in later components relying on the clock divider signal.

Additionally, I had very little information on linking components together into one top level design. Many times, I ran into issues related to incompatibility between related signals I was attempting to connect to one another. However, a little elbow grease smoothed these issues.

I discovered that I very much enjoy writing in VHDL. I can't describe how interesting it is to see a behavioral model interpreted by the synthesizer into the building blocks that I learned in DLD. It's wicked fun.

I discovered how to write a test bench that effectively communicates with the component I'm interested in testing. In fact, I wasted a ton of time, happily, just manipulating the test bench in different ways to see what my circuit would do.

Questions/Follow Up

I feel like I have a solid grasp on the fundamental operation of all circuit components. But I do feel, or rather know, that I could have more efficiently written the necessary code. At times I would grow frustrated and begin hacking at certain pieces of code without having a clear game plan in mind. Almost universally my frustration would only grow until I decided to give up for the night. Upon returning to the problem, with a clear mindset and pen and paper on which to write out my thoughts, the problems I was facing typically yielded.