David Elliott

Evan Yeh

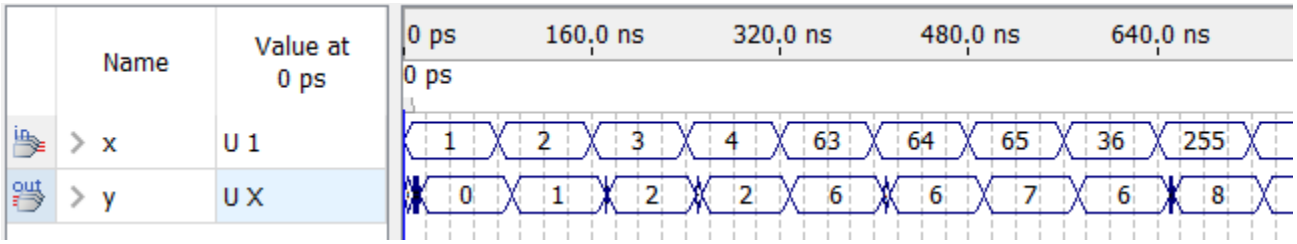
EE125 Homework #4

Exercise 1:

a.) Code:

```
01  ----------------------------------------------------------------------------
02  library ieee;
03  use ieee.std_logic_1164.all;
04  use ieee.numeric_std.all;
05  use ieee.math_real.all;
06  ----------------------------------------------------------------------------
07  entity log2_prob is
08      generic (BITS: natural := 8);
09      port (x: in std_logic_vector(BITS - 1  downto 0);
10              -- generically:
11              -- y: out std_logic_vector(ceil(log2(BITS)) downto 0);
                --    not synthesizeable
12              -- for simulation:
13              y: out std_logic_vector(3 downto 0));
14  end entity;
15  ----------------------------------------------------------------------------
16  architecture problem_one of log2_prob is
17      function ceil_log2 (input: std_logic_vector(BITS - 1  downto 0))
18          return integer is
19          variable i: integer := 0;
20          variable inp_int: integer:= 0;
21      begin
22          inp_int := to_integer(unsigned(x));
23          while (2 ** i < inp_int and i <= BITS) loop
24              i := i + 1;
25          end loop;
26          return i;
27      end function ceil_log2;
28
29      signal x_int: integer;
30  begin
31      -- generically:
32      -- y <= std_logic_vector(to_unsigned(ceil_log2(x), ceil_log2(BITS));
33      -- for simulation:
34      y <= std_logic_vector(to_unsigned(ceil_log2(x), 4));
35  end architecture;
36  ----------------------------------------------------------------------------
```

b.) Simulation:



Exercise 2:

a.) Code:

natural_to_thermometer.vhd

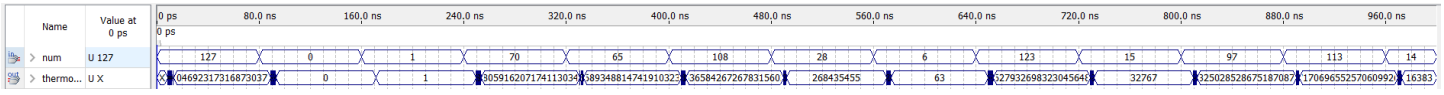
```
01  --------------------------------------------------------------------------------
02  library ieee;
03  use ieee.std_logic_1164.all;
04  use ieee.numeric_std.all;
05  --------------------------------------------------------------------------------
06  package natural_to_thermometer is
07      constant IN_BITS: integer := 7;
08      function natural_to_thermometer(signal num: natural) return std_logic_vector;
09  end natural_to_thermometer;
10  --------------------------------------------------------------------------------
11  package body natural_to_thermometer is
12      function natural_to_thermometer(signal num: natural) return std_logic_vector is
13          constant OUT_BITS: integer := 2**IN_BITS;
14          variable thermometer: std_logic_vector(OUT_BITS - 1 downto 0);
15      begin
16          for i in 0 to OUT_BITS - 1 loop
17              if i < num then
18                  thermometer(i) := '1';
19              else
20                  thermometer(i) := '0';
21              end if;
22          end loop;
23          return thermometer;
24      end function;
25  end package body;
26  --------------------------------------------------------------------------------
```

natural_to_thermometer_test.vhd

```
01  --------------------------------------------------------------------------------
02  library ieee;
03  use ieee.std_logic_1164.all;
04  use ieee.numeric_std.all;
05  library work;
06  use work.natural_to_thermometer.all;
07  --------------------------------------------------------------------------------
08  entity natural_to_thermometer_test is
09      generic(IN_BITS: integer := 7);
10      PORT (num: in std_logic_vector(IN_BITS-1 downto 0);
11              thermometer: out std_logic_vector(2**IN_BITS - 1 downto 0));
12  end entity;
13  --------------------------------------------------------------------------------
14  architecture test of natural_to_thermometer_test is
15  begin
16      thermometer <= natural_to_thermometer(to_integer(unsigned(num)));
17  end architecture;
18  --------------------------------------------------------------------------------
```

b.) Simulation:

Binary values are too large to display in the waveform, displaying with unsigned decimals instead:

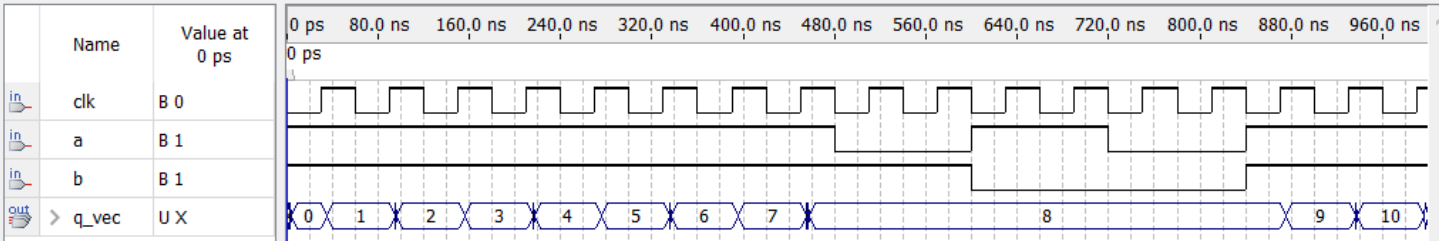


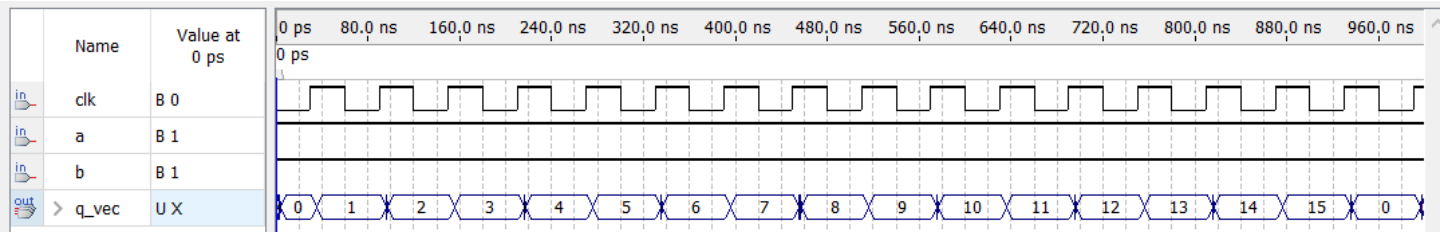
Exercise 3:

a.) Code:

```
01  ---- tflipflop:-------------------------------------------------------------
02  library ieee;
03  use ieee.std_logic_1164.all;
04  --------------------------------------------------------------------------------
05  entity tflipflop is
06      port (a, b, clk: in std_logic;
07              x: out std_logic;
08              q: buffer std_logic);
09  end entity;
10  --------------------------------------------------------------------------------
11  architecture tflipflop of tflipflop is
12  begin
13      process(clk)
14      begin
15          x <= a and b;
16          if rising_edge(clk) then
17              q <= (a and b) xor q;
18          end if;
19      end process;
20  end architecture;
21  ----Main code:-------------------------------------------------------------
22  library ieee;
23  use ieee.std_logic_1164.all;
24  entity sync_counter is
25      generic (BITS: natural := 4);
26      port (a, b, clk: in std_logic;
27              q_vec: buffer std_logic_vector(BITS - 1 downto 0));
28  end entity;
29  --------------------------------------------------------------------------------
30  architecture sync_counter of sync_counter is
31      signal x_vec: std_logic_vector(BITS - 1 downto 0);
32
33      component tflipflop is
34          port (a, b, clk: in std_logic;
35                  x: out std_logic;
36                  q: buffer std_logic);
37      end component;
38
39  begin
40      cell: tflipflop PORT MAP (a, b, clk, x_vec(0), q_vec(0));
41      gen: for i in 1 to BITS - 1 generate
42          cells: tflipflop PORT MAP (x_vec(i-1), q_vec(i-1), clk, x_vec(i),
                    q_vec(i));
43      end generate gen;
44
45  end architecture;
```

b.) Simulation:





Exercise 4

a.) Code:

pwm_demo.vhd

```
01  ----------------------------------------------------------------------------
02  library ieee;
03  use ieee.std_logic_1164.all;
04  ----------------------------------------------------------------------------
05  entity pwm_demo is
06      generic (F_CLK: integer:=50;
07          T_CLK_PWM: integer:=120;
08          BITS_DUTY: integer:=3);
09      PORT (clk: in std_logic;
10              duty: in std_logic_vector(BITS_DUTY-1 downto 0);
11              pwm_out: out std_logic;
12              count: out std_logic_vector(BITS_DUTY-1 downto 0));
13  END ENTITY;
14  ----------------------------------------------------------------------------
15  architecture pwm_demo of pwm_demo is
16      signal pwm_clk: std_logic;
17  begin
18      pwm: entity work.pwm(pwm) generic map(BITS_DUTY)
19          port map(duty, pwm_clk, pwm_out, count);
20      process(clk)
21
22      constant CYCLES_MAX: integer := (T_CLK_PWM * F_CLK) / 2000 - 1;
23      variable cycles: integer range 0 to CYCLES_MAX := CYCLES_MAX;
24      begin
25          if rising_edge(clk) then
26              if cycles = CYCLES_MAX then
27                  cycles := 0;
28              else
29                  cycles := cycles + 1;
30              end if;
31              if cycles = 0 then
32                  pwm_clk <= NOT(pwm_clk);
33              end if;
34          end if;
35      end process;
36  end architecture;
37  ----------------------------------------------------------------------------
```

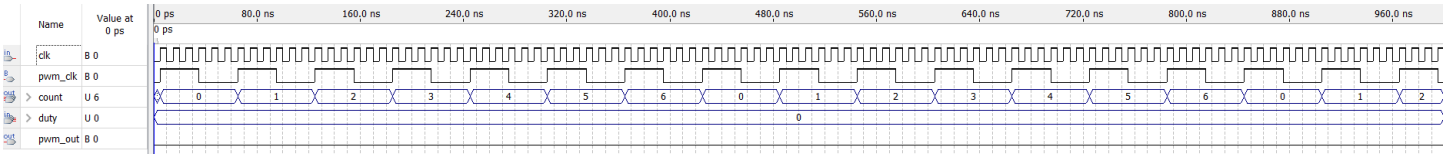
pwm.vhd

```
01  ----------------------------------------------------------------------------
02  library ieee;
03  use ieee.std_logic_1164.all;
04  use ieee.numeric_std.all;
05  ----------------------------------------------------------------------------
06  entity pwm is
07      generic (BITS_DUTY: integer := 3);
08      PORT (duty: in std_logic_vector(BITS_DUTY-1 downto 0);
```

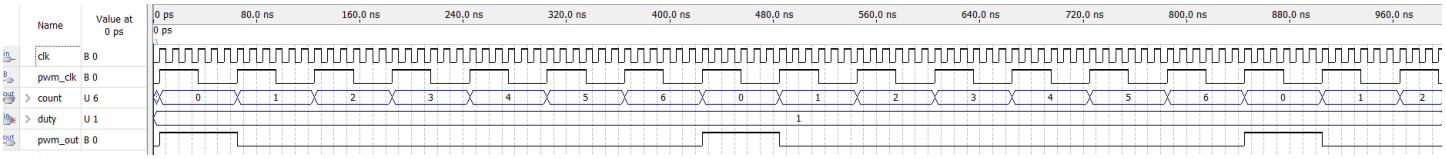
```
09                pwm_clk: in std_logic;
10                pwm: out std_logic;
11                count_vec: out std_logic_vector(BITS_DUTY-1 downto 0));
12    END ENTITY;
13    -------------------------------------------------------------------------
14    architecture pwm of pwm is
15        signal duty_int: integer range 0 to 2**BITS_DUTY - 1;
16    begin
17        duty_int <= to_integer(unsigned(duty));
18        process(pwm_clk)
19            constant COUNT_MAX: integer := 2**BITS_DUTY - 2;
20            variable count: integer range 0 to COUNT_MAX := COUNT_MAX;
21            begin
22                if rising_edge(pwm_clk) then
23                    if count = COUNT_MAX then
24                        count := 0;
25                    else
26                        count := count + 1;
27                    end if;
28                    if count = 0 then
29                        pwm <= '1';
30                    end if;
31                    if count = duty_int then
32                        pwm <= '0';
33                    end if;
34                end if;
35                count_vec <= std_logic_vector(to_unsigned(count, BITS_DUTY));
36            end process;
37    end architecture;
38    -------------------------------------------------------------------------
```

b.)

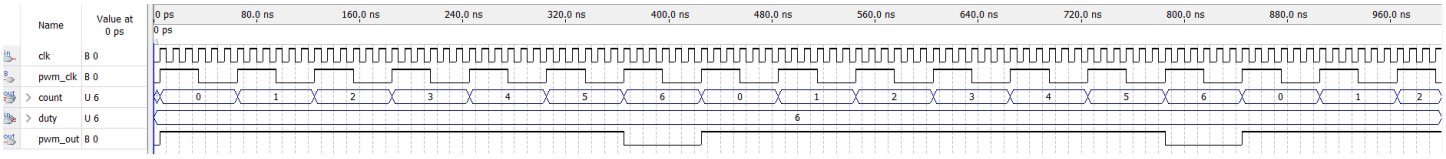
Duty = 0



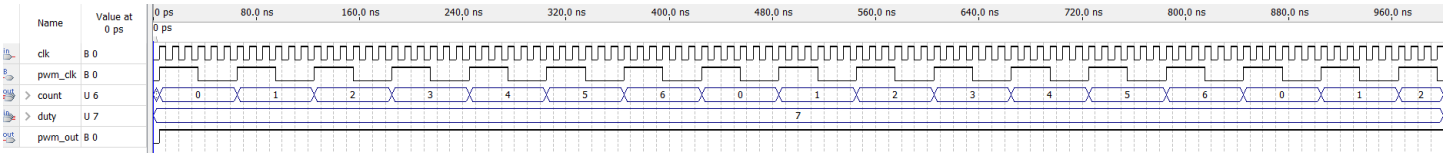
Duty = 1



Duty = 6



Duty = 7



c.) Simulations are similar to the figure and the pwm pulse does start at zero.

d.) To demonstrate the PWM functionality, I used pwm_demo to divide the 50hz CLK frequency down to CLK_PWM. I specified BITS_DUTY=3, and used three switches on the Terasic as the three duty input bits. I also set the pwm output to RLED0, and observed that the LED was dimmed more when the duty cycle was less, turned off completely when the duty cycle was zero, and turned on completely when the duty cycle was 7.