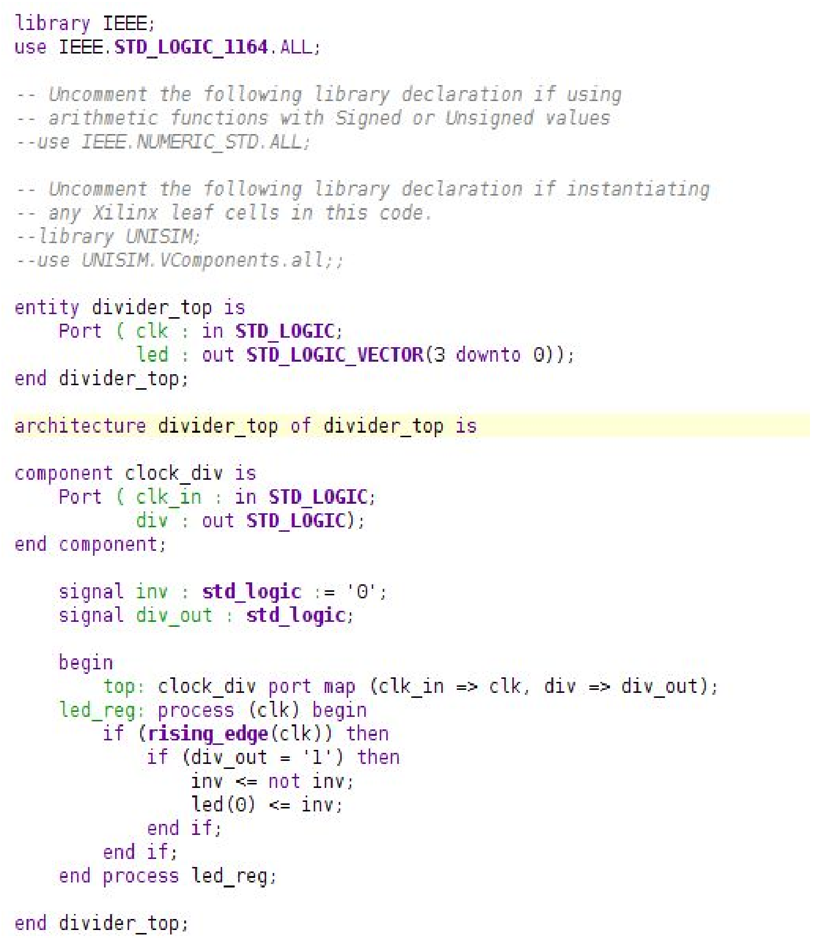
- Title
    - Embedded Systems
    - Lab Report #1
    - Alexander Riveron
    - 2/28/19
- Purpose
    - To get the student acquainted with the clock frequency of the Zybo, and create systems that work at any desired frequency

1. My Clock is Just Right

Theory:

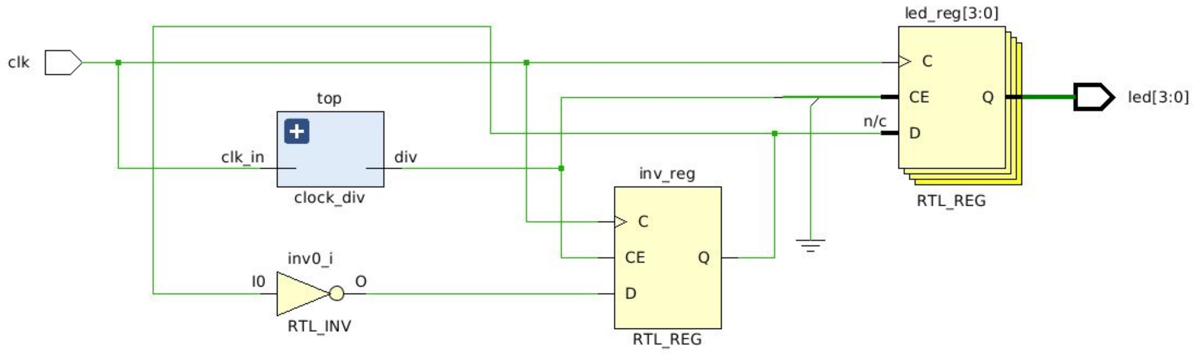
The clock divider is a simple circuit that takes a clock's high frequency input, and outputs a clock running at a slower frequency. In theory, the clock divider should output a square wave just like the clock generated by the zybo, but for the circuit designed in part 3, a pulse is more convenient for getting a counter to work properly

VHDL:

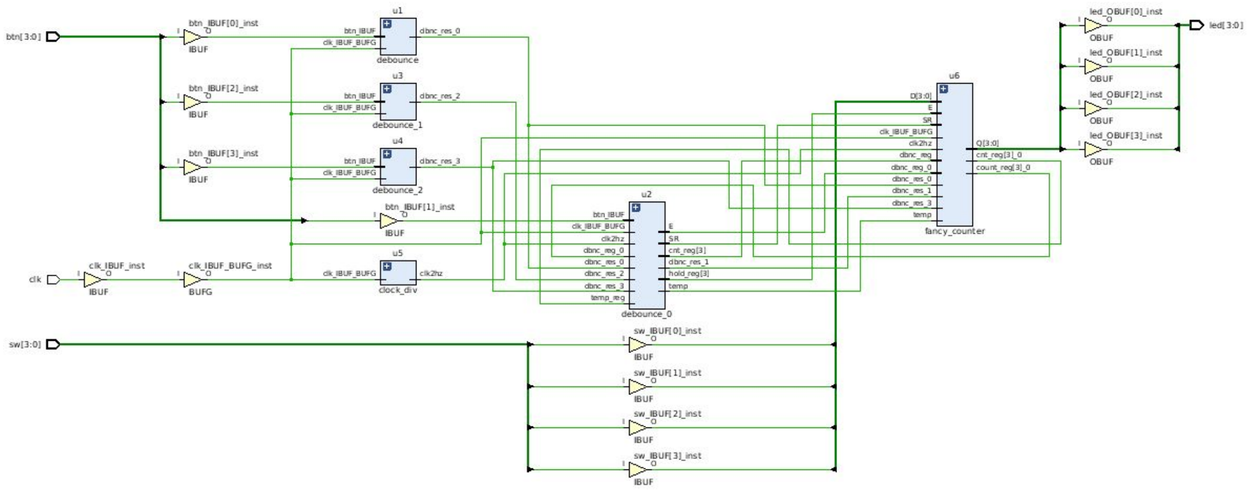


```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- 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 divider_top is
    Port ( clk : in STD_LOGIC;
           led : out STD_LOGIC_VECTOR(3 downto 0));
end divider_top;

architecture divider_top of divider_top is

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

    signal inv : std_logic := '0';
    signal div_out : std_logic;

    begin
        top: clock_div port map (clk_in => clk, div => div_out);
    led_reg: process (clk) begin
        if (rising_edge(clk)) then
            if (div_out = '1') then
                inv <= not inv;
                led(0) <= inv;
            end if;
        end if;
    end process led_reg;

end divider_top;
```

Elaborated Design:



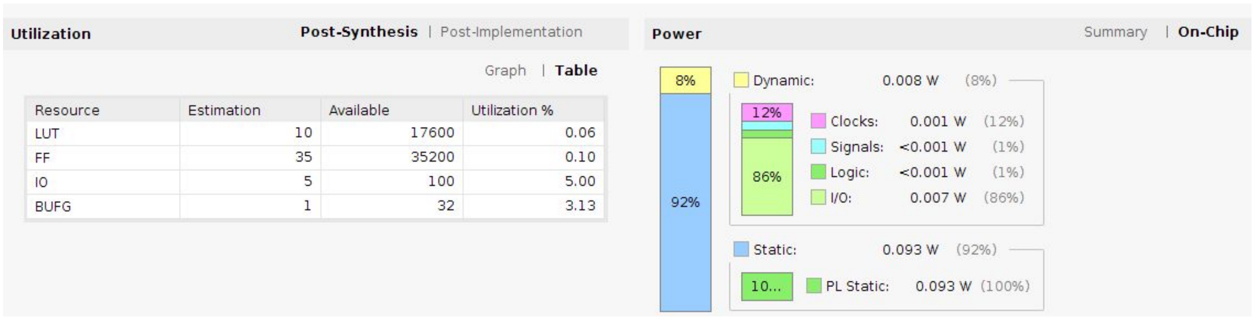
Synthesis Schematic



XDC

```
##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
```

Project Summary:Project Summary:



Utilization   Post-Synthesis | Post-Implementation

Graph | Table

Resource	Estimation	Available	Utilization %
LUT	10	17600	0.06
FF	35	35200	0.10
IO	5	100	5.00
BUFG	1	32	3.13

Power   Summary | On-Chip

Dynamic:	0.008 W	(8%)
Clocks:	0.001 W	(12%)
Signals:	<0.001 W	(1%)
Logic:	<0.001 W	(1%)
I/O:	0.007 W	(86%)
Static:	0.093 W	(92%)
PL Static:	0.093 W	(100%)

Questions:

1) 62.5 MHz
2) 28 bits

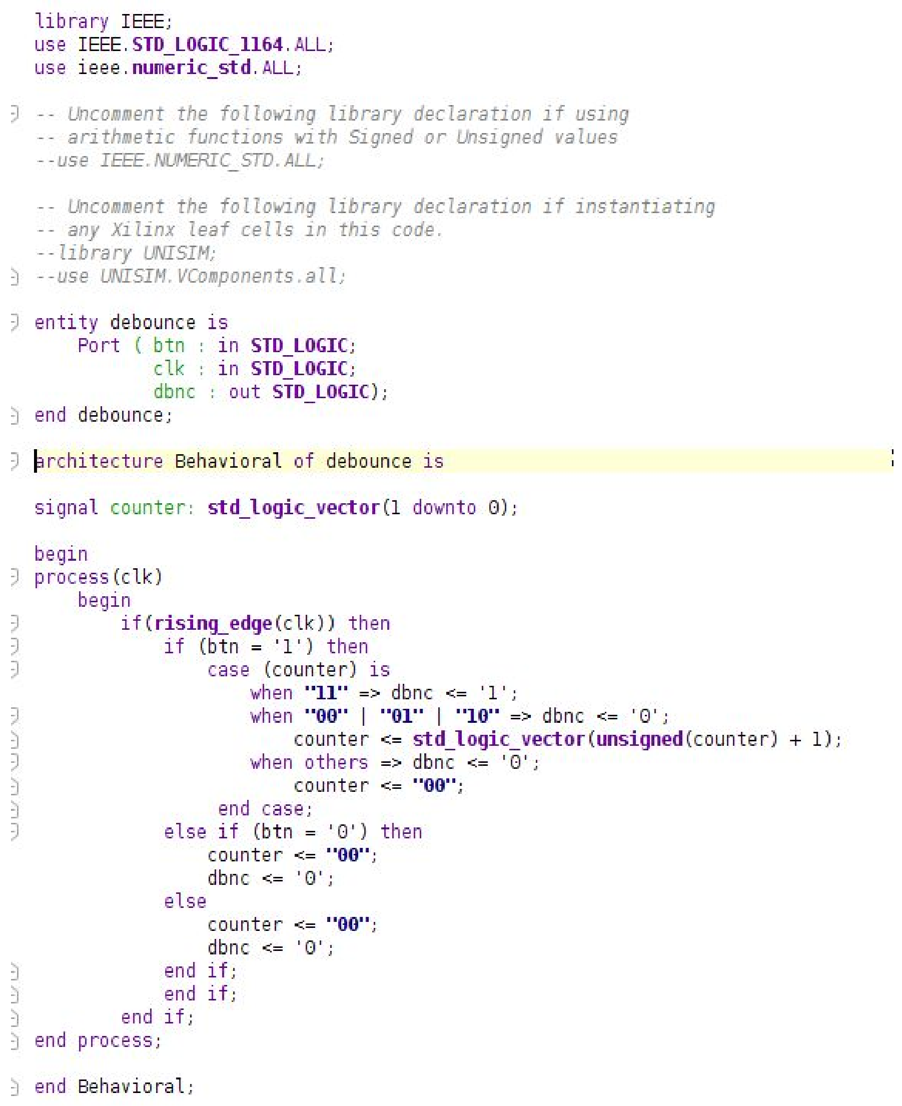
Observations: Intuitively to change the period of a square wave to a certain faction would just require the user to divide one frequency to the other to understand the total cycles needed to be counted before changing the state of the divided clock.

2.Bouncy Buttons

Theory:

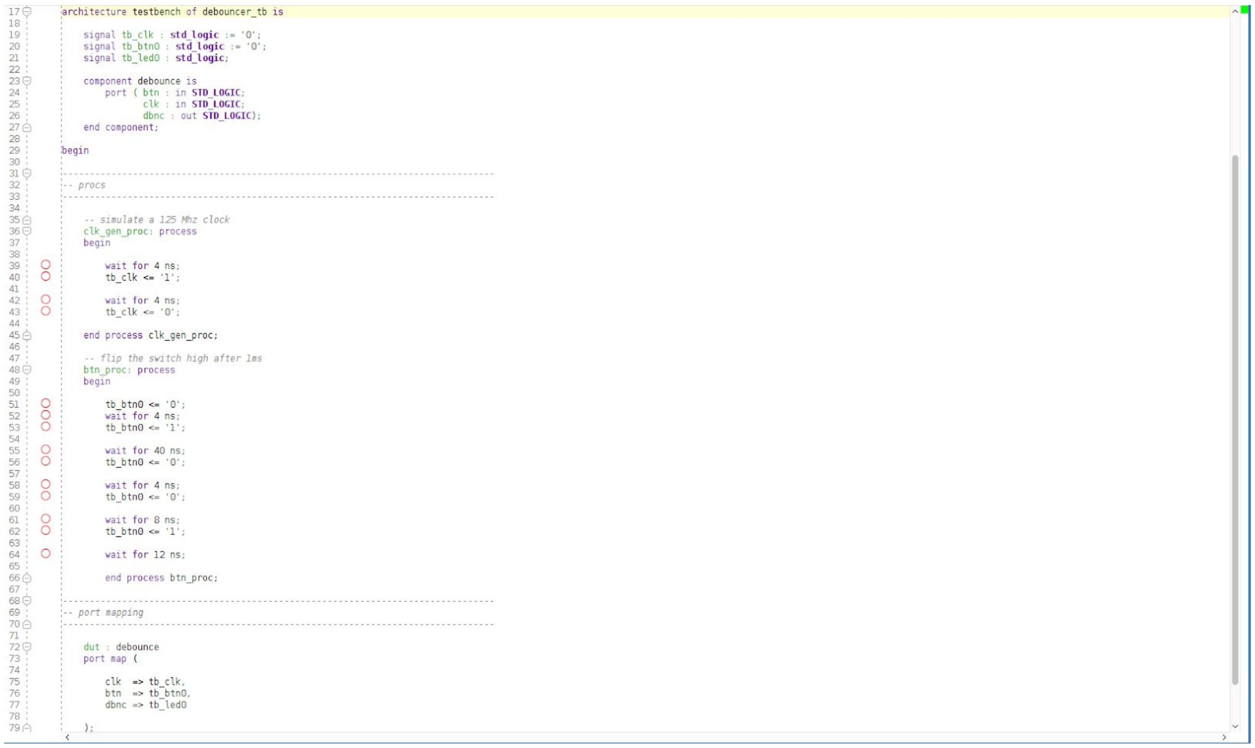
This circuit receives 2 inputs: a clock, and a button. If the button is held for 4 cycles of the clock, then the circuit outputs a ‘1’, The button is required to be held for 4 cycles instead of one to prevent uncertainties when the button is held between a ‘lo’ and a ‘hi’ value.

VHDL:



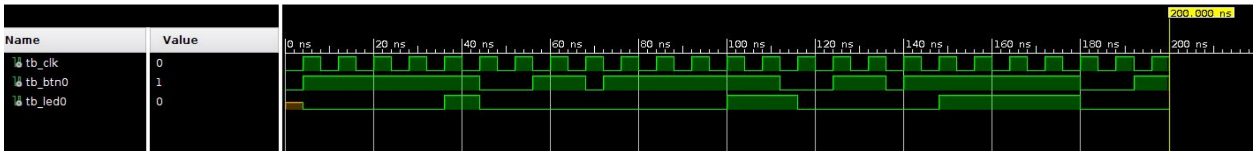
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.ALL;

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

architecture Behavioral of debounce is

signal counter: std_logic_vector(1 downto 0);

begin
process(clk)
    begin
        if(rising_edge(clk)) then
            if (btn = '1') then
                case (counter) is
                    when "11" => dbnc <= '1';
                    when "00" | "01" | "10" => dbnc <= '0';
                        counter <= std_logic_vector(unsigned(counter) + 1);
                    when others => dbnc <= '0';
                        counter <= "00";
                 end case;
            else if (btn = '0') then
                counter <= "00";
                dbnc <= '0';
            else
                counter <= "00";
                dbnc <= '0';
            end if;
            end if;
        end if;
end process;

end Behavioral;
```

Testbench VHDL:



```vhdl
architecture testbench of debouncer_tb is

    signal tb_clk : std_logic := '0';
    signal tb_btn0 : std_logic := '0';
    signal tb_led0 : std_logic;

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

begin

-------------------------------------------------------------------------------
-- procs
-------------------------------------------------------------------------------


    -- 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;

    -- flip the switch high after 1ms
    btn_proc: process
    begin

        tb_btn0 <= '0';
        wait for 4 ns;
        tb_btn0 <= '1';

        wait for 40 ns;
        tb_btn0 <= '0';

        wait for 4 ns;
        tb_btn0 <= '0';

        wait for 8 ns;
        tb_btn0 <= '1';

        wait for 12 ns;

        end process btn_proc;

-------------------------------------------------------------------------------
-- port mapping
-------------------------------------------------------------------------------


    dut : debounce
    port map (

        clk  => tb_clk,
        btn  => tb_btn0,
        dbnc => tb_led0

    );
```

Simulation:



Questions:

1) ‘1’
2) omit
3) 2500000
4) 22

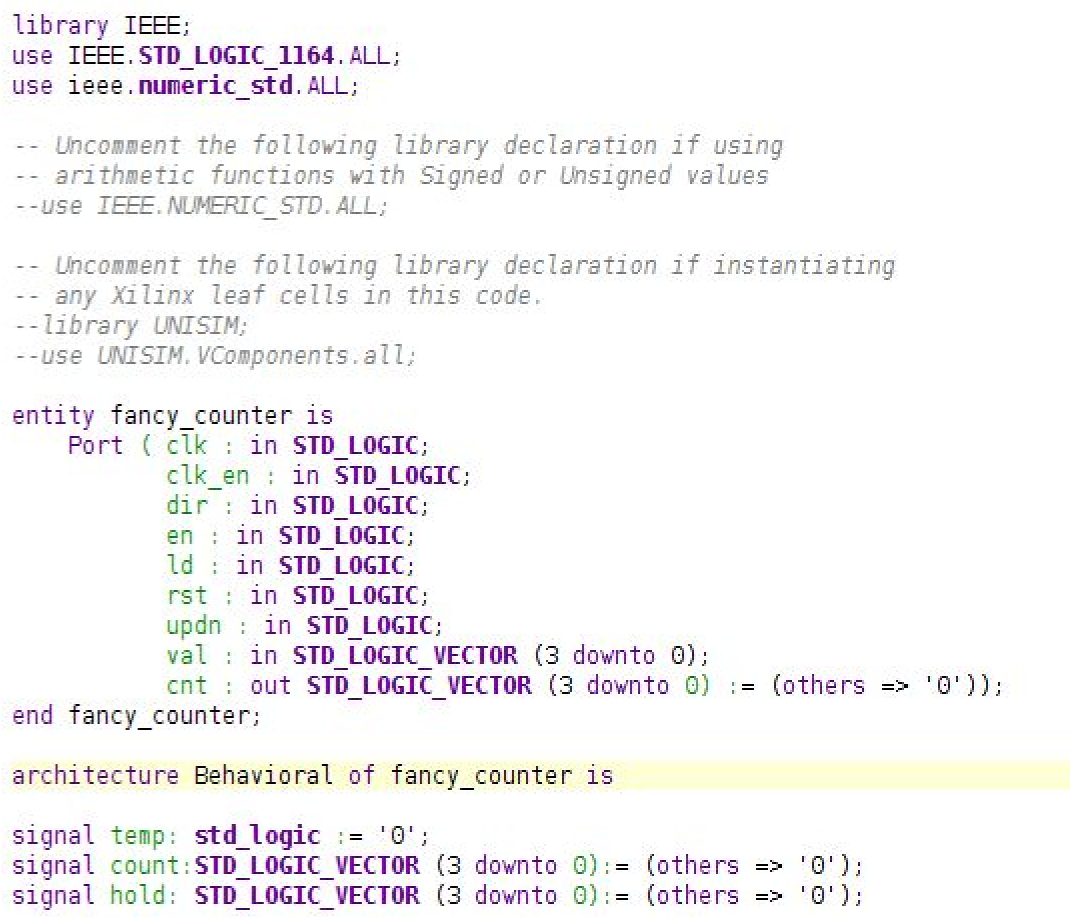
Observations: When running the simulations from the test bench, I noticed that in a few instances, the compiler considers a value at it’s rising edge to be ‘1’ as seen in the waveform above at 122 - 142ns in some cases, while ignoring the rising edge in another like in 64ns.

3. Actually Using a Counter to Count

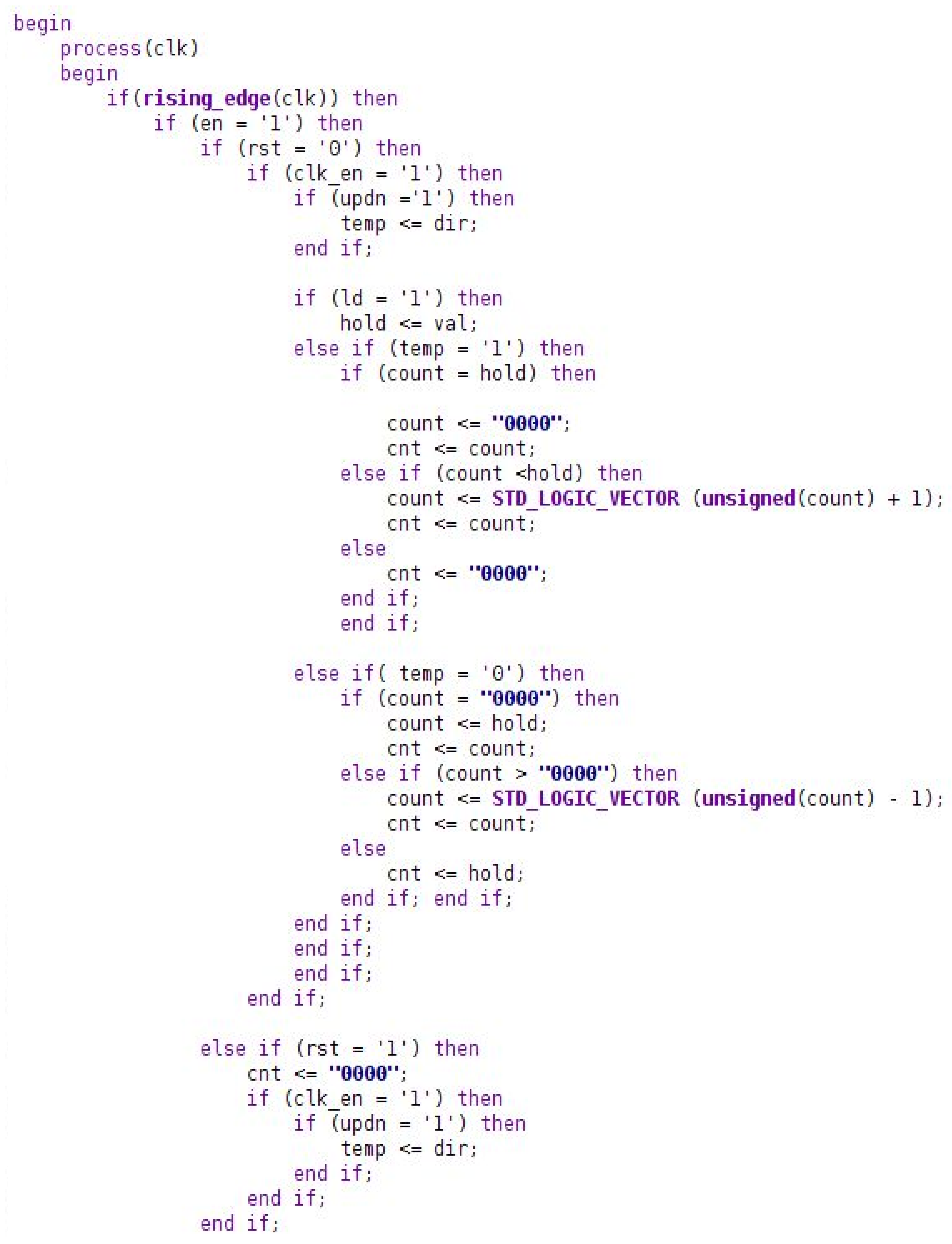
Theory:

This circuit takes in a clock and a value, and several other inputs that determine how the 4 bit output value acts per clock cycle. The inputs are clock, clock enabler, direction logical input, enabler, load, reset, updn (direction logical input enabler), and a 4 bit value. The 4 bit value acts as an upper and lower limit for the counter, of which depends on whether the counter is incrementing or decrementing per each clock cycle. The updn input prevents the counter from accidentally changing direction if the direction input changes, if updn is enabled, then the counter will change to whatever dir is set to. If tasked to load by the load input, the output will be set to the current 4 bit value being inserted. And if reset is on, the counter output gets reset to 0 regardless of the input except for the main clock.

VHDL:



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.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;
           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) := (others => '0'));
end fancy_counter;

architecture Behavioral of fancy_counter is

signal temp: std_logic := '0';
signal count:STD_LOGIC_VECTOR (3 downto 0):= (others => '0');
signal hold: STD_LOGIC_VECTOR (3 downto 0):= (others => '0');
```



```vhdl
begin
    process(clk)
    begin
        if(rising_edge(clk)) then
            if (en = '1') then
                if (rst = '0') then
                    if (clk_en = '1') then
                        if (updn ='1') then
                            temp <= dir;
                        end if;

                        if (ld = '1') then
                            hold <= val;
                        else if (temp = '1') then
                            if (count = hold) then

                                count <= "0000";
                                cnt <= count;
                            else if (count <hold) then
                                count <= STD_LOGIC_VECTOR (unsigned(count) + 1);
                                cnt <= count;
                            else
                                cnt <= "0000";
                            end if;
                            end if;

                        else if( temp = '0') then
                            if (count = "0000") then
                                count <= hold;
                                cnt <= count;
                            else if (count > "0000") then
                                count <= STD_LOGIC_VECTOR (unsigned(count) - 1);
                                cnt <= count;
                            else
                                cnt <= hold;
                            end if; end if;
                        end if;
                        end if;
                        end if;
                    end if;

                else if (rst = '1') then
                    cnt <= "0000";
                    if (clk_en = '1') then
                        if (updn = '1') then
                            temp <= dir;
                        end if;
                    end if;
                end if;
```

*rest of code is more ‘end’ statements

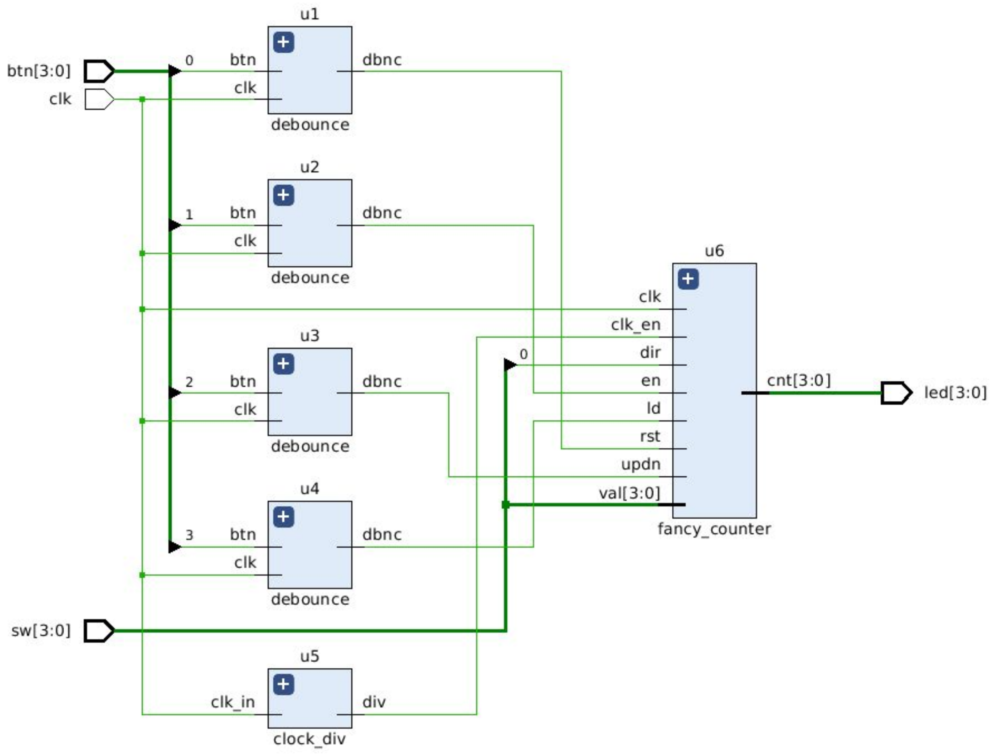
Observations: When writing this, I had assumed that ‘ld’ loads a new value to the counter, instead of just loading the current inserted value to the value register. Also, I had attempted to use ‘elsifs’ instead of ‘else ifs,’ but VHDL gave me an error every time. Regardless, I ended up learning how to manage codes that involve a lot of conditions.

4. Bringing it all together

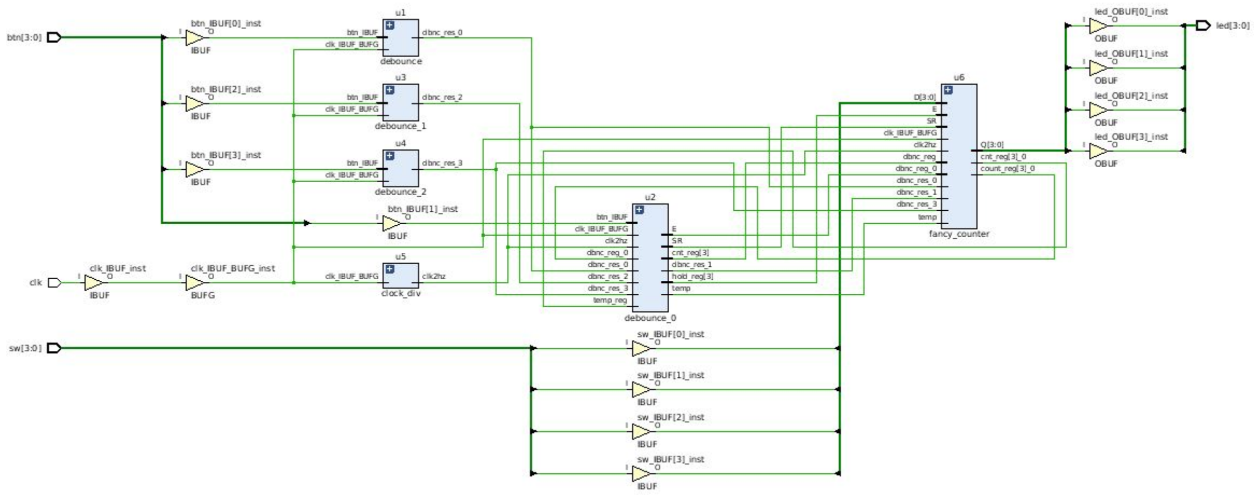
Theory:

This circuit is designed to implement every previous circuit together to test the fancy counter, every button input must be processed by the debouncer before entering the fancy counter, and the clock enabler is ran through a clock divider so the counter can display the output in a desired frequency. It is because of this design that the clock divider outputs in pulses instead of squares, VHDL does not allow two signals of the same architecture to read on rising edge,if clock_div outputs in squares, clk_en will read “1”, and the counter will change at 125 MHz instead of 2 Hz (as desired by the clock_div.) Switches do not need to run through the debouncer due to the static and polar nature of a switch over a button. Besides this, the purpose of this design is to test the fancy counter.

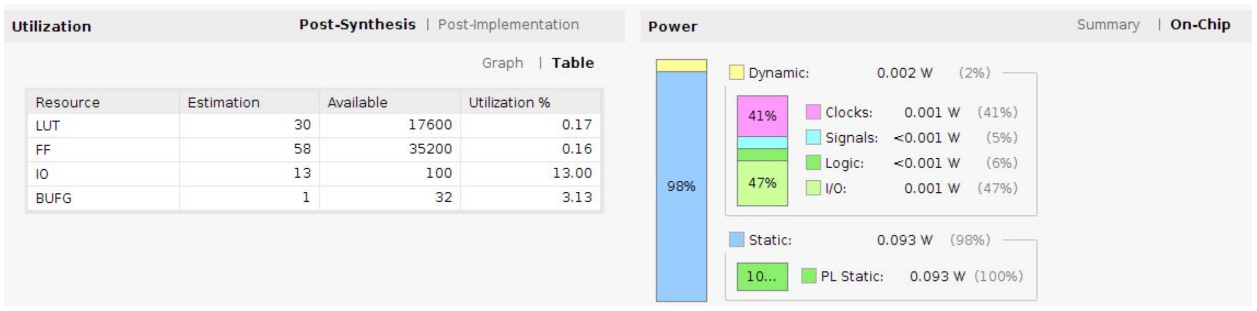
Elaborated Design:



Synthesis Schematic:



Project Summary:



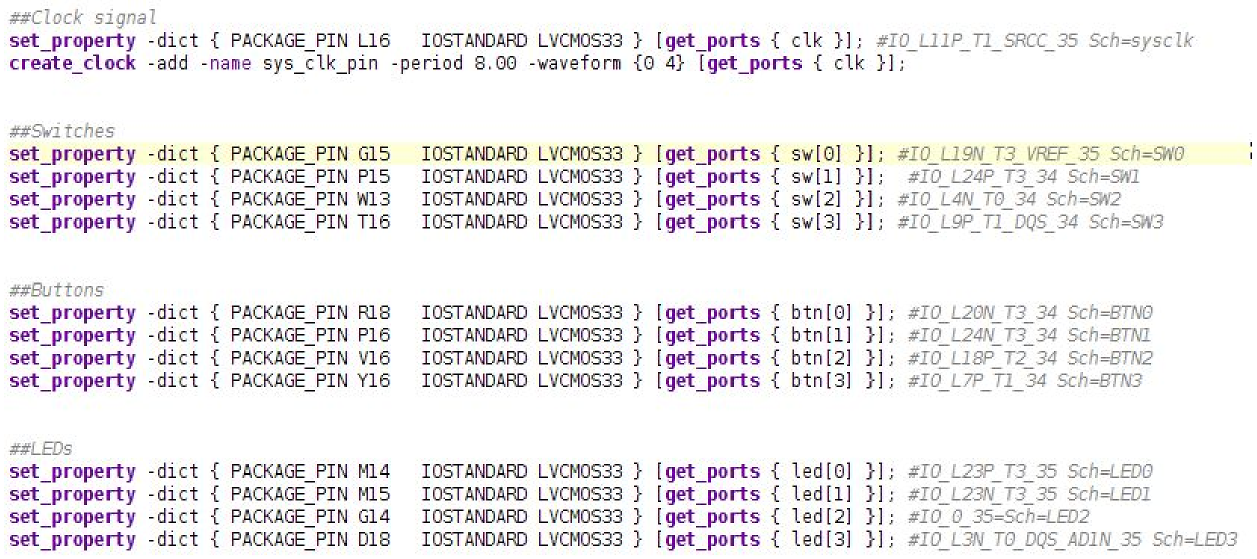
Utilization — **Post-Synthesis** | Post-Implementation — Graph | **Table**

Resource	Estimation	Available	Utilization %
LUT	30	17600	0.17
FF	58	35200	0.16
IO	13	100	13.00
BUFG	1	32	3.13

Power — Summary | **On-Chip**

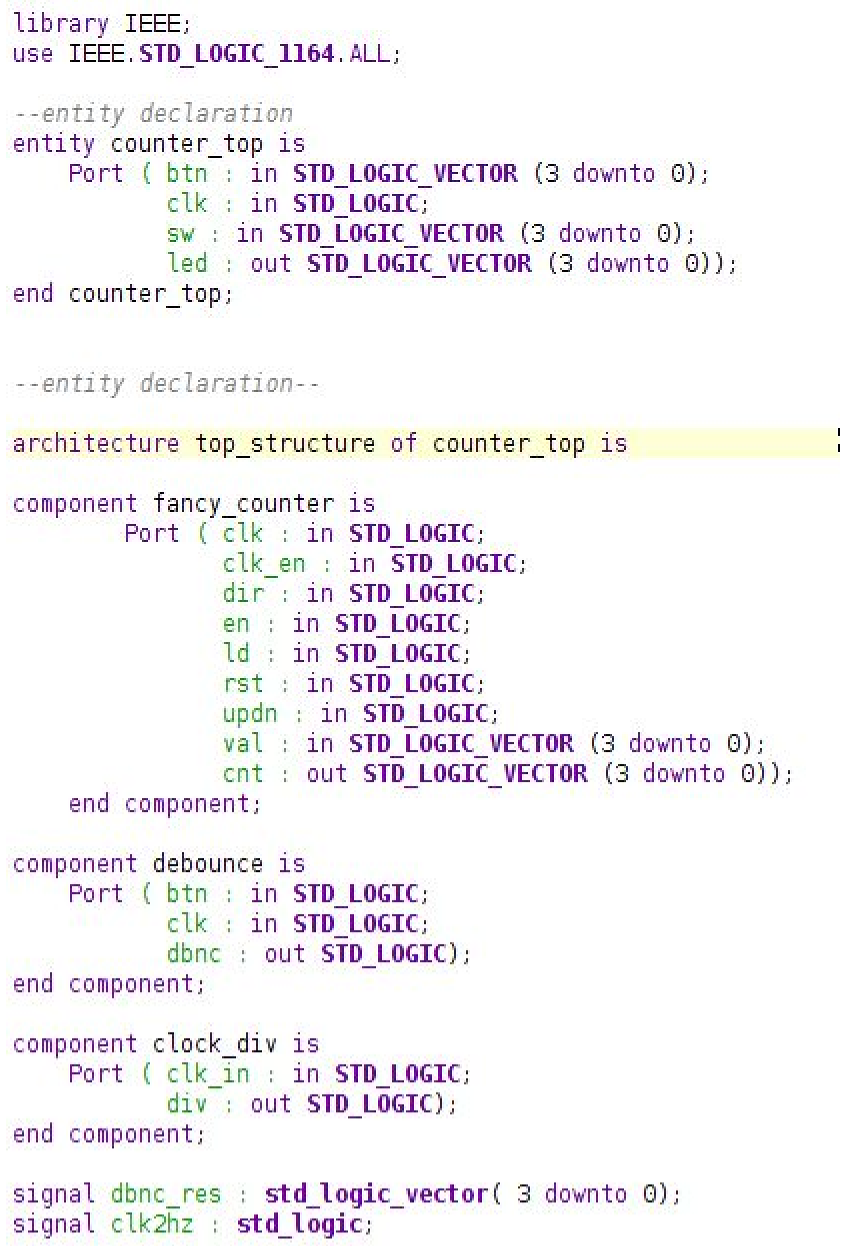
🟨	Dynamic:	0.002 W	(2%)
🟪	Clocks:	0.001 W	(41%)
🟦	Signals:	<0.001 W	(5%)
🟩	Logic:	<0.001 W	(6%)
🟩	I/O:	0.001 W	(47%)
🟦	Static:	0.093 W	(98%)
🟩	PL Static:	0.093 W	(100%)

XDC:

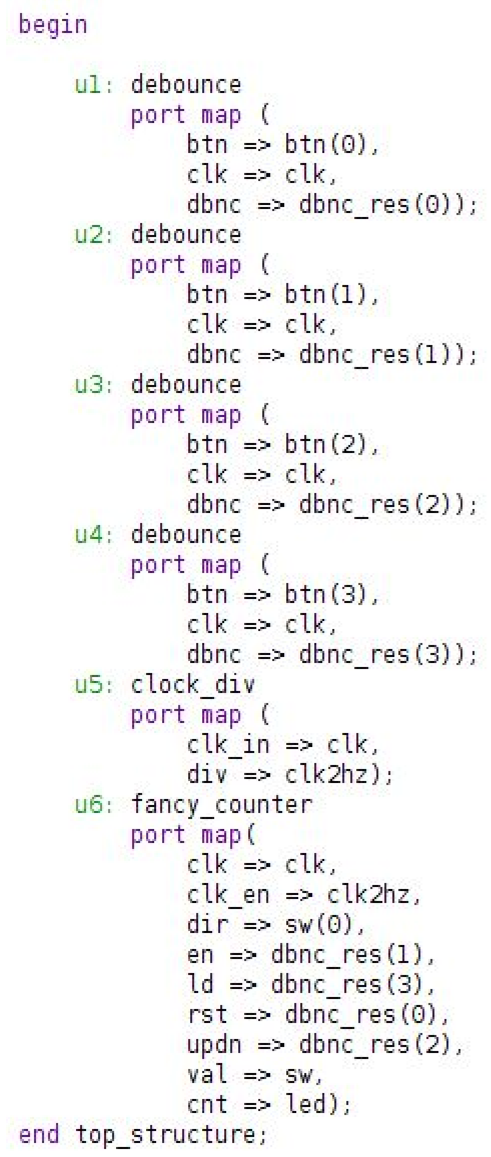


```
##Clock signal
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L11P_T1_SRCC_35 Sch=sysclk
create_clock -add -name sys_clk_pin -period 8.00 -waveform {0 4} [get_ports { clk }];


##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
```

VHDL:



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

--entity declaration
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 : out STD_LOGIC_VECTOR (3 downto 0));
end counter_top;


--entity declaration--

architecture top_structure of counter_top is

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;

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

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

signal dbnc_res : std_logic_vector( 3 downto 0);
signal clk2hz : std_logic;
```



```
begin

    u1: debounce
        port map (
            btn => btn(0),
            clk => clk,
            dbnc => dbnc_res(0));
    u2: debounce
        port map (
            btn => btn(1),
            clk => clk,
            dbnc => dbnc_res(1));
    u3: debounce
        port map (
            btn => btn(2),
            clk => clk,
            dbnc => dbnc_res(2));
    u4: debounce
        port map (
            btn => btn(3),
            clk => clk,
            dbnc => dbnc_res(3));
    u5: clock_div
        port map (
            clk_in => clk,
            div => clk2hz);
    u6: fancy_counter
        port map(
            clk => clk,
            clk_en => clk2hz,
            dir => sw(0),
            en => dbnc_res(1),
            ld => dbnc_res(3),
            rst => dbnc_res(0),
            updn => dbnc_res(2),
            val => sw,
            cnt => led);
end top_structure;
```

Observations:

One lesson I learned when writing the port maps was to utilize signals to act as conduits between outputs and inputs due to VHDL not being able to read outputs instead of changing the outputs to ‘inout,’ which doesn’t implement well with the design.