VHDL LAB #5

Rachel Ruddy - 261162987
Natasha Lawford - 261178596
Rachel Bunsick - 261159677

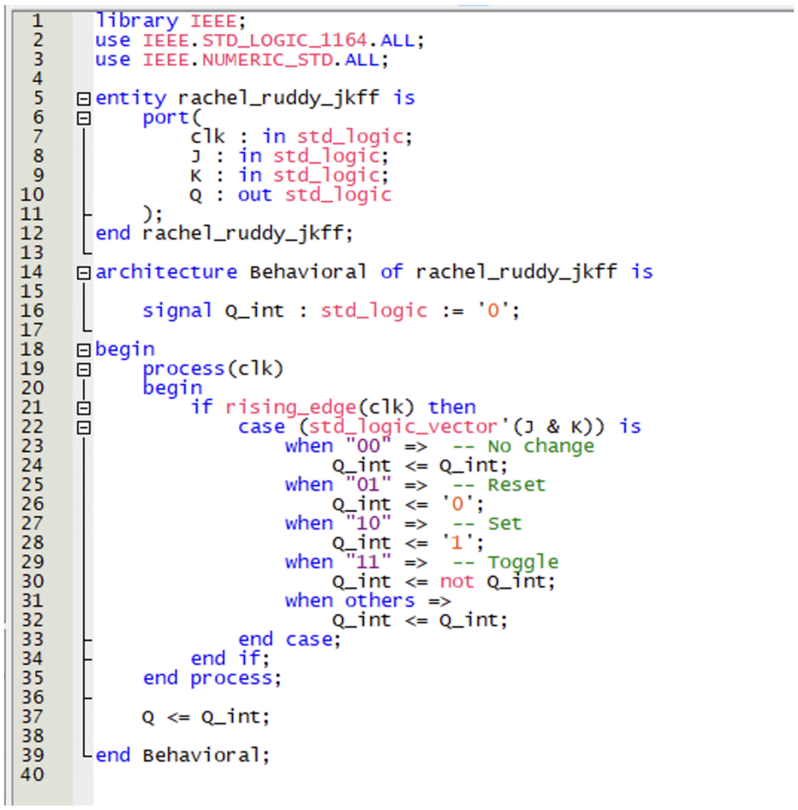
I. Executive Summary:

In this lab, we modeled and implemented sequential circuits on the Altera DE1-S FPGA board. We first modeled the JK-flip flop by writing a behavioral VHDL description along with a testbench to test its functionality in each case. Then, we created a 3-bit up counter which counts the numbers from 0 to 7 and then resets back to 0. The 3-bit up counter was also implemented using a behavioral description and tested with a testbench file. We also developed a clock divider behavioral description which asserts 1 after 1 second (50,000,000 clock cycles). The clock divider was also tested with a testbench file which tests its functionality based on different inputs. Finally, we created a wrapper VHDL file to create a 3-bit up counter which increments up each second. The counter was tested on the Altera DE1-S FPGA board using the 7-segment decoder code to display the counter on the board.

II. Questions:

1. Briefly explain your VHDL code implementation of all circuits

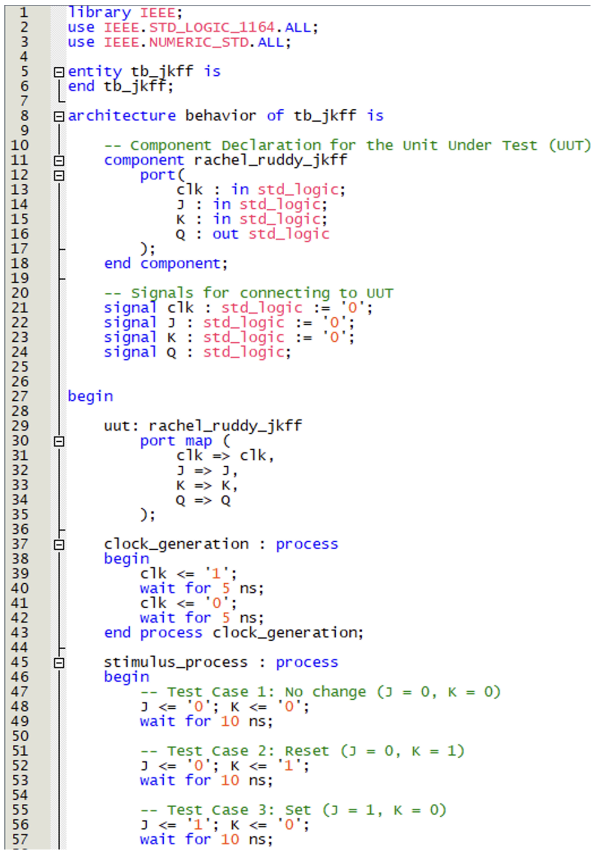
JKFF:



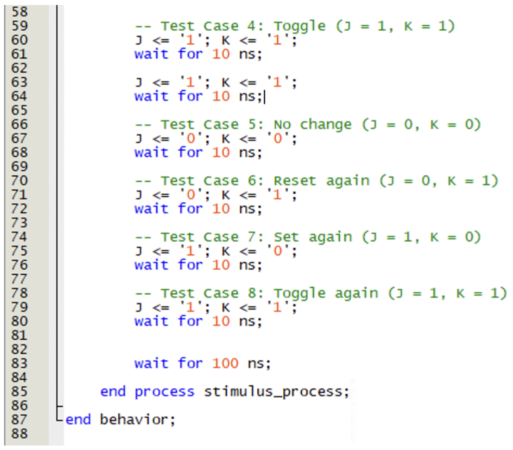
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity rachel_ruddy_jkff is
    port(
        clk : in std_logic;
        J : in std_logic;
        K : in std_logic;
        Q : out std_logic
    );
end rachel_ruddy_jkff;

architecture Behavioral of rachel_ruddy_jkff is

    signal Q_int : std_logic := '0';

begin
    process(clk)
    begin
        if rising_edge(clk) then
            case (std_logic_vector'(J & K)) is
                when "00" =>  -- No change
                    Q_int <= Q_int;
                when "01" =>  -- Reset
                    Q_int <= '0';
                when "10" =>  -- Set
                    Q_int <= '1';
                when "11" =>  -- Toggle
                    Q_int <= not Q_int;
                when others =>
                    Q_int <= Q_int;
            end case;
        end if;
    end process;

    Q <= Q_int;

end Behavioral;

```

The JKFF code uses a sequential process which is run each time that the value of the clock signal changes. If the clock is at a positive (or rising) edge, then we check each case of the input values of J and K. 1If both J and K are equal to 0, then the flip flop retains the previous state. If J = 0 and K = 1, then the reset functionality is activated and the current state is reset to 0. If J = 1 and K = 0, then the set functionality is activated and the current state is updated to 1. If J = K = 1, then the toggle functionality is activated and the current state is set to be the opposite of the previous state. This reflects the functionality of a JKFF as expected.

JKFF Testbench:



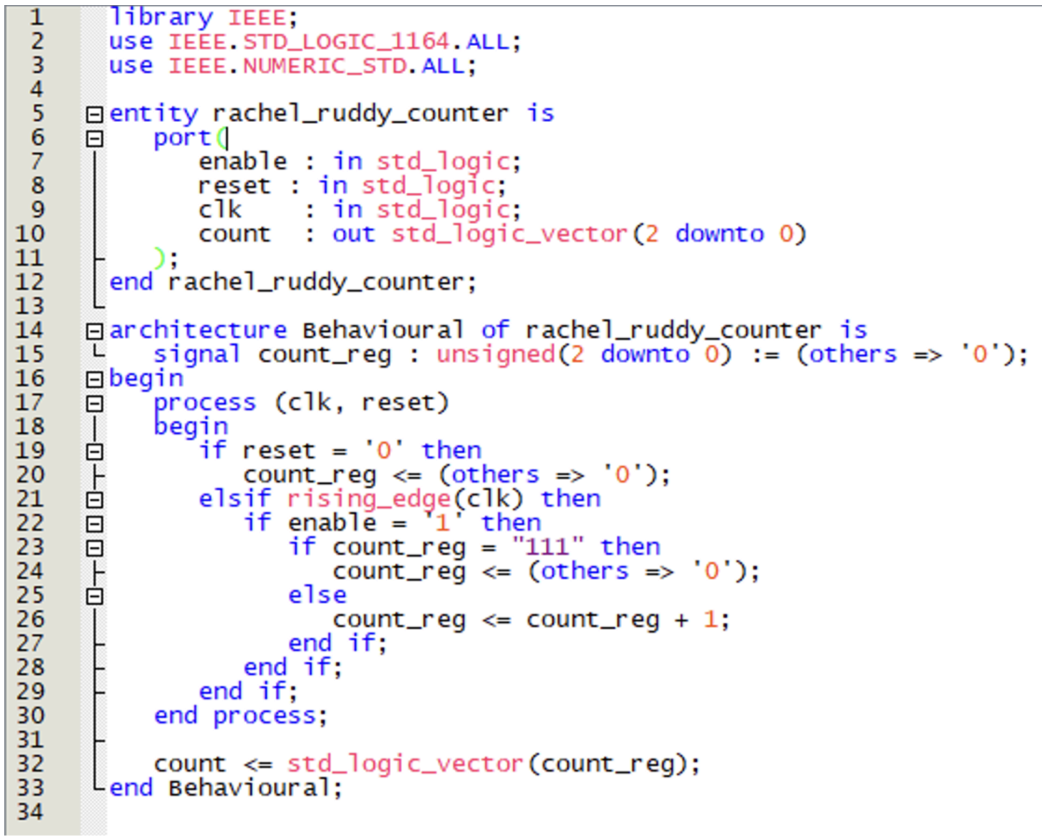
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity tb_jkff is
end tb_jkff;

architecture behavior of tb_jkff is

    -- Component Declaration for the Unit Under Test (UUT)
    component rachel_ruddy_jkff
        port(
            clk : in std_logic;
            J : in std_logic;
            K : in std_logic;
            Q : out std_logic
        );
    end component;

    -- Signals for connecting to UUT
    signal clk : std_logic := '0';
    signal J : std_logic := '0';
    signal K : std_logic := '0';
    signal Q : std_logic;


begin

    uut: rachel_ruddy_jkff
        port map (
            clk => clk,
            J => J,
            K => K,
            Q => Q
        );

    clock_generation : process
    begin
        clk <= '1';
        wait for 5 ns;
        clk <= '0';
        wait for 5 ns;
    end process clock_generation;

    stimulus_process : process
    begin
        -- Test Case 1: No change (J = 0, K = 0)
        J <= '0'; K <= '0';
        wait for 10 ns;

        -- Test Case 2: Reset (J = 0, K = 1)
        J <= '0'; K <= '1';
        wait for 10 ns;

        -- Test Case 3: Set (J = 1, K = 0)
        J <= '1'; K <= '0';
        wait for 10 ns;
```



```vhdl

        -- Test Case 4: Toggle (J = 1, K = 1)
        J <= '1'; K <= '1';
        wait for 10 ns;

        J <= '1'; K <= '1';
        wait for 10 ns;

        -- Test Case 5: No change (J = 0, K = 0)
        J <= '0'; K <= '0';
        wait for 10 ns;

        -- Test Case 6: Reset again (J = 0, K = 1)
        J <= '0'; K <= '1';
        wait for 10 ns;

        -- Test Case 7: Set again (J = 1, K = 0)
        J <= '1'; K <= '0';
        wait for 10 ns;

        -- Test Case 8: Toggle again (J = 1, K = 1)
        J <= '1'; K <= '1';
        wait for 10 ns;


        wait for 100 ns;

    end process stimulus_process;

end behavior;
```

The testbench for the JKFF first instantiates the JKFF. Then, we start two sequential processes, one which generates the clock signal and the other which tests exhaustively the 4 possible combinations of J and K. The testbench iterates through each case twice to ensure that the memory and update of the states functions as expected.

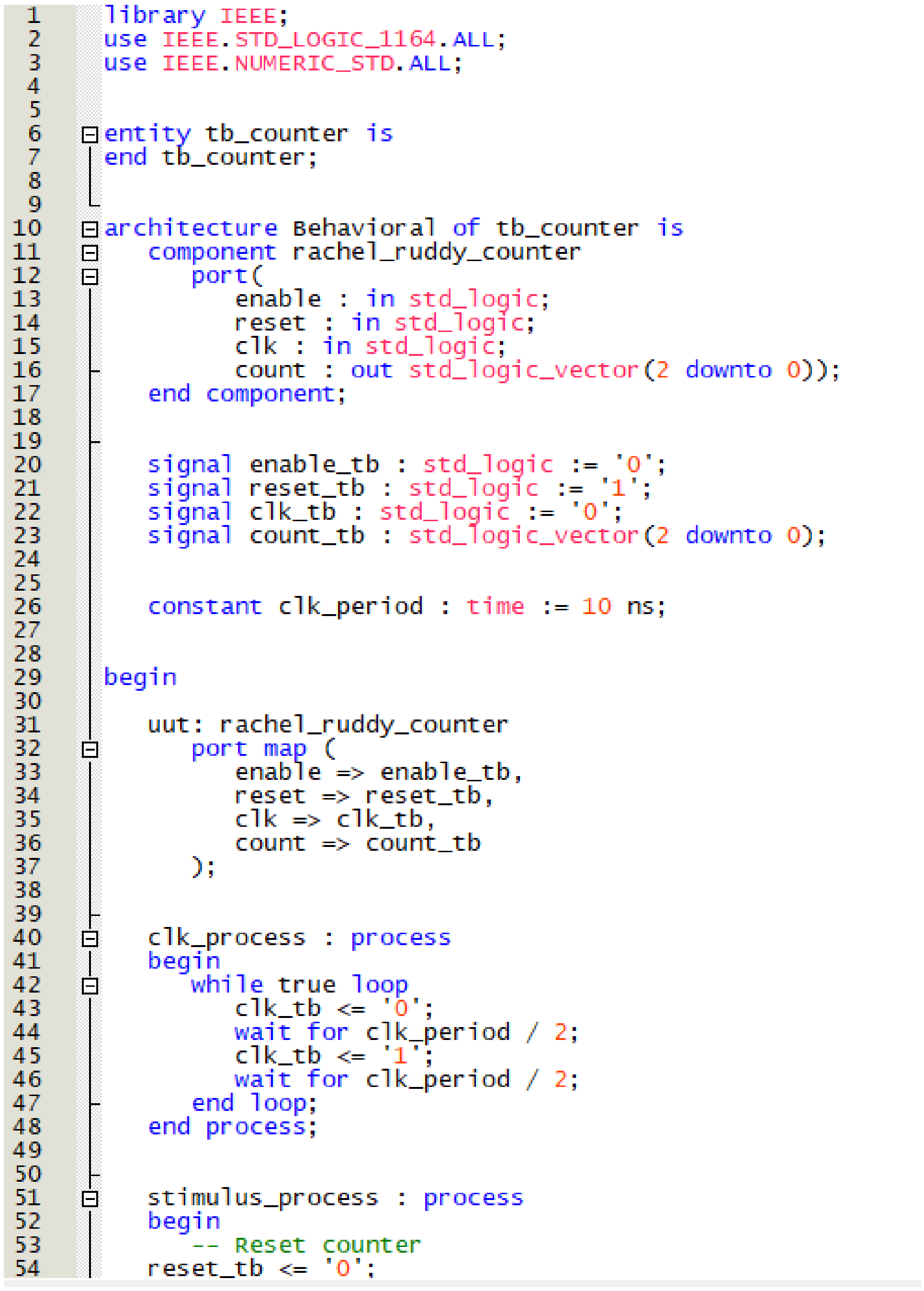
Counter:



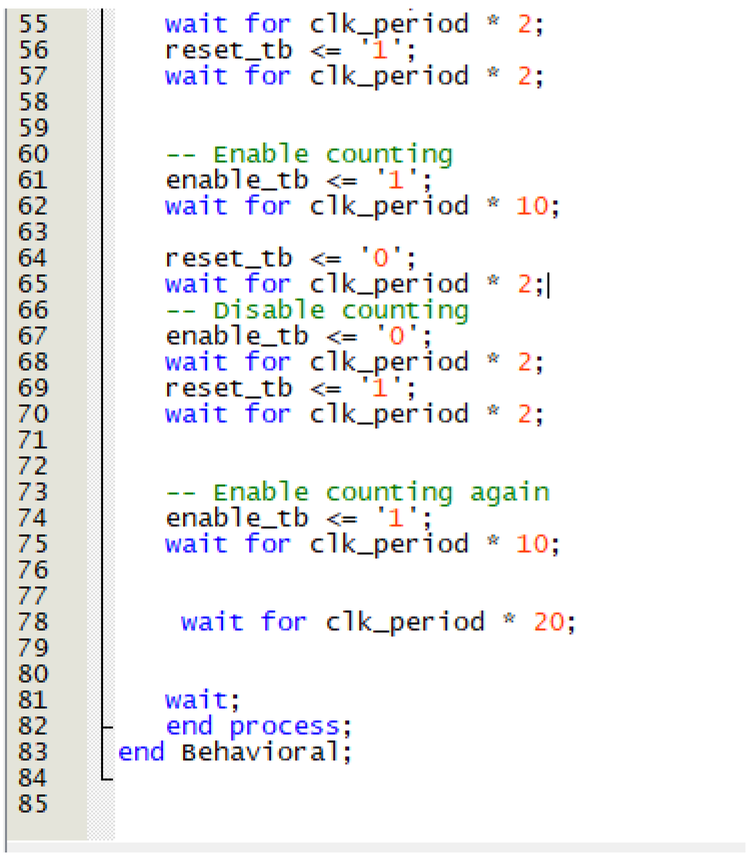
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity rachel_ruddy_counter is
    port(
        enable : in std_logic;
        reset : in std_logic;
        clk    : in std_logic;
        count  : out std_logic_vector(2 downto 0)
    );
end rachel_ruddy_counter;

architecture Behavioural of rachel_ruddy_counter is
    signal count_reg : unsigned(2 downto 0) := (others => '0');
begin
    process (clk, reset)
    begin
        if reset = '0' then
            count_reg <= (others => '0');
        elsif rising_edge(clk) then
            if enable = '1' then
                if count_reg = "111" then
                    count_reg <= (others => '0');
                else
                    count_reg <= count_reg + 1;
                end if;
            end if;
        end if;
    end process;

    count <= std_logic_vector(count_reg);
end Behavioural;
```

The counter module is a behavioral description of a 3-bit synchronous up counter. The counter starts at a value of 0 and increments up by 1 until the counter reaches a value of 7_{10} . The counter description includes a process which is executed when the clock or reset signal is triggered. If the reset is set to be 0 (since this is an active-low reset), then the counter is immediately reset to 0. If the reset was not changed, then if the clock signal is at a rising edge, we check if enable is active. If the enable is active then we proceed with updating the counter and otherwise we end the process. If the count is equal to 7_{10}, we reset the counter to 0 and otherwise, we increment the count up by one.

Counter Testbench:



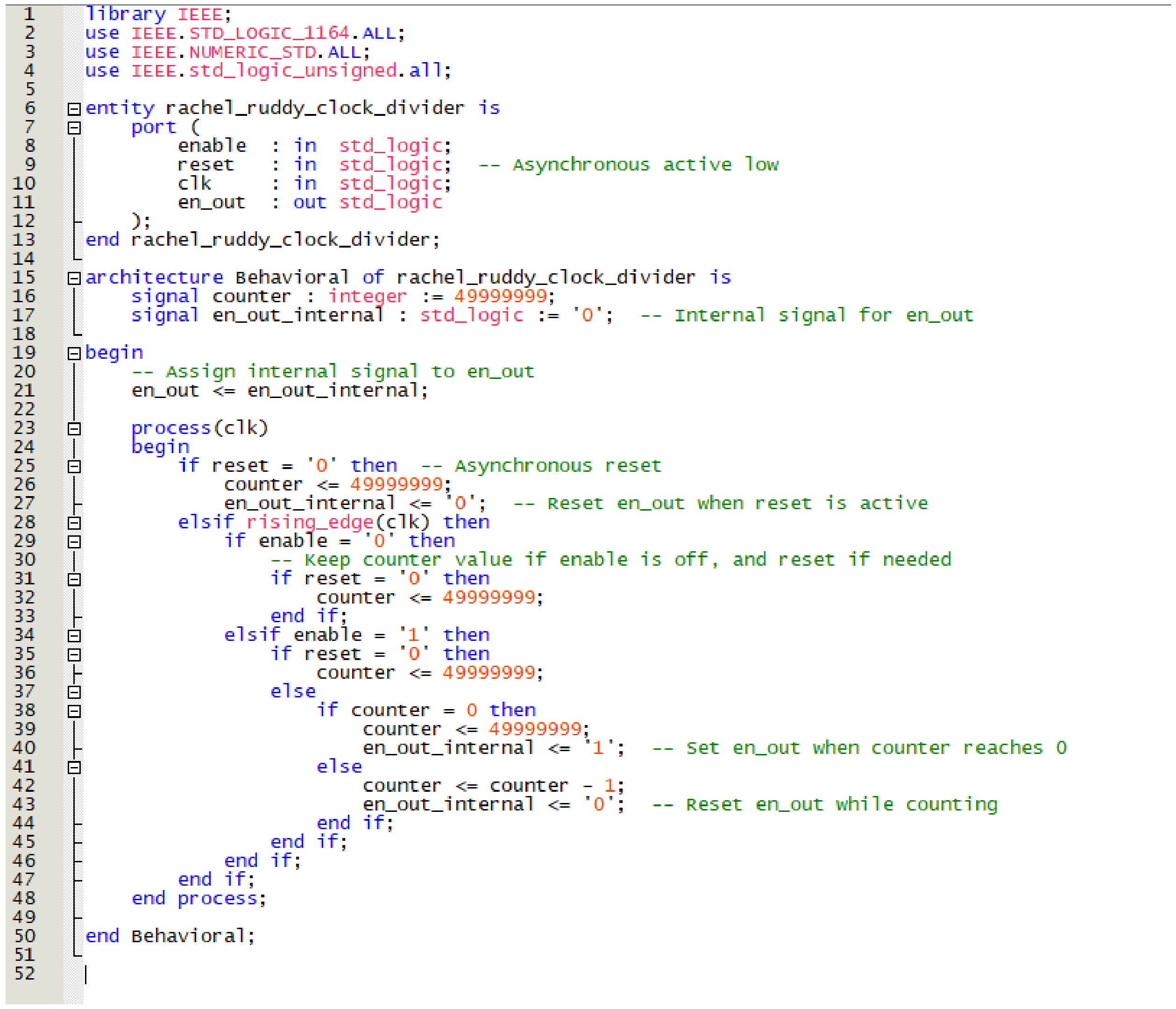
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;


entity tb_counter is
end tb_counter;


architecture Behavioral of tb_counter is
   component rachel_ruddy_counter
      port(
         enable : in std_logic;
         reset : in std_logic;
         clk : in std_logic;
         count : out std_logic_vector(2 downto 0));
   end component;


   signal enable_tb : std_logic := '0';
   signal reset_tb : std_logic := '1';
   signal clk_tb : std_logic := '0';
   signal count_tb : std_logic_vector(2 downto 0);


   constant clk_period : time := 10 ns;


begin

   uut: rachel_ruddy_counter
      port map (
         enable => enable_tb,
         reset => reset_tb,
         clk => clk_tb,
         count => count_tb
      );


   clk_process : process
   begin
      while true loop
         clk_tb <= '0';
         wait for clk_period / 2;
         clk_tb <= '1';
         wait for clk_period / 2;
      end loop;
   end process;


   stimulus_process : process
   begin
      -- Reset counter
   reset_tb <= '0';
```



```
    wait for clk_period * 2;
    reset_tb <= '1';
    wait for clk_period * 2;


    -- Enable counting
    enable_tb <= '1';
    wait for clk_period * 10;

    reset_tb <= '0';
    wait for clk_period * 2;
    -- Disable counting
    enable_tb <= '0';
    wait for clk_period * 2;
    reset_tb <= '1';
    wait for clk_period * 2;


    -- Enable counting again
    enable_tb <= '1';
    wait for clk_period * 10;


     wait for clk_period * 20;


    wait;
    end process;
end Behavioral;
```

The testbench for the counter instantiates the counter for testing. In a similar way to the JKFF testbench, we run two process blocks. One properly instantiates the clock signal while the other process sets the values of the different inputs of the counter to exhaustively test all the possible cases. The testbench specifically tests the reset to ensure that it properly sets the counter back to 0 as well as allowing the counter to run for 10 clock cycles with enable = 1 and reset = 0 to make sure it loops through 0 to 7 and then is reset to 0 again.

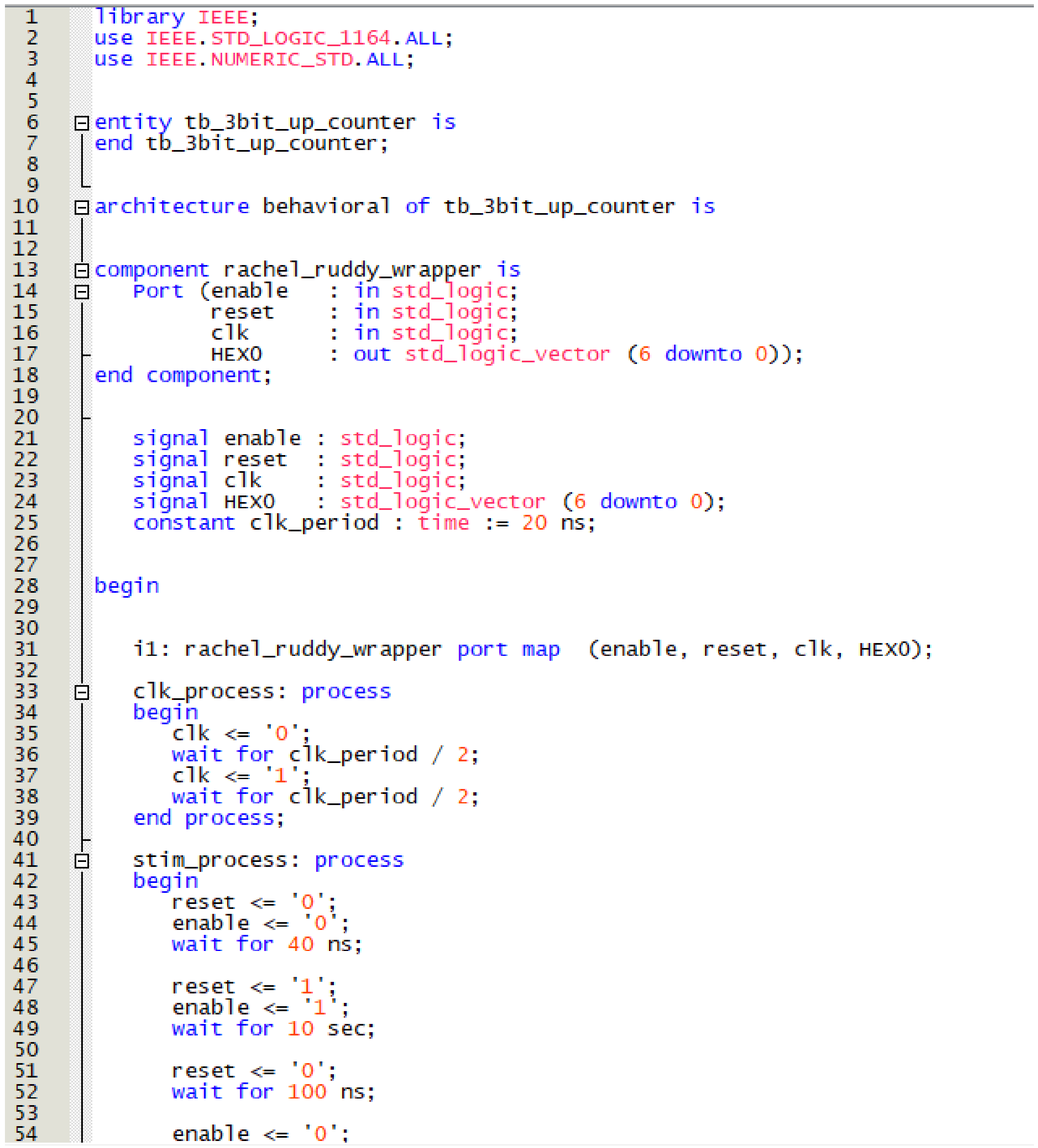
Clock Divider:



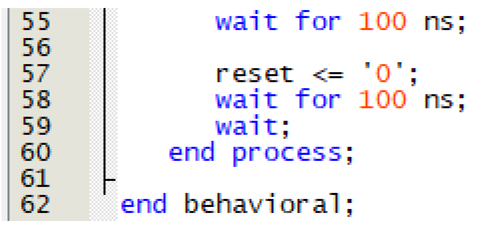
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.std_logic_unsigned.all;

entity rachel_ruddy_clock_divider is
    port (
        enable  : in  std_logic;
        reset   : in  std_logic;   -- Asynchronous active low
        clk     : in  std_logic;
        en_out  : out std_logic
    );
end rachel_ruddy_clock_divider;

architecture Behavioral of rachel_ruddy_clock_divider is
    signal counter : integer := 49999999;
    signal en_out_internal : std_logic := '0';  -- Internal signal for en_out

begin
    -- Assign internal signal to en_out
    en_out <= en_out_internal;

    process(clk)
    begin
        if reset = '0' then  -- Asynchronous reset
            counter <= 49999999;
            en_out_internal <= '0';  -- Reset en_out when reset is active
        elsif rising_edge(clk) then
            if enable = '0' then
                -- Keep counter value if enable is off, and reset if needed
                if reset = '0' then
                    counter <= 49999999;
                end if;
            elsif enable = '1' then
                if reset = '0' then
                    counter <= 49999999;
                else
                    if counter = 0 then
                        counter <= 49999999;
                        en_out_internal <= '1';  -- Set en_out when counter reaches 0
                    else
                        counter <= counter - 1;
                        en_out_internal <= '0';  -- Reset en_out while counting
                    end if;
                end if;
            end if;
        end if;
    end process;

end Behavioral;

```

To implement the logic for the clock divider, the number of clock cycles in 1 second was calculated to determine the starting value of the clock divider. We know that the initial value should be equal to the number of clock cycles - 1, since 0 is used to label the last clock cycle before the divider outputs 1. This module uses a process block which is executed at the change of the clock signal. If the reset (active low) is triggered at any point, then the count of the number of clock cycles is reset to 49,999,999, which is the starting point of the counter. Then, we check if the clock is at a rising edge. If enable is 0, we do not make any changes to the count since there should be no update. If the enable is 1, we then check for two cases. If the counter is at 0, we reset it to the initial state (49,999,999) and set the output to assert 1. Otherwise, we set the counter to be equal to counter - 1.

Clock divider testbench:



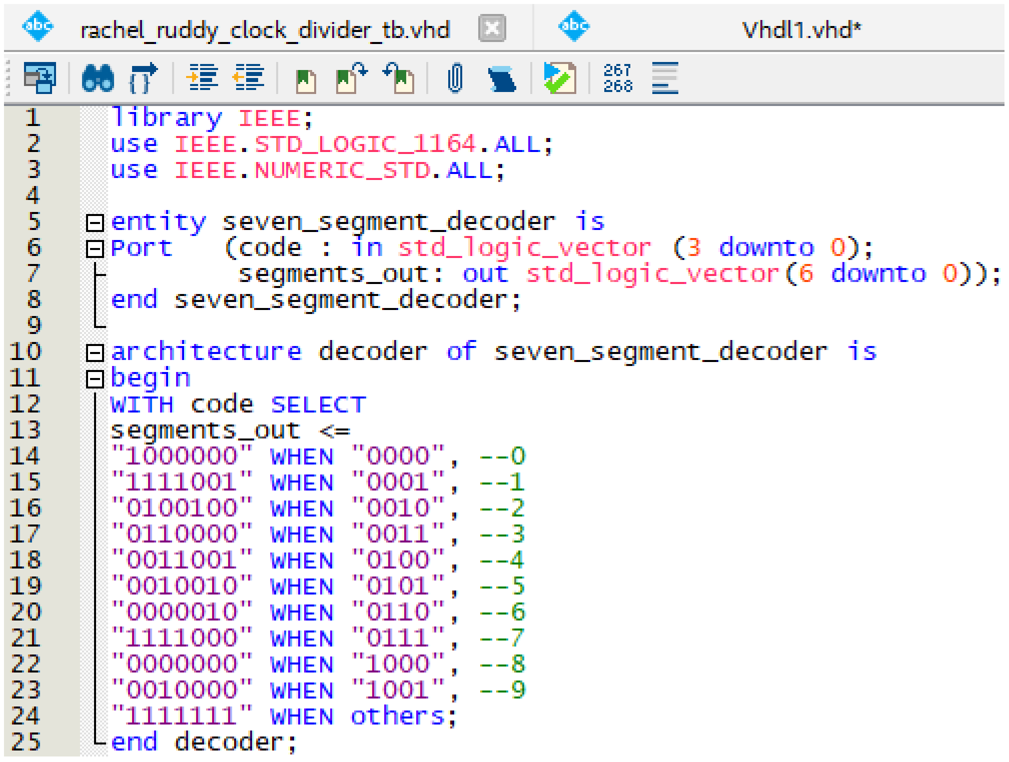
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;


entity tb_3bit_up_counter is
end tb_3bit_up_counter;


architecture behavioral of tb_3bit_up_counter is


component rachel_ruddy_wrapper is
   Port (enable    : in std_logic;
         reset     : in std_logic;
         clk       : in std_logic;
         HEX0      : out std_logic_vector (6 downto 0));
end component;


   signal enable : std_logic;
   signal reset  : std_logic;
   signal clk    : std_logic;
   signal HEX0   : std_logic_vector (6 downto 0);
   constant clk_period : time := 20 ns;


begin


   i1: rachel_ruddy_wrapper port map  (enable, reset, clk, HEX0);

   clk_process: process
   begin
      clk <= '0';
      wait for clk_period / 2;
      clk <= '1';
      wait for clk_period / 2;
   end process;

   stim_process: process
   begin
      reset <= '0';
      enable <= '0';
      wait for 40 ns;

      reset <= '1';
      enable <= '1';
      wait for 10 sec;

      reset <= '0';
      wait for 100 ns;

      enable <= '0';
```



```vhdl
      wait for 100 ns;

      reset <= '0';
      wait for 100 ns;
      wait;
   end process;

end behavioral;
```

The test bench for the clock divider first instantiates the clock in the clock process block. Then, the stimulus process block checks its functionality by first setting the initial state for a few clock cycles. Then, we test the functionality of the circuit when both reset and enable are triggered. Then, we deactivate the reset and allow the divider to run for 10 seconds to confirm that the output is 1 after each 1 second time interval. We then turn enable off to ensure that the circuit is no longer updating the counter value. Finally, we test the reset function again and wait indefinitely.

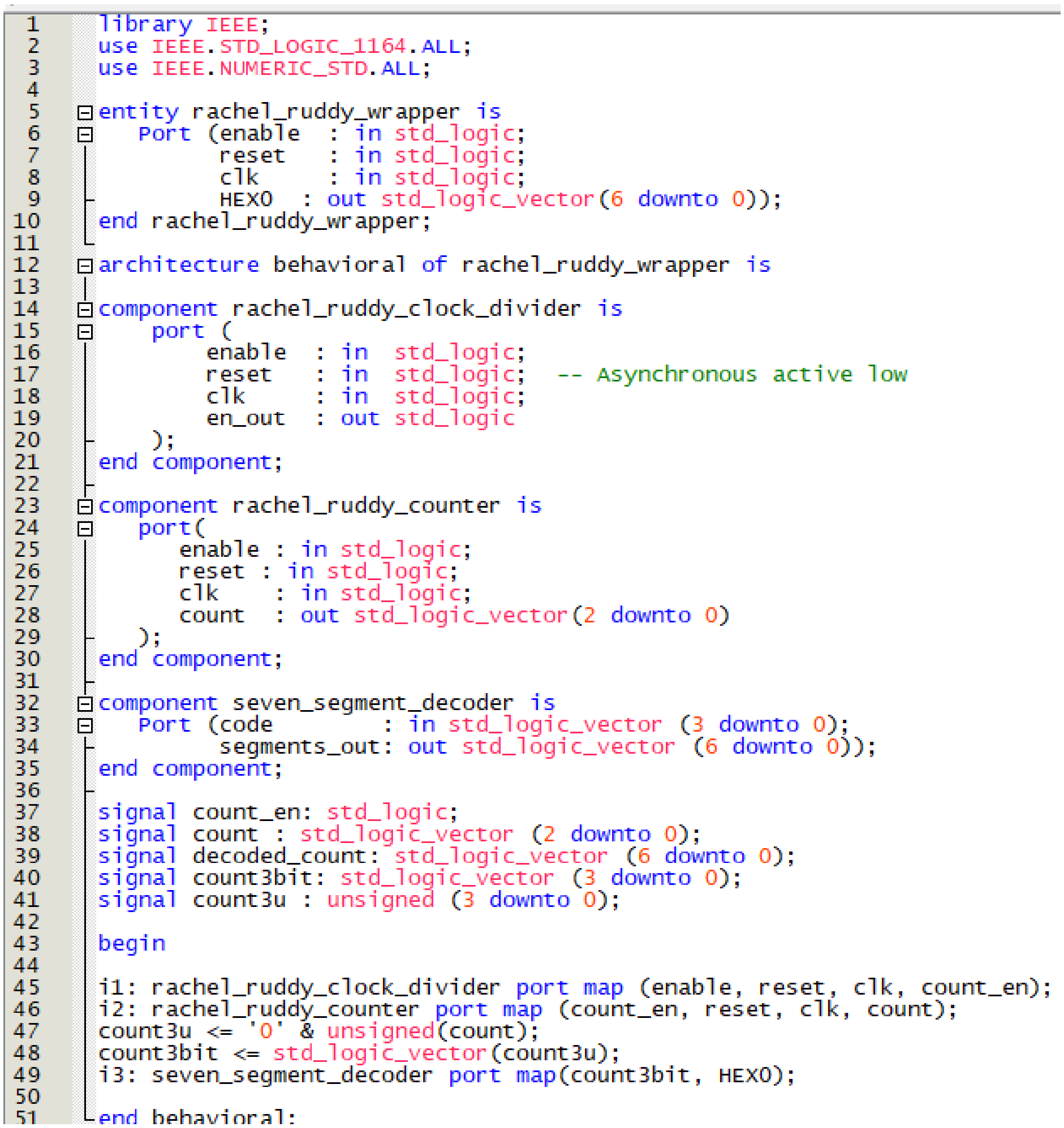
Seven Segment Decoder:



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

entity seven_segment_decoder is
Port    (code : in std_logic_vector (3 downto 0);
         segments_out: out std_logic_vector(6 downto 0));
end seven_segment_decoder;

architecture decoder of seven_segment_decoder is
begin
WITH code SELECT
segments_out <=
"1000000" WHEN "0000", --0
"1111001" WHEN "0001", --1
"0100100" WHEN "0010", --2
"0110000" WHEN "0011", --3
"0011001" WHEN "0100", --4
"0010010" WHEN "0101", --5
"0000010" WHEN "0110", --6
"1111000" WHEN "0111", --7
"0000000" WHEN "1000", --8
"0010000" WHEN "1001", --9
"1111111" WHEN others;
end decoder;
```

The seven-segment decoder file was provided in lab #4 and instantiated in the wrapper file to display the timed 3-bit up counter on the FPGA board.

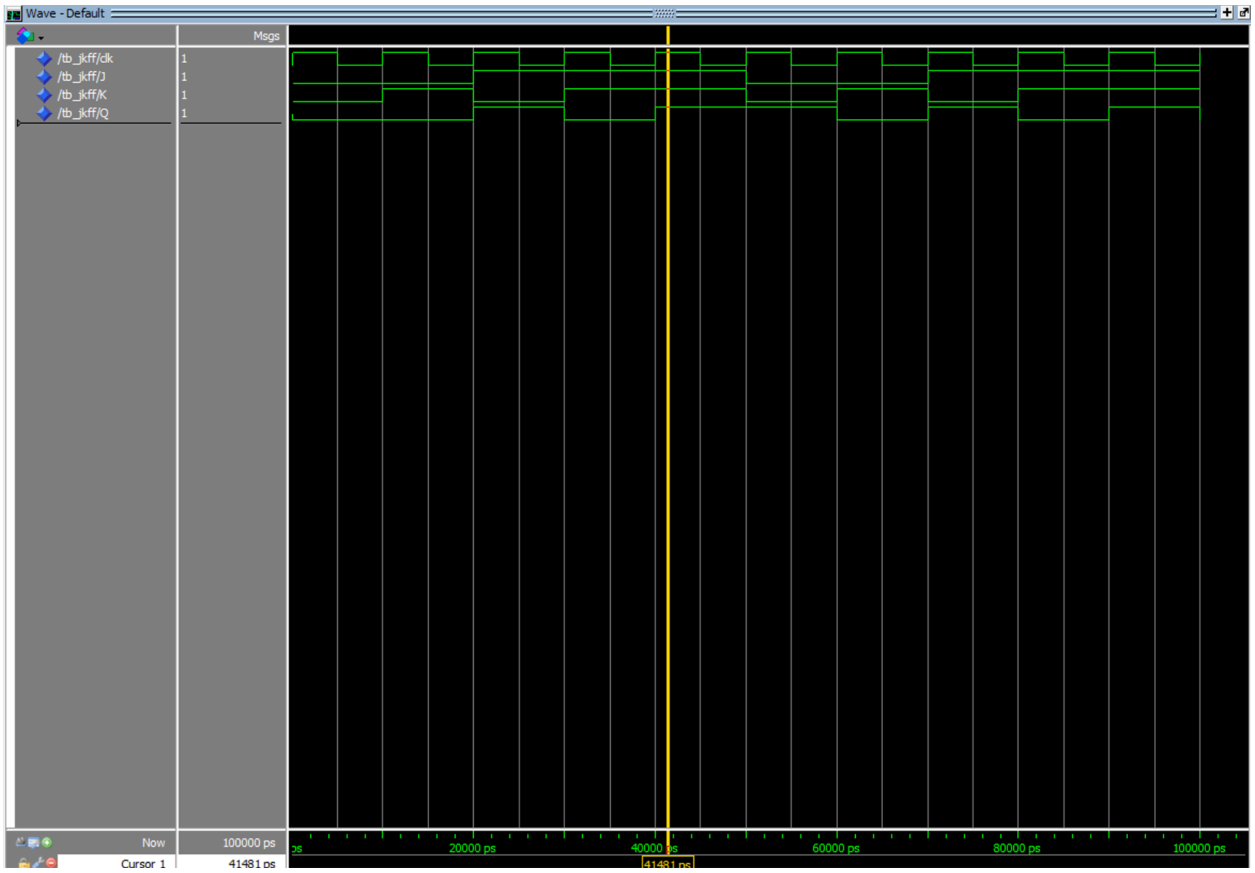
Wrapper:



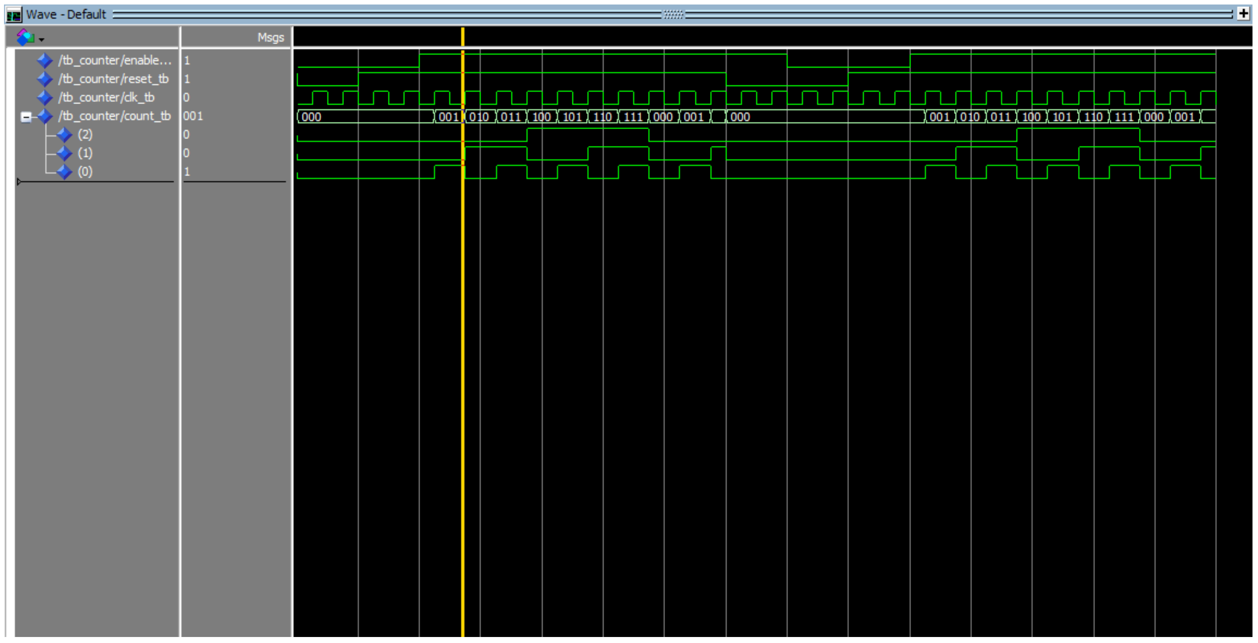
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity rachel_ruddy_wrapper is
   Port (enable  : in std_logic;
         reset   : in std_logic;
         clk     : in std_logic;
         HEX0   : out std_logic_vector(6 downto 0));
end rachel_ruddy_wrapper;

architecture behavioral of rachel_ruddy_wrapper is

component rachel_ruddy_clock_divider is
    port (
        enable  : in  std_logic;
        reset   : in  std_logic;  -- Asynchronous active low
        clk     : in  std_logic;
        en_out  : out std_logic
    );
end component;

component rachel_ruddy_counter is
   port(
      enable : in std_logic;
      reset : in std_logic;
      clk    : in std_logic;
      count  : out std_logic_vector(2 downto 0)
   );
end component;

component seven_segment_decoder is
   Port (code          : in std_logic_vector (3 downto 0);
         segments_out: out std_logic_vector (6 downto 0));
end component;

signal count_en: std_logic;
signal count : std_logic_vector (2 downto 0);
signal decoded_count: std_logic_vector (6 downto 0);
signal count3bit: std_logic_vector (3 downto 0);
signal count3u : unsigned (3 downto 0);

begin

i1: rachel_ruddy_clock_divider port map (enable, reset, clk, count_en);
i2: rachel_ruddy_counter port map (count_en, reset, clk, count);
count3u <= '0' & unsigned(count);
count3bit <= std_logic_vector(count3u);
i3: seven_segment_decoder port map(count3bit, HEX0);

end behavioral;
```

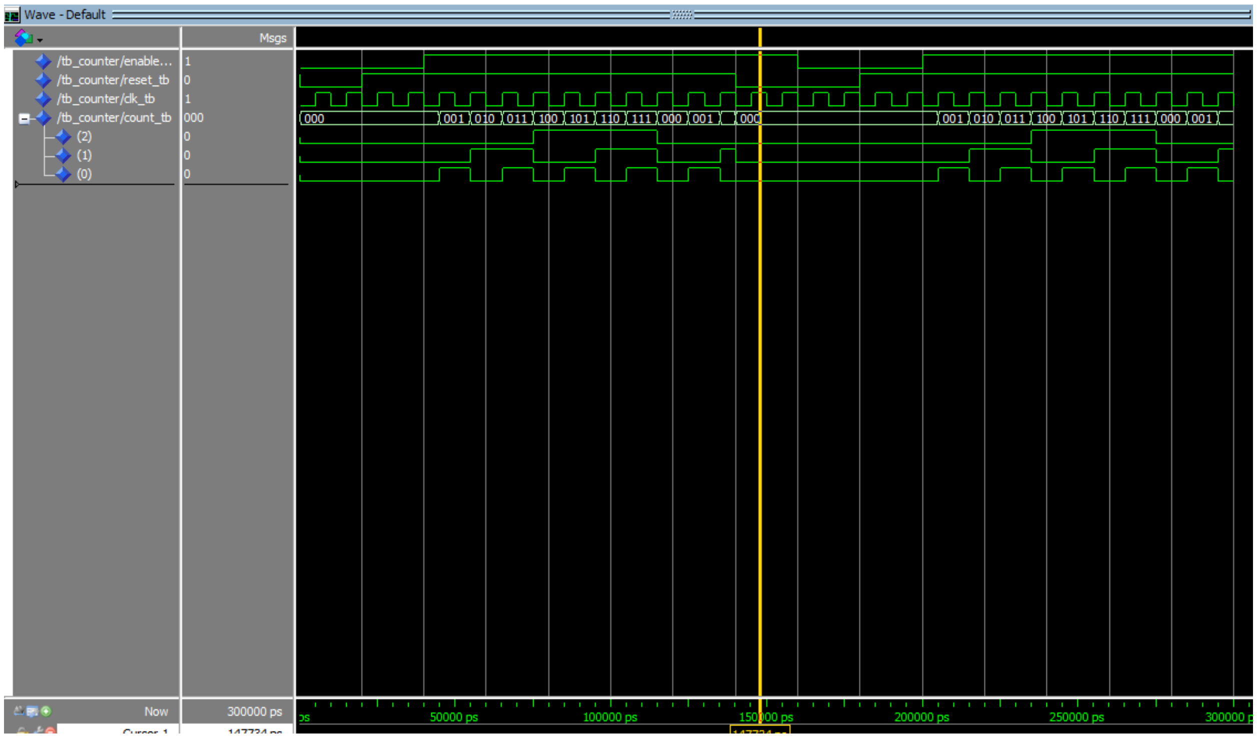
The wrapper file creates 3 components: a clock divider, a 3-bit up counter, and a 7-segment decoder. Additionally, signals were created to ensure that the inputs to the seven segment display were of the correct size to avoid compilation errors. The components are instantiated and the inputs are properly mapped to each component so that the counter is only enabled when the clock divider asserts 1.

JK Flip-Flop Waveform:

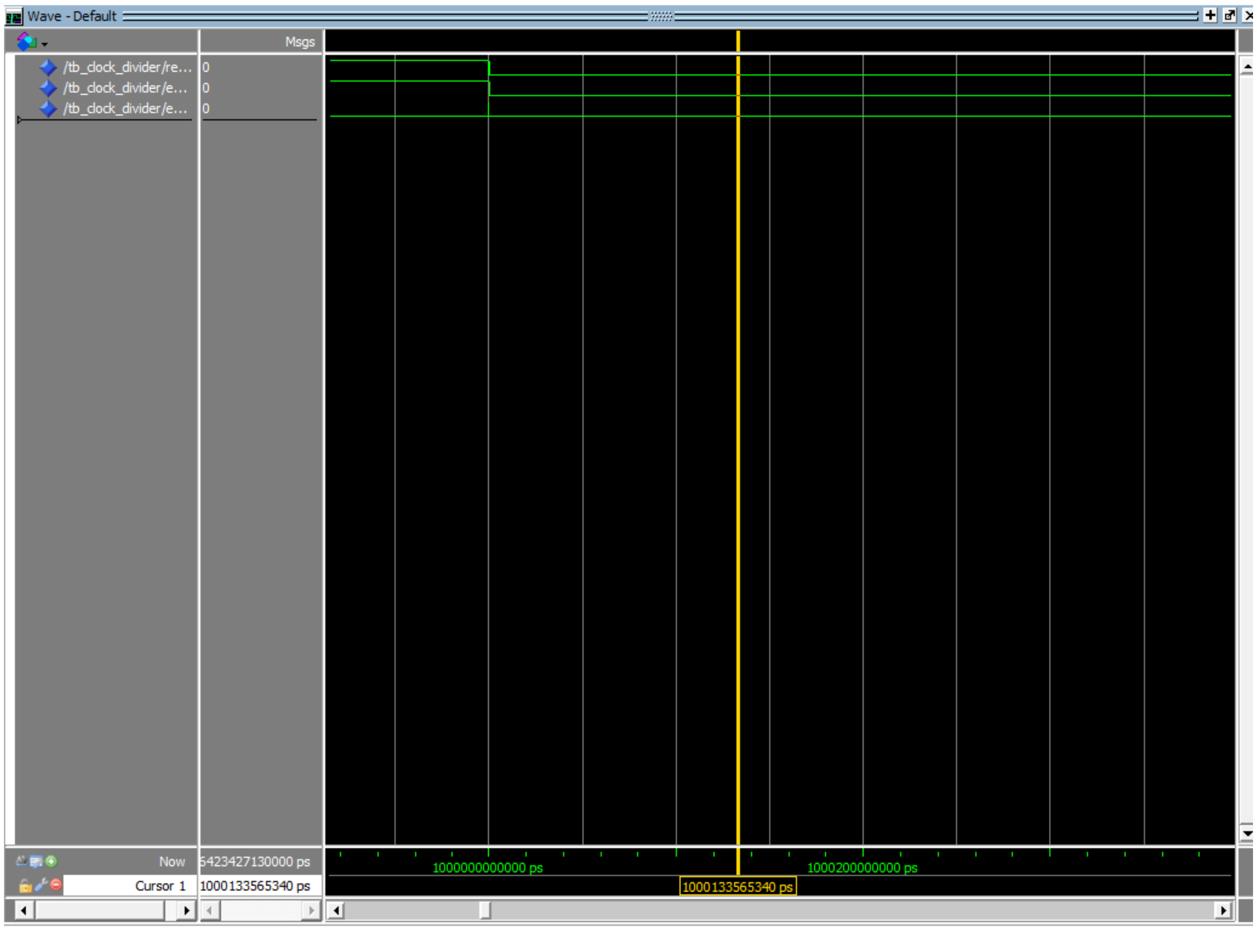


Counter Waveform:

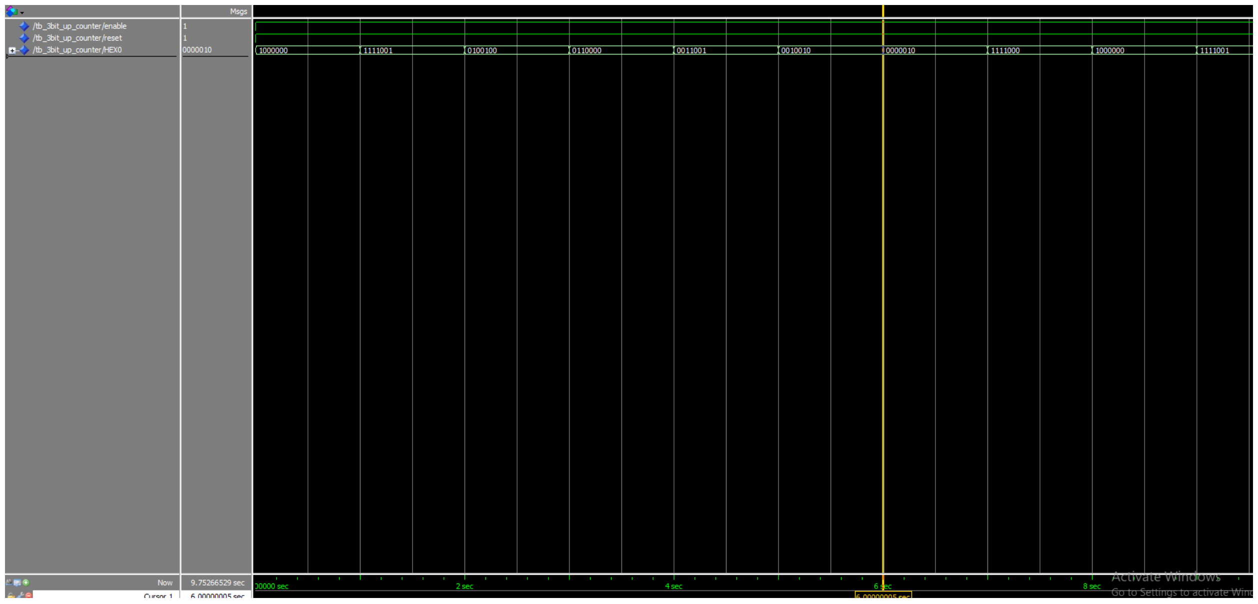


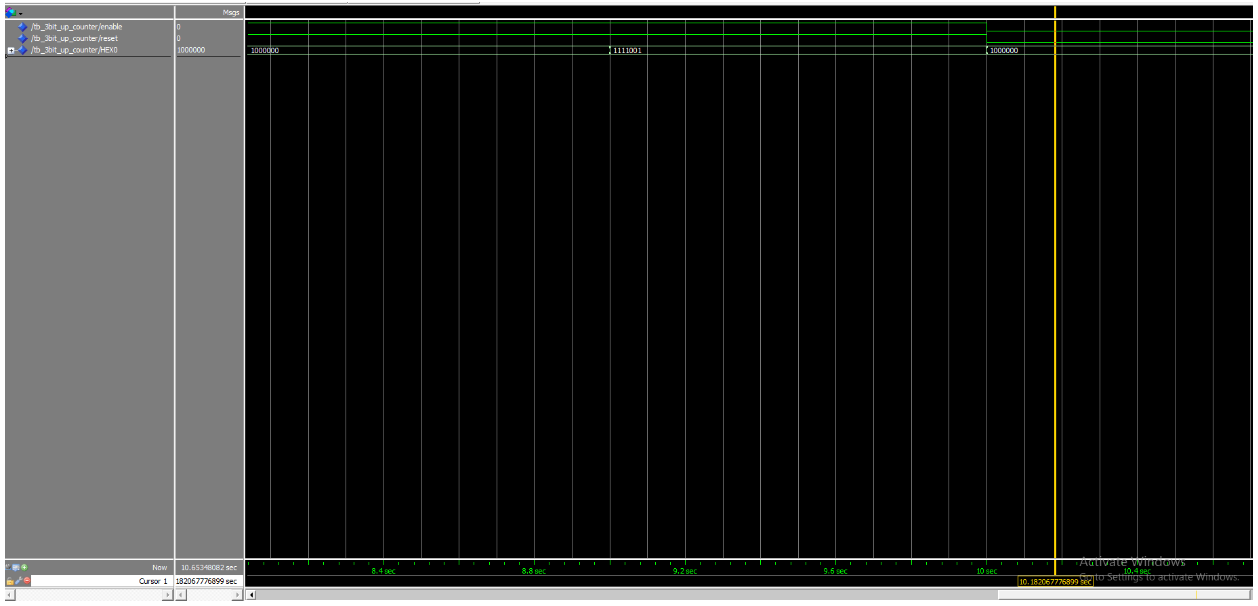


Clock Divider after 1 second :



wrapper:



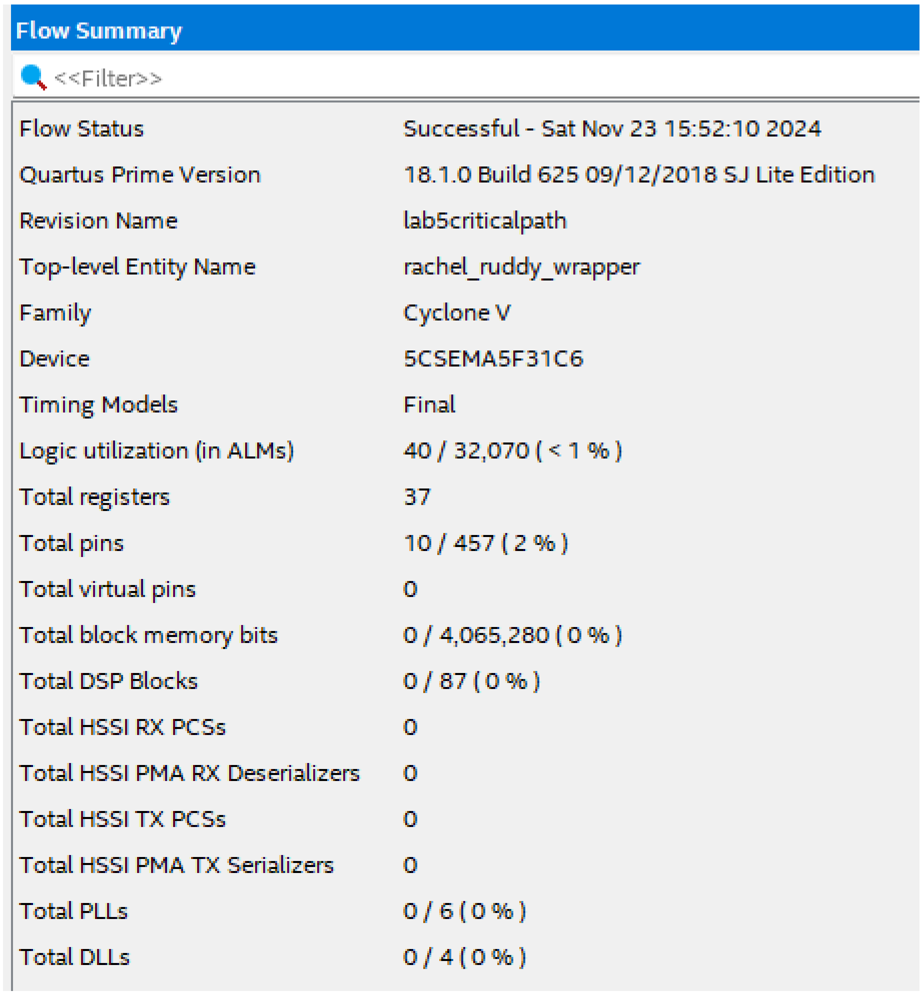


In accordance with our test bench, we see the 3 bit counter working and then resetting after 10 seconds.

3. Perform timing analysis (slow 1,100 mV, 85C model) of the 3-bit up counter and find the critical path(s) of the circuit. What is the delay of the critical path(s)?

We sent out an email regarding this part. Similarly to Lab 4, after troubleshooting we still got the message "Nothing to report" on our timing analysis, despite our waveform being correct and our sdc file being set up correctly too.

4. Report the number of pins and logic modules used to fit your 3-bit up counter design on the FPGA board.



Flow Summary

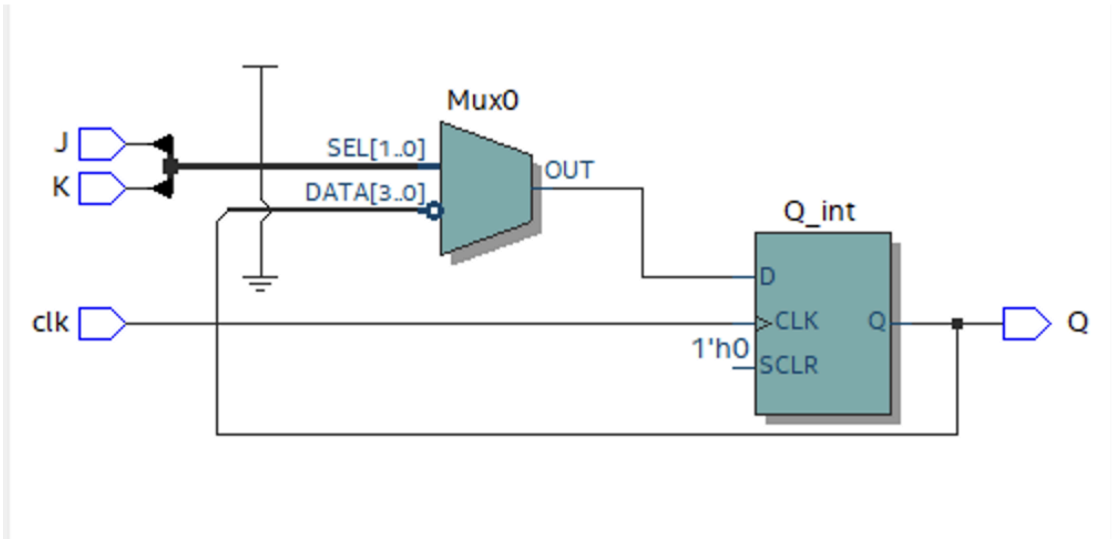
<<Filter>>

Flow Status	Successful - Sat Nov 23 15:52:10 2024
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	lab5criticalpath
Top-level Entity Name	rachel_ruddy_wrapper
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	40 / 32,070 (< 1 %)
Total registers	37
Total pins	10 / 457 (2 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

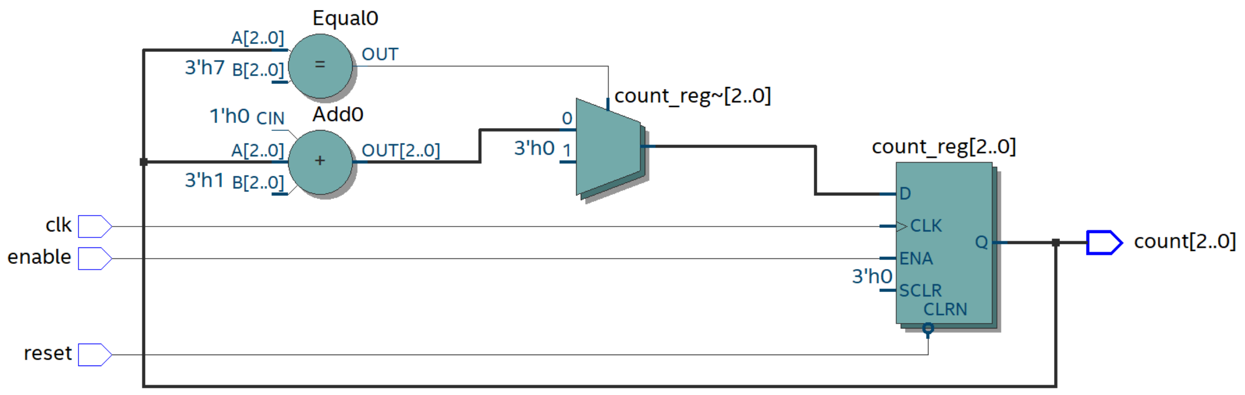
10 pins and 40 ALMs.

III. Schematics and simulation results (and explanations/labels of figures/important points on the simulation plots)

JKFF:



Counter:



IV. Conclusions:

In this lab, we successfully designed and implemented several sequential circuits, including a JK flip-flop, a 3 bit up counter, and a clock divider using VHDL. Using the FPGA board to test the wrapper circuit demonstrated the practical application of sequential circuit design, with the counter accurately incrementing each second with its expected display. This lab reinforced key concepts in sequential circuit design and provided practical skills in testing and FPGA development.