Kevin Honeker

Embedded Systems

Lab Report 1

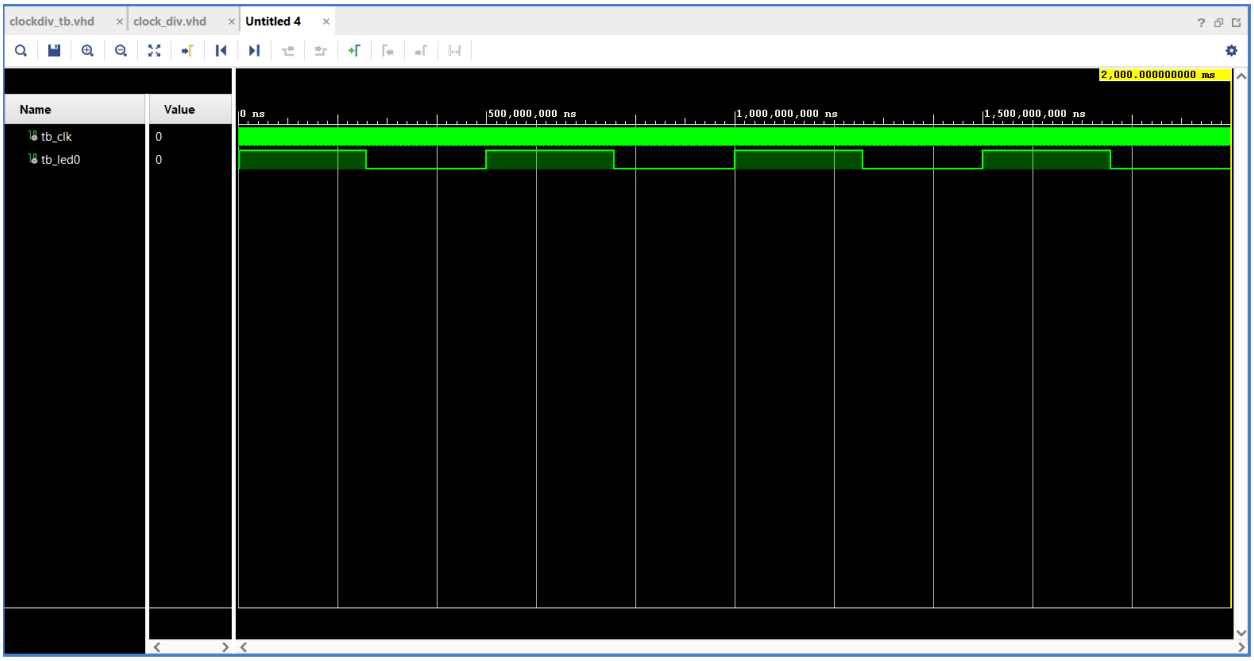
2/28

Purpose: The purpose of this lab was to create a sophisticated counter operated by buttons whose signals were passed through debouncers.

Part 1: clock divider

Theory of operation: the clock divider takes a 125MHz input and converts it to a 2Hz clock.

Below are the wave forms for the clock divider. The simulation time was two seconds and the out has a frequency of 2Hz

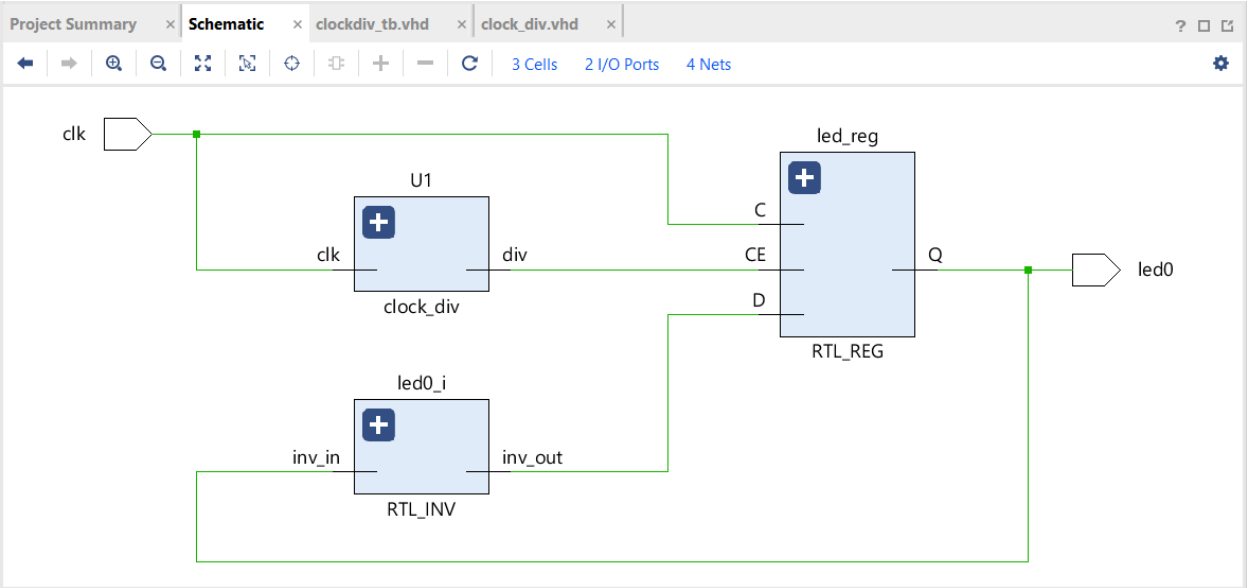



```
library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;

entity clock_div is
    port(
```

```
        clk  : in std_logic;
        div : out std_logic);
end clock_div;

architecture behavior 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) < 62499999) then
                counter <= std_logic_vector(unsigned(counter) + 1);

            else
                counter <= (others => '0');

            end if;


            if (unsigned(counter) = 31250000) then
                div <= '1';

            else
```

```
            div <= '0';

          end if;

      end if;
  end process;

end behavior;
```

Divider_top schematic:




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

entity RTL_REG is
  port (C, CE, D : in std_logic;
        Q     : out std_logic);
end RTL_REG;
```

```
architecture dff of RTL_REG is
begin
process(C)
begin
  if C'event and C='1' then
     if CE ='1' then
       Q <= D;
     end if;
  end if;
end process;
end dff;

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


entity RTL_INV is
   port(inv_in  : in std_logic;
       inv_out : out std_logic);
end RTL_INV;

architecture inverter of RTL_INV is
begin
   inv_out <= not inv_in;
end inverter;

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);
end divider_top;



architecture divider_arch of divider_top is
 component clock_div
    port(clk : in std_logic;
         div : out std_logic);
  end component;

 component RTL_INV
    port (inv_in  : in std_logic;
          inv_out : out std_logic);
 end component;
 component RTL_REG
    port (C, CE, D : in std_logic;
          Q      : out std_logic);
end component;

signal inv_res, led0_out, div_sig : std_logic;

begin
    led_reg : RTL_REG
        port map ( C  => clk,
```

```
            CE => div_sig,
            D  => inv_res,
            Q  => led0_out);
   led0_i : RTL_INV
      port map(inv_in  => led0_out,
               inv_out => inv_res);


   U1 : clock_div
      port map(clk => clk,
               div => div_sig);

   led0 <= led0_out;
end divider_arch;
```

TEST BENCH FILE

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

entity clkdiv_tb is
end clkdiv_tb;

architecture testbench of clkdiv_tb is

   signal tb_clk : std_logic := '0';

   signal tb_led0 : std_logic;
```

```
    component clock_div is
        port(

            clk  : in std_logic;
            div : out std_logic

        );
    end component;

begin


    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;
```

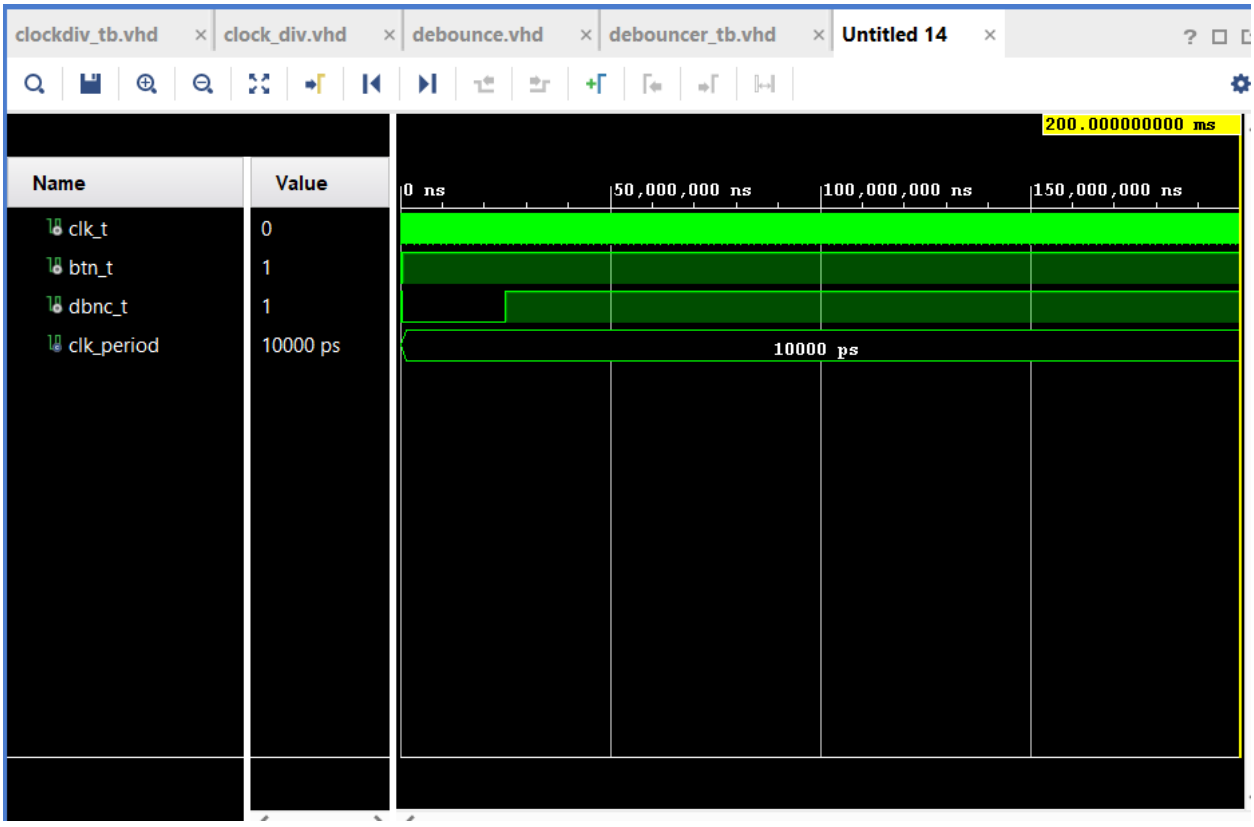
Q 1.1) The input needed to be divided by 62500000 (half of 125MHz) in order to get a 2Hz output.

Q 1.2) We would need at least 26 bits for this counter.

Part 2: Debouncer

Theory of Operation: The purpose of the debouncer is to stabilize the signal caused by the press of a button.

Below are the waveforms for the debouncer



Q2.1)The button has an output hi when it is pressed

Q2.3)We need 2.5 million clock ticks for a 20ms debounce time

Q2.4We need at least 22 bits for this counter

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

entity debouncer is
   port ( btn, clk : in std_logic;
          dbnc     : out std_logic);
end debouncer;
architecture debounce of debouncer is
signal count : std_logic_vector(24 downto 0);
signal shift_reg : std_logic_vector(1 downto 0);
begin

process (clk)

begin
  if clk='1' and clk'event then
   shift_reg(0) <= btn;
   shift_reg(1) <= shift_reg(0);
   if shift_reg(1) = '1' then
      count <= std_logic_vector(unsigned(count) +1);
      if unsigned(count) < 2500000 then
         dbnc <= '0';
      else
         dbnc <= '1';
      end if;
   else
      count<=(others =>'0');
      dbnc <='0';
    -- if btn= not '1' then
      --  count := 0;         --old code
    -- elsif btn='1' then
      --  count := count + 1;
     end if;
 -- end if;


   end if;
   --end if;
end process;
end debounce;
```

TEST BENCH

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

entity debouncer_tb is
end debouncer_tb;


architecture tst of debouncer_tb is
Component debouncer
port(clk :in std_logic;
   btn :in std_logic;
   dbnc :out std_logic);
end Component;

signal clk_t : std_logic := '0';
signal btn_t : std_logic;
signal dbnc_t : std_logic;

constant clk_period : time := 10 ns;

begin
uut: debouncer
 port map (clk => clk_t,
         btn => btn_t,
         dbnc => dbnc_t);

   clk_p :process
   begin
   clk_t <= '0';
      wait for clk_period/2;
      clk_t <= '1';
      wait for clk_period/2;


end process;


stim_p: process
begin
   wait for 100ns;
   btn_t<='1';
   --wait for 200ns;
   --btn_t<='0';
```

```
   --wait;

   end process;
end;
```

Part 3: Fancy Counter

Theory of Operation: For this part we are creating an up/down counter that we can input the maximum value it counts to/resets to when it hits zero.

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


entity fancy_counter is
   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 fancy_counter;


architecture fancy_behavior of fancy_counter is
signal direction : std_logic;
signal value    : std_logic_vector(3 downto 0);
signal count    : std_logic_vector(3 downto 0);
begin
   process (clk)


      begin


          if clk='1'and clk'event then
             if en = '1' then
```

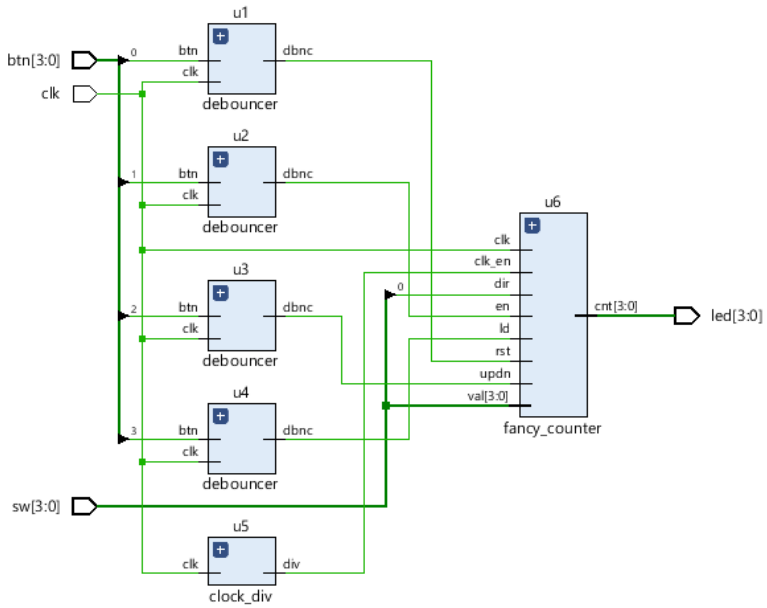
```
            if rst = '1' then
              cnt <= (others => '0');
              value <= "0000";
              count <= "0000";
            elsif clk_en = '1' then
            if updn ='1' then
               direction <= dir;
            end if;
            if ld = '1' then
               value <= val;
            end if;

            if direction = '1' then
              count <= std_logic_vector(unsigned(count) + 1);
              if (count = value) then
                 count <= "0000";
              end if;

            elsif direction = '0' then
              count <= std_logic_vector(unsigned(count) - 1);
              if count = "0000" then
                 count <= value;
              end if;
            end if;
            cnt <= std_logic_vector(count);
        end if;
     end if;
   end if;
end process;
```

```
end fancy_behavior;
```

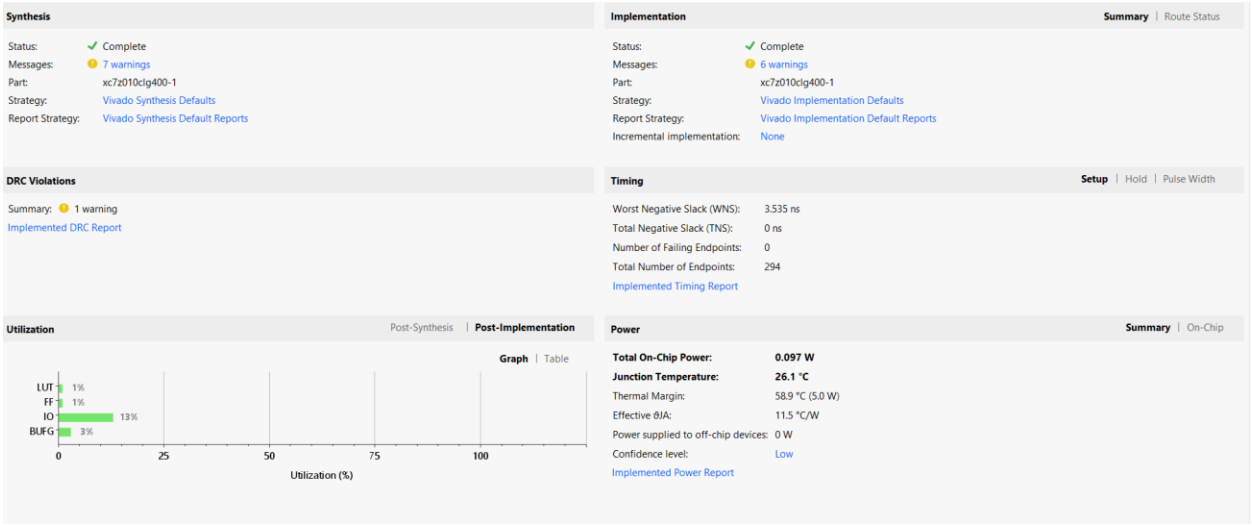
Part 4:Bringing it all together

Theory of operation: The counter will have inputs that are operated by buttons whose signals pass through debouncers. The input clock is 2Hz.

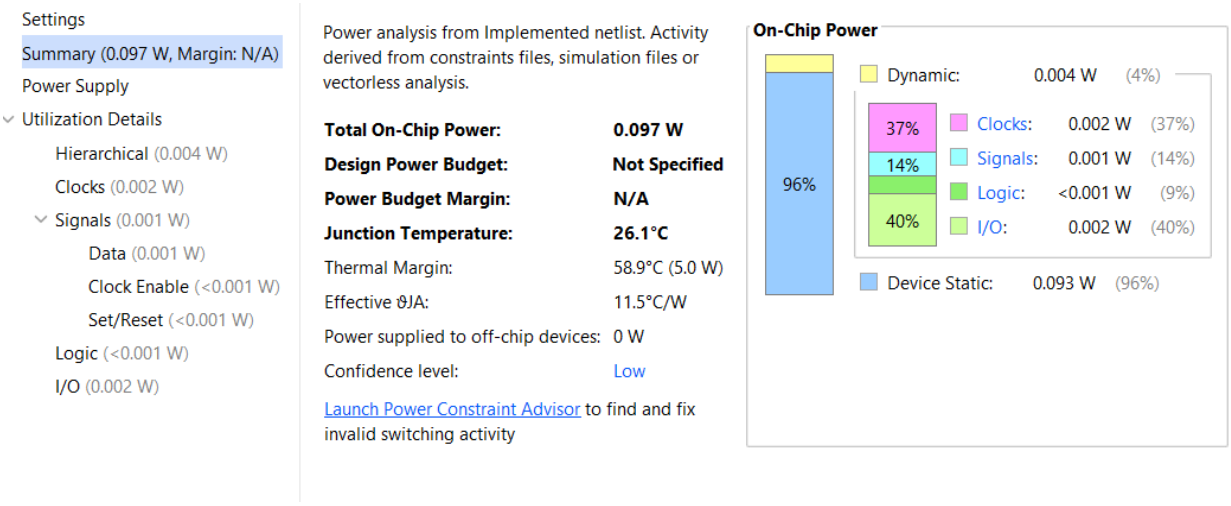
Block Diagram



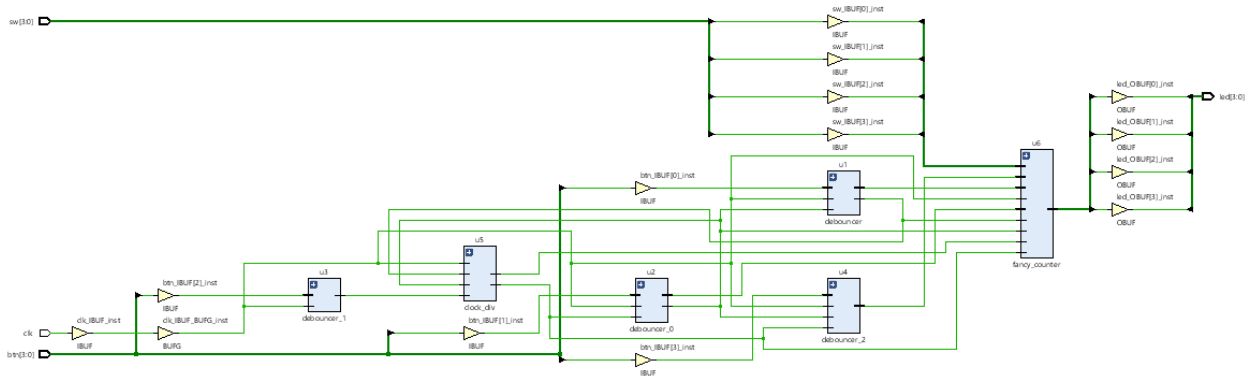
Project summary



Power analysis



Synthesized 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_arch of counter_top is
    signal dbnc1, dbnc2, dbnc3, dbnc4, div_sig : std_logic;
    component debouncer
     port ( btn, clk : in std_logic;
           dbnc     : out std_logic);
    end component;

    component fancy_counter
        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;

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

begin
    u1 : debouncer
      port map (btn => btn(0),
            clk => clk,
            dbnc => dbnc1);

    u2 : debouncer
      port map (btn => btn(1),
            clk => clk,
            dbnc => dbnc2);

     u3 : debouncer
      port map (btn => btn(2),
            clk => clk,
            dbnc => dbnc3);

      u4 : debouncer
      port map (btn => btn(3),
            clk => clk,
```

```
            dbnc => dbnc4);


    u5 : clock_div
     port map( clk=> clk,
            div=>div_sig);


    u6: fancy_counter
     port map (clk    => clk,
            clk_en => div_sig,
            dir    => sw(0),
            en     => dbnc2,
            ld     => dbnc4,
            rst    => dbnc1,
            updn   => dbnc3,
            val    => sw,
            cnt    => led);


end counter_arch;
```

Observations:

In this lab I learned a lot about using the Vivado software and VHDL Logic. In my debouncer, I had to figure out a way to get the counter to stop counting after it reached the desired limit, because the counter would overflow and reset back to zero. This caused a brief flicker in the LEDs which in turn distorted the functionality of my final circuit.