David Castro

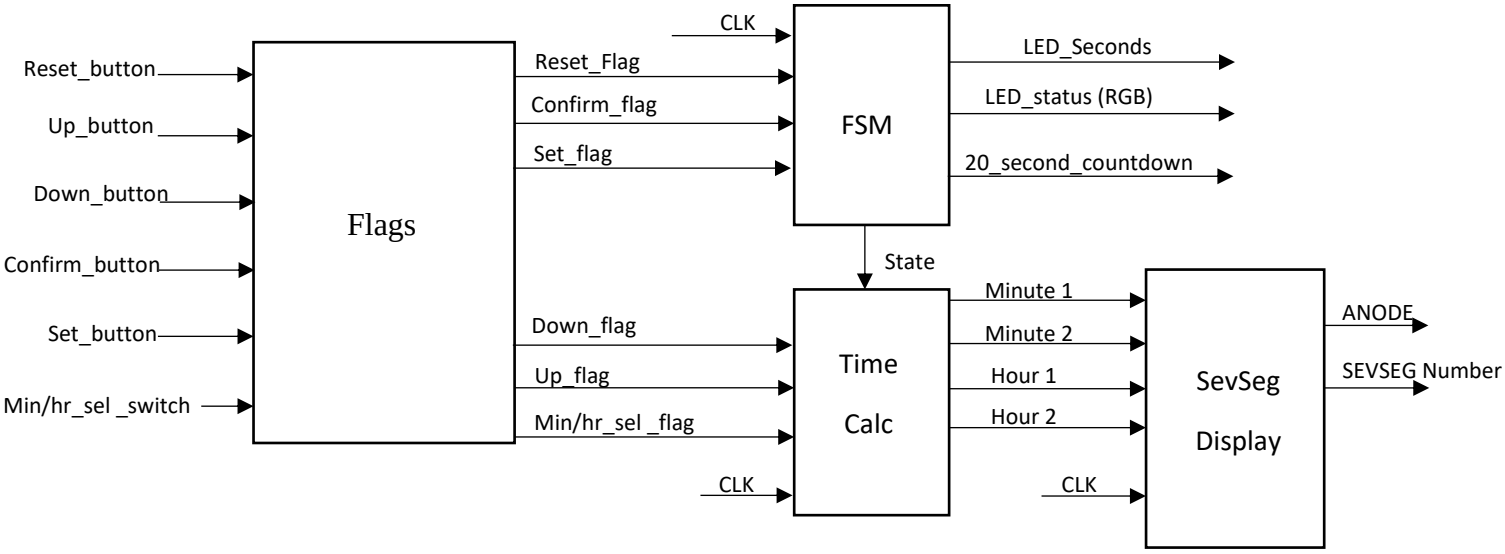
CWID: 898162227

EGEE-448

LAB – 4

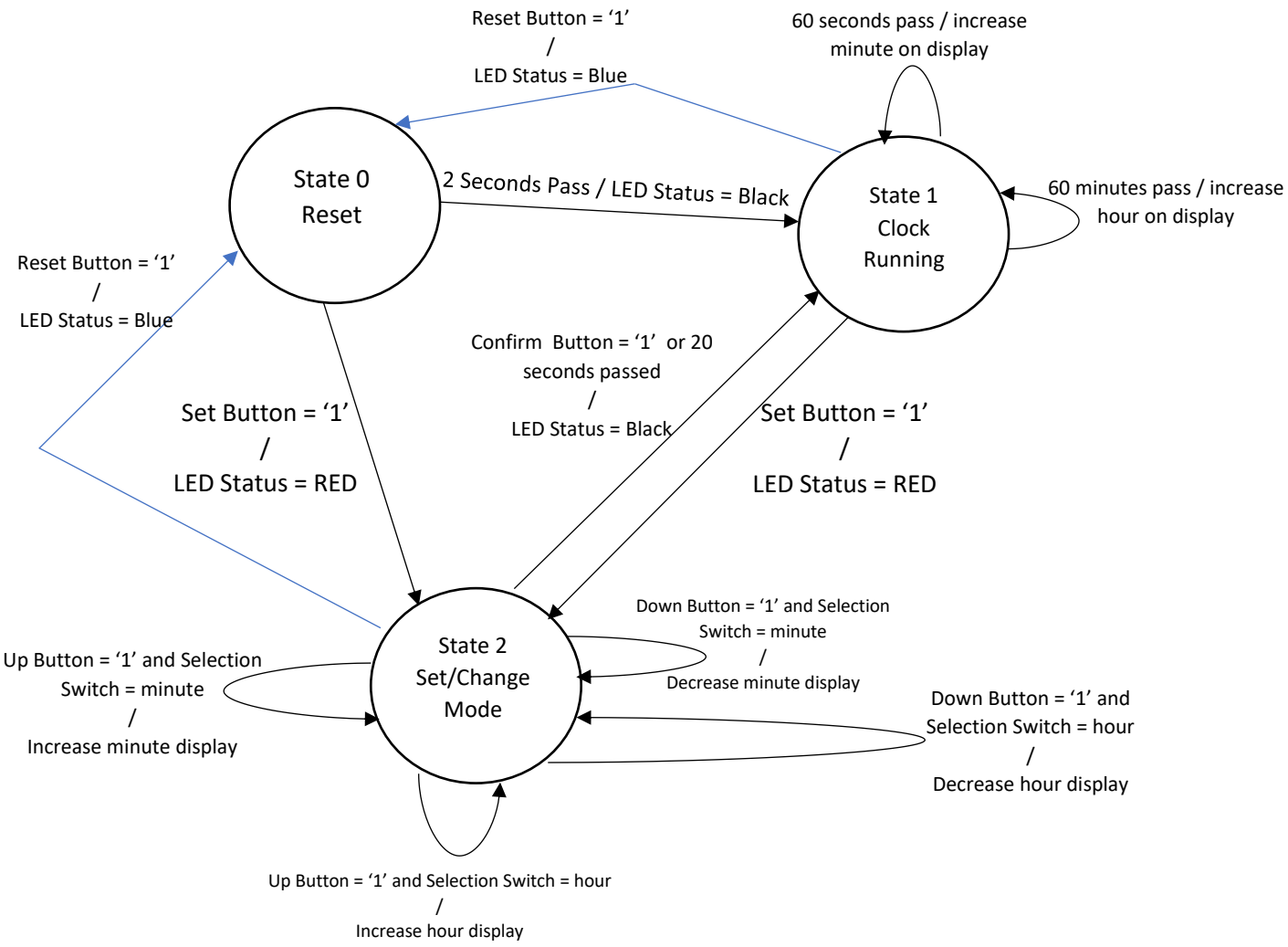
Digital Clock

1) Based on the above project design requirements, how will you design this digital clock system? Please draw below a block diagram for the subcomponents needed for this system; and explain briefly the functionality of each block/subsystem.



The Flags subsystem basically takes all the inputs and sets the flags. It's a way of making things more streamlined so that all the inputs and their flags could be placed in one block. The FSM block controls the states the system is in, and the LED's outputs such as the RGB for the status. It also has the counter for the seconds since for some of the states it requires counting seconds to transition to the next state so all the calculations having to do with seconds was implemented in the FSM block. From here it will output the current state to Time Calc. Time Calc handles all the calculations for the clock display. It will increment by one minute on the seven segment display when 60 seconds passes or increases the time when the up button is pushed in the set state. It basically controls the numbers being outputted to the seven segment display. The SevSeg Display block controls the seven segment display. It will make sure the clock appears correctly and the numbers are in the right order at the correct refresh rate.

2) Additionally, what will be the state diagram describing the transition of states among the above-mentioned three states and their corresponding input/outputs?



No extra states were added. It's the standard 3 states: reset, running, and set/change. Although it could be setup with more states, I kept it to the 3 states and just made it so that inputs or conditions in that state would trigger an output in that state. For example, with the Set/Change state if the up button was pressed it would stay in the same state, but the minute or hour would be increased on the display. Everything follows the instructions where the transitions occur off either a desired time such as 2 seconds in the reset state or a button being pressed like the Set button.

3) Per your design, what is your VHDL entity code for this digital clock system? Please insert below the source code for the VHDL entity file(s) and describe the functionality of each VHDL entity file included in your project.

There are 5 entity files. Four of them are the blocks in the first question while the 5th one is the structural entity that connects them all together.

Structural File:

The Structural File puts everything together. It connects the entity files together where it will send outputs from one file to inputs of another. There is one extra thing in the entity file and that's the duty cycle for the status LED. This requires using a clock scaler converting the internal 100 MHz clock to a 100 Hz clock so there is code for doing this in this file.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity clock_structure is
    Port(
        clk: in std_logic;
        reset_button: in std_logic;
        up_button: in std_logic;
        down_button: in std_logic;
        confirm_button: in std_logic;
        set_button: in std_logic;
        min_hr_switch: in std_logic_vector(1 downto 0);
        SEVSEG: out std_logic_vector(7 downto 0);
        ANODE: out std_logic_vector(7 downto 0);
        --led status (RGB)
        LED_status: out std_logic_vector(2 downto 0);
      --second display for LEDS
        LED_seconds: out std_logic_vector(5 downto 0);
        LED_seconds_countdown: out std_logic_vector(5 downto 0)
    );
end clock_structure;
```

```
architecture Behavioral of clock_structure is
    --signals to interconnect the entities together
        signal signal_reset_flag: std_logic;
        signal signal_up_flag: std_logic;
        signal signal_down_flag: std_logic;
        signal signal_set_flag: std_logic;
        signal signal_confirm_flag: std_logic;
        signal signal_min_hr_flag: std_logic_vector(1 downto 0);
        signal signal_clk_display_min_1: std_logic_vector(3 downto 0);
        signal signal_clk_display_min_2: std_logic_vector(3 downto 0);
        signal signal_clk_display_hr_1: std_logic_vector(3 downto 0);
        signal signal_clk_display_hr_2: std_logic_vector(3 downto 0);
        signal signal_LED_status: std_logic_vector(2 downto 0);
        signal state_signal: Integer Range 0 to 2;

        --intiailzing a counter for the slow clock
        signal counter: Integer := 0;

        --intializing a slow clock for 100 HZ
        signal slw_clk: std_logic;



  --entities being used
    Component flags
        Port(
        reset_button: in std_logic;
        up_button: in std_logic;
        down_button: in std_logic;
        confirm_button: in std_logic;
        set_button: in std_logic;
        min_hr_switch: in std_logic_vector(1 downto 0);
        reset_flag: out std_logic;
```

```
    up_flag: out std_logic;
    down_flag: out std_logic;
    set_flag: out std_logic;
    confirm_flag: out std_logic;
    min_hr_flag: out std_logic_vector(1 downto 0)
);
end Component;

Component state_machine_clock
    Port(
    --clk input for hardware
    clk: in std_logic;
    --flag inputs
    reset_flag: in std_logic;
    set_flag: in std_logic;
    confirm_flag: in std_logic;
  --led status (RGB)
    LED_status: out std_logic_vector(2 downto 0);
  --second display for LEDS
    LED_seconds: out std_logic_vector(5 downto 0);
    LED_seconds_countdown: out std_logic_vector(5 downto 0);
    state_output: out Integer Range 0 to 2
);
end Component;

Component time_calc
    Port(
        clk: in std_logic;
        state: in Integer Range 0 to 2;
        up_flag: in std_logic;
        down_flag: in std_logic;
        min_hr_flag: in std_logic_vector(1 downto 0);
```

```
            clk_display_min_1: out std_logic_vector(3 downto 0);
            clk_display_min_2: out std_logic_vector(3 downto 0);
            clk_display_hr_1: out std_logic_vector(3 downto 0);
            clk_display_hr_2: out std_logic_vector(3 downto 0)
        );
    end Component;

    Component sevseg_display
        Port(
        clk: in std_logic;
        clk_display_min_1: in std_logic_vector(3 downto 0);
        clk_display_min_2: in std_logic_vector(3 downto 0);
        clk_display_hr_1: in std_logic_vector(3 downto 0);
        clk_display_hr_2: in std_logic_vector(3 downto 0);
--        alarm_display_min: in std_logic_vector(5 downto 0);
--        alarm_display_hr: in std_logic_vector(4 downto 0);
        SEVSEG: out std_logic_vector(7 downto 0);
        ANODE: out std_logic_vector(7 downto 0)
    );
    end Component;

begin

    --Process for implementing a slow clock
    --Internal CLK is 100MHZ
    --want a 100 HZ clk
    --therefore 100MHZ / 1MHZ = 100 HZ clk
    SLOW_CLK: Process(clk)
        begin
            --act as a counter for the clock
            if(rising_edge(clk)) then
                if(counter = 1000000) then  --resets back to 0 when 1MHZ
```

```
            counter <= 0;
        else
            counter <= counter + 1;
        end if;
    end if;

end Process SLOW_CLK;



--connecting the entities together (structure)
intial_inputs: flags Port Map
    (
        reset_button => reset_button,
        up_button => up_button,
        down_button => down_button,
        confirm_button => confirm_button,
        set_button => set_button,
        min_hr_switch => min_hr_switch,
        reset_flag => signal_reset_flag,
        up_flag => signal_up_flag,
        down_flag => signal_down_flag,
        set_flag => signal_set_flag,
        confirm_flag => signal_confirm_flag,
        min_hr_flag => signal_min_hr_flag
    );

FSM: state_machine_clock Port Map
    (
        clk => clk,
        reset_flag => signal_reset_flag,
        set_flag => signal_set_flag,
        confirm_flag => signal_confirm_flag,
```

```
        LED_status => signal_LED_status,
        LED_seconds => LED_seconds,
        LED_seconds_countdown => LED_seconds_countdown,
        state_output => state_signal
    );

timer: time_calc Port Map
    (
        clk => clk,
        state => state_signal,
        up_flag => signal_up_flag,
        down_flag => signal_down_flag,
        min_hr_flag => signal_min_hr_flag,
        clk_display_min_1 => signal_clk_display_min_1,
        clk_display_min_2 => signal_clk_display_min_2,
        clk_display_hr_1 => signal_clk_display_hr_1,
        clk_display_hr_2 => signal_clk_display_hr_2
    );

SevenSegment_dipslay: sevseg_display Port Map
    (
        clk => clk,
        clk_display_min_1 => signal_clk_display_min_1,
        clk_display_min_2 => signal_clk_display_min_2,
        clk_display_hr_1 => signal_clk_display_hr_1,
        clk_display_hr_2 => signal_clk_display_hr_2,
        SEVSEG => SEVSEG,
        ANODE => ANODE
    );



--in order to get 10% duty cycle we want the slow clock on for 10%
```

```
    --10% of the slow_clock is 10% of the counter (100,000)
    slw_clk <= '1' when counter <= 100000 else '0';
    LED_status <= signal_LED_status when slw_clk = '1' else "000";
end Behavioral;
```

Flags:

This entity is for setting flags based off inputs. The flags entity is not necessary in this case. The inputs could directly go to other entity files, since I did not generate any extra flags off these inputs. In case though the code required more flags, I left the entity in place so that modification could be made easier for later.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity flags is
    Port(
        reset_button: in std_logic;
        up_button: in std_logic;
        down_button: in std_logic;
        confirm_button: in std_logic;
        set_button: in std_logic;
        min_hr_switch: in std_logic_vector(1 downto 0);
        reset_flag: out std_logic;
        up_flag: out std_logic;
        down_flag: out std_logic;
        set_flag: out std_logic;
        confirm_flag: out std_logic;
        min_hr_flag: out std_logic_vector(1 downto 0)
    );
end flags;

architecture Behavioral of flags is
```

```
    begin


        --setting the flags based off inputs
        reset_flag <= '1' when reset_button = '1' else '0';
        up_flag <= '1' when up_button = '1' else '0';
        down_flag <= '1' when down_button = '1' else '0';
        confirm_flag <= '1' when confirm_button = '1' else '0';
        set_flag <= '1' when set_button = '1' else '0';


        min_hr_flag <= "00" when min_hr_switch = "00" else
                    "01" when min_hr_switch = "01" else
                    "10" when min_hr_switch = "10" else
                    "11";
end Behavioral;
```

State Machine:

The state machine handles the states based off the flags and conditions. There is a clock prescaler for 1 second in this code because some of the transitions between states require a waiting time of two seconds or twenty seconds. Since I want a two second delay or twenty second delay, I end up having two processes that count twenty seconds and two seconds. This is a state machine so there are processes for setting the states it should enter. In here the RGB LED (LED Status) will be set to the color that we have defined for being in that state. Finally, there is the seconds counter. This will output the binary number for the number of seconds that has passed in the current minute to the LEDs.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;


entity state_machine_clock is
    Port(
        --clk input for hardware
```

```
        clk: in std_logic;
        --flag inputs
        reset_flag: in std_logic;
        set_flag: in std_logic;
        confirm_flag: in std_logic;
       --led status (RGB)
        LED_status: out std_logic_vector(2 downto 0);
       --second display for LEDS
        LED_seconds: out std_logic_vector(5 downto 0);
        LED_seconds_countdown: out std_logic_vector(5 downto 0);
        state_output: out Integer Range 0 to 2
    );
end state_machine_clock;


architecture Behavioral of state_machine_clock is

    --signal  for states
    signal state: Integer range 0 to 2;
    signal next_state: Integer range 0 to 2;

    --signal for making a slow clock that is 1 Hz
    signal clkCnt: Integer := 0;
    signal slowClk: std_logic;

    --signal for seconds counter
    signal seconds: std_logic_vector(5 downto 0);

    --signal for two second delay in reset state
    signal two_seconds: Integer := 0;

    --signal for 20 seconds delay in set/change state
```

```
    signal twenty_seconds: std_logic_vector(5 downto 0);

begin

  --clk scaler with 100 MHz clk (make 1 Hz clock)
  second_Prescaler: Process(clk)
   begin
    if rising_edge(clk) then
        if (clkCnt = 100000000 - 1) then -- if counts up to 100 million
            clkCnt <= 0;    -- when one second passes reset counter
        else
            clkCnt <= clkCnt + 1; --incrementing counter to simulate seconds
        end if;
    end if;
  end process second_Prescaler;



   --process that counts the seconds of each minute
    seconds_counter: Process(clk, slowClk, state)
        begin
            if(rising_edge(clk)) then
                if(slowClk = '1') then
                    --reason for being 59 is because of how statement is written
                    if(seconds ="111011") then    --reset back to 0 when 60 seconds pass
                        seconds <= "000000";
                    else
                        seconds <= seconds + "000001"; --increment seconds counter
                    end if;
                end if;
            end if;

            --reseting seconds off reset input
```

```
        if(state = 0 or state = 2) then
            seconds <= "000000";
        end if;
 end Process seconds_counter;

--process that counts 2 seconds
two_second_delay: Process(clk, slowClk, state)
    begin
        if(rising_edge(clk)) then
            if(slowClk = '1' and state = 0) then
                --incrementing by 1 second
                two_seconds <= two_seconds + 1;
            end if;
        end if;

        --reseting seconds after done with reset state
        if(state = 1 or state = 2) then
            two_seconds <= 0;
        end if;
 end Process two_second_delay;

--process that counts to 20 seconds
twenty_second_delay: Process(clk, slowClk, state)
    begin
        if(rising_edge(clk)) then
            if(slowClk = '1' and state = 2) then
                --incrementing by 1 second
                twenty_seconds <= twenty_seconds + "000001";
            end if;
        end if;

        --reseting seconds after done with reset state
```

```
        if(state = 0 or state = 1) then
            twenty_seconds <= "000000";
        end if;
 end Process twenty_second_delay;

--Process that will tranistion the state to the nextstate
--using slowClk to act as a slight debounce
transition_next_state: Process(clk)
    begin
        if(rising_edge(clk)) then
            if(slowClk = '1') then
                state <= next_state;
            end if;
        end if;
 end Process transition_next_state;

--Process for entering next state
next_state_condition: Process(clk)
    begin
        if(rising_edge(clk)) then
            if(reset_flag = '1') then
                next_state <= 0;
            end if;

            if(two_seconds = 1 and state = 0) then
                next_state <= 1;
            end if;

            if(set_flag = '1') then
                next_state <= 2;
            end if;
```

```
            if(state = 2 and (confirm_flag = '1' or twenty_seconds = "010100")) then
                next_state <= 1;
            end if;
        end if;
    end Process;



  --process for state outputs
  states: Process(state)
    begin
        case state is
            when 0 => --reset state
                LED_status <= "001"; --outputting blue (RGB)

            when 1 => --clock running state
                LED_status <= "000"; --outputting black (RGB)

            when 2 =>
                LED_status <= "100"; --outputttting red (RGB)
            end case;
  end Process states;

--Concurrent Logic CODE BELOW:

  --this would be the slowclk that converts 100Mhz to 1Hz
  slowClk <= '1' when clkCnt = 100000000 - 1 else '0';

  --setting the output for the seconds counter to be displayed
  LED_seconds <= seconds;

  --setting the set/change counter
```

```
    LED_seconds_countdown <= twenty_seconds;


    --setting the state output for transfering to other entity
    state_output <= state;


end Behavioral;
```

Time Calculations:

This file is meant for setting the time on the seven segment display. It will handle all the conditions if a button is pushed for increasing the time or decreasing the time. Besides this it also increments the time normally when the clock is in the running state. So, if 60 minutes has passed while the clock is in the running state it will increment the hour on the seven segment display. Same thing for the minute where if 60 seconds passed it will increment by 1 minute on the seven segment display. This is done by having a 1 Hz clock (1 second) and then counting 60 seconds off this to trigger a 1 minute clock. The hour is done differently where once the clock hits minute 60 it will cause for the hour clock to trigger. There is no internal timer that counts 1 hour, but instead looks at when clock has reached the number 60 to trigger a 1 hour clock. There is also a debounce process in here for the buttons when increasing or decreasing while in the set/change state.

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


entity time_calc is
    Port(
        clk: in std_logic;
        state: in Integer Range 0 to 2;
        up_flag: in std_logic;
        down_flag: in std_logic;
        min_hr_flag: in std_logic_vector(1 downto 0);
        clk_display_min_1: out std_logic_vector(3 downto 0);
        clk_display_min_2: out std_logic_vector(3 downto 0);
        clk_display_hr_1: out std_logic_vector(3 downto 0);
```

```
        clk_display_hr_2: out std_logic_vector(3 downto 0)
        );
end time_calc;

architecture Behavioral of time_calc is
  --signals for the entity
    --signal for making a slow clock that is 1 Hz
    signal clkCnt: Integer := 0;
    signal slowClk: std_logic;

   --signal for min clock(needed to use for transition from 60 seconds to 1 min)
   signal min_clk: std_logic;

   --signal for hour clock
   signal hour_clk: std_logic;

   --signal for min_clk process(will be used as a counter inside the process)
   signal minCount: Integer := 0;

    --signal for minutes digit counter
    signal minute_1: Integer range 0 to 10;
    signal minute_2: Integer range 0 to 10;

    --signal for hours digit counter
    signal hour_1: Integer range 0 to 10;
    signal hour_2: Integer range 0 to 10;

   --signal for debounce counter
   signal debounce_counter: Integer := 0;
   signal debounce_clk: std_logic;

   --signals for up button and down button for minute
```

```
  signal up_minute: std_logic;
  signal down_minute: std_logic;

  --signals for up button and down button for minute
  signal up_hour: std_logic;
  signal down_hour: std_logic;

begin

 --clk scaler with 100 MHz clk (make 1 Hz clock)
 Prescaler: Process(clk)
  begin
   if rising_edge(clk) then
      if (clkCnt = 100000000 - 1) then -- if counts up to 100 million
         clkCnt <= 0;    -- when one second passes reset counter
      else
         clkCnt <= clkCnt + 1; --incrementing counter to simulate seconds
      end if;
   end if;
 end process Prescaler;

 --triggering 1 min clk cycle (making it so it trigger for a single clock rising)
 min_Prescaler: Process(clk)
  begin
   if rising_edge(clk) then
      if(slowClk = '1') then
         --reason for the minus 1 is because there's a 1 second delay otheriwse
         if (minCount = 60 - 1) then -- 60 seconds in 1 in minute
            minCount <= 0;    -- when one hour passes reset counter
         else
            minCount <= minCount + 1; --incrementing counter to simulate seconds
         end if;
```

```
    end if;
  end if;

  --reseting minute counter upon reset state because of the two second delay (will make an error otherwise)
  if(state = 0 or state = 2) then
    minCount <= 0;
  end if;
end process min_Prescaler;

  --process for debounce delay on up and down buttons
  debounce_delay: Process(clk)
    begin
      if(rising_edge(clk)) then
        if(debounce_counter = 40000000) then
          debounce_counter <= 0;
          debounce_clk <= '1';
        else
          debounce_counter <= debounce_counter + 1;
          debounce_clk <= '0';
        end if;
      end if;
  end Process debounce_delay;


  --process that counts and increments the minutes of the clock
  mins_counter: Process(clk, slowClk, min_clk, state, up_minute, down_minute)
    begin
      --when 60 seconds increment 1 min
      if(rising_edge(clk)) then
        --requiring synchronization of different clocks
        hour_clk <= '0'; --setting hour_clk to zero
        if(slowClk = '1' and min_clk = '1') then
```

```
        --when hits 60 mins increment back
        --reason for number being 59 is because how statement is written
        if(minute_2 = 5 and minute_1 = 9) then
           minute_1 <= 0;
           minute_2 <= 0;
           hour_clk <= '1';
        elsif(minute_1 = 9) then    --when hits 10 then increment second digit
                minute_1 <= 0;
                minute_2 <= minute_2 + 1;
        else
           minute_1 <= minute_1 + 1;  --increment by 1 minute
        end if;
   end if;

--condition that the minute/hour switch is set for minutes
--if statement for incrementing by 1 when increase button is pressed
   if(up_minute = '1') then
        --when hits 60 mins increment back to 0
        --reason for number being 59 is because how statement is written
      if(minute_2 = 5 and minute_1 = 9) then
                minute_1 <= 0;
                minute_2 <= 0;
                hour_clk <= '1';
      elsif(minute_1 = 9) then    --when hits 10 then increment second digit
                minute_1 <= 0;
                minute_2 <= minute_2 + 1;
      else
           minute_1 <= minute_1 + 1;  --increment by 1 minute
      end if;
   end if;

--if statement for decrementing by 1 when the decrease button is pressed
```

```
        if(down_minute = '1') then
            --when hits 0 decrements back to 59
            --reason for number being 59 is because how statement is written
            if(minute_1 = 0 and minute_2 = 0) then
                minute_2 <= 5;
                minute_1 <= 9;
            elsif(minute_1 = 0) then    --when hits 10 then increment second digit
                minute_1 <= 9;
                minute_2 <= minute_2 - 1;
            else
                minute_1 <= minute_1 - 1;  --increment by 1 minute
            end if;
        end if;

        --reseting minutes off reset input
        if(state = 0) then
            minute_1 <= 0;
            minute_2 <= 0;
        end if;
      end if;

end Process mins_counter;

--process that counters and increments the hours of the clock
 hours_counter: Process(clk, hour_clk, state, up_hour, down_hour)
   begin
      --when 60 minutes increment 1 hour
     if(rising_edge(clk)) then
     --synchronization of different clocks to make sure it triggers correctly
         if(hour_clk = '1') then
                --when reach number ten then increment second digit
                if(hour_1 = 9) then
```

```
            hour_1 <= 0;
            hour_2 <= hour_2 + 1;

        --when hits 23 hours then reset back to 0
        elsif(hour_2 = 2 and hour_1 = 3) then
            hour_1 <= 0;
            hour_2 <= 0;

        --increment normally
        else
            hour_1 <= hour_1 + 1;
        end if;
end if;

--if the min/hour switch is set for hour
--if statement for incrementing by 1 when increase button is pressed
if(up_hour = '1') then
            --when reach number ten then increment second digit
            if(hour_1 = 9) then
                hour_1 <= 0;
                hour_2 <= hour_2 + 1;

            --when hits 23 hours then reset back to 0
            elsif(hour_2 = 2 and hour_1 = 3) then
                hour_1 <= 0;
                hour_2 <= 0;

           --increment normally
            else
                hour_1 <= hour_1 + 1;
            end if;
```

```
            end if;

            --if statement for decrementing by 1 when the decrease button is pressed
            if(down_hour = '1') then

                    --when reach number 9 then decrement second digit
                    if(hour_1 = 0 and hour_2 /= 0) then
                       hour_1 <= 9;
                       hour_2 <= hour_2 - 1;

                    --when hits 0 hours then reset back to 23
                    elsif(hour_1 = 0 and hour_2 = 0) then
                       hour_2 <= 2;
                       hour_1 <= 3;

                    --decrement normally
                    else
                       hour_1 <= hour_1 - 1;
                    end if;
            end if;

            --reseting seconds off reset input
            if(state = 0) then
               hour_1 <= 0;
               hour_2 <= 0;

            end if;
         end if;
    end Process hours_counter;

--Concurrent Logic CODE:
   --this would be the slowclk that converts 100Mhz to 1Hz
```

```
slowClk <= '1' when clkCnt = 100000000 - 1 else '0';

--this wouild be the minuites that triggers off slowClk
min_clk <= '1' when minCount = 60 - 1 else '0';


--setting conditions for up/down button for mins
up_minute <= '1' when state = 2 and up_flag = '1' and min_hr_flag = "00" and debounce_clk = '1' else '0';
down_minute <= '1'when state = 2 and down_flag = '1' and min_hr_flag = "00" and debounce_clk = '1' else '0';

--setting conditions for up/down button for mins
up_hour <= '1' when state = 2 and up_flag = '1' and min_hr_flag = "01" and debounce_clk = '1' else '0';
down_hour <= '1' when state = 2 and down_flag = '1' and min_hr_flag = "01" and debounce_clk = '1' else '0';

--setting the output for the minutes counter to be displayed
clk_display_min_1 <= "0000" when minute_1 = 0 else
                     "0001" when minute_1 = 1 else
                     "0010" when minute_1 = 2 else
                     "0011" when minute_1 = 3 else
                     "0100" when minute_1 = 4 else
                     "0101" when minute_1 = 5 else
                     "0110" when minute_1 = 6 else
                     "0111" when minute_1 = 7 else
                     "1000" when minute_1 = 8 else
                     "1001" when minute_1 = 9 else
                     "1111"; --F as in error

clk_display_min_2 <= "0000" when minute_2 = 0 else
                     "0001" when minute_2 = 1 else
                     "0010" when minute_2 = 2 else
                     "0011" when minute_2 = 3 else
                     "0100" when minute_2 = 4 else
```

```
                "0101" when minute_2 = 5 else
                "0110" when minute_2 = 6 else
                "0111" when minute_2 = 7 else
                "1000" when minute_2 = 8 else
                "1001" when minute_2 = 9 else
                "1111"; --F as in error

    --setting the output for the hours counter to be displayed
    clk_display_hr_1 <= "0000" when hour_1 = 0 else
                "0001" when hour_1 = 1 else
                "0010" when hour_1 = 2 else
                "0011" when hour_1 = 3 else
                "0100" when hour_1 = 4 else
                "0101" when hour_1 = 5 else
                "0110" when hour_1 = 6 else
                "0111" when hour_1 = 7 else
                "1000" when hour_1 = 8 else
                "1001" when hour_1 = 9 else
                "1111"; --F as in error

    clk_display_hr_2 <= "0000" when hour_2 = 0 else
                "0001" when hour_2 = 1 else
                "0010" when hour_2 = 2 else
                "0011" when hour_2 = 3 else
                "0100" when hour_2 = 4 else
                "0101" when hour_2 = 5 else
                "0110" when hour_2 = 6 else
                "0111" when hour_2 = 7 else
                "1000" when hour_2 = 8 else
                "1001" when hour_2 = 9 else
                "1111"; --F as in error
end Behavioral;
```

SEVSEG Display:

This entity handles outputting to the seven segment display. It does all the calculations so that the seven segment display refreshes at the proper rate. It also makes sure that the numbers being received will output the correctly to the seven segment display. There is a state machine in this so that it properly transitions between each seven segment when outputting.

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

entity sevseg_display is
    Port(
        clk: in std_logic;
        clk_display_min_1: in std_logic_vector(3 downto 0);
        clk_display_min_2: in std_logic_vector(3 downto 0);
        clk_display_hr_1: in std_logic_vector(3 downto 0);
        clk_display_hr_2: in std_logic_vector(3 downto 0);
        SEVSEG: out std_logic_vector(7 downto 0);
        ANODE: out std_logic_vector(7 downto 0)
    );
end sevseg_display;

architecture Behavioral of sevseg_display is
    --signal for display
    signal code: std_logic_vector(3 downto 0);

    --creating a type for the states
    type eg_state_type is (s0, s1, s2, s3);

    --signals for states
    signal state_reg, state_next: eg_state_type;    --for FSM

begin
```

```
--internal clock is at 100MHZ
--wanna refresh at 60 Hz which is 0.016 sec
-- there are 4 sevseg therefore we want each on for 0.004 sec
-- 0.004 sec = 250 HZ
-- 100MHZ/ 250HZ = 400,000
-- therefore have to count up to 400,000
--for simulation change counter to 1
    timer: Process(clk)
        variable counter: integer := 0;          -- setting counter to implement a cycle
        begin
            if(rising_edge(clk)) then
                if(counter = 400000) then   --CURRENTLY SET FOR 4 SEVSEGS
                    state_reg <= state_next;
                    counter := 0;
                else
                    counter := counter + 1;
                end if;
            end if;
    end Process timer;


  --on simulation the AN will appear wrong because we have the LED refreshing right to left here and not based off numbers
  --Process for switching the 7SEG Light (AN)
  LED: Process(state_reg)
    begin
      case state_reg is
          when s0 => --for displaying first sevseg (clk min_1)
              code <= clk_display_min_1;
              state_next <= s1; -- go to next 7SEG
              ANODE <= "11111110";    --enabling first LED


          when s1 => --for displaying second sevseg (clk min_2)
```

```
            code <= clk_display_min_2;
            state_next <= s2; -- go to next 7SEG
            ANODE <= "11111101";    --enabling second LED

         when s2 => --for displaying third sevseg(clk hr_1)
            code <= clk_display_hr_1;
            state_next <= s3; -- go to next 7SEG
            ANODE <= "11111011";    --enabling third LED

         when s3 => --for displaying fourth sevseg(clk hr_2)
            code <= clk_display_hr_2;
            state_next <= s0; -- go back to first 7SEG
            ANODE <= "11110111";    --enabling fourth LED
      end case;
   end Process LED;

   --SEVSEG = | CA | CB | CC | CD | CE | CF | CG | DP
   SEVSEG <= "00000011" when code <= "0000" else   -- display 0
         "10011111" when code <= "0001" else   -- display 1
         "00100101" when code <= "0010" else   -- display 2
         "00001101" when code <= "0011" else   -- display 3
         "10011001" when code <= "0100" else   -- display 4
         "01001001" when code <= "0101" else   -- display 5
         "01000001" when code <= "0110" else   -- display 6
         "00011111" when code <= "0111" else   -- display 7
         "00000001" when code <= "1000" else   -- display 8
         "00001001" when code <= "1001" else   -- display 9
         "01110001" when code <= "1111" else   -- display F
         "11111111";

end Behavioral;
```

4) Based on your design, what will be a good test bench for simulating the operation of your system? Please describe it and insert below the source code from your VHDL testbench file

This testbench just confirms the basic operation of the design. I just wanted to test if the proper states are entered. The timing is not going to be correct because we can't really simulate a 100 MHz clock and work in seconds because that simulation would take hours to run. Because of this all the time clock processes were set to 1 with some slight changes for simulation purposes. Because of these slight changes I've also posted their code below.

TESTBENCH

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

entity Clock_TB is
end Clock_TB;

architecture Behavioral of Clock_TB is
    --signals for TESTBENCH simulation
    signal clk: std_logic;
    signal reset_button: std_logic;
    signal up_button: std_logic;
    signal down_button: std_logic;
    signal confirm_button: std_logic;
    signal set_button: std_logic;
    signal min_hr_switch: std_logic_vector(1 downto 0);
    signal SEVSEG: std_logic_vector(7 downto 0);
    signal ANODE: std_logic_vector(7 downto 0);
    signal LED_status: std_logic_vector(2 downto 0);
    signal LED_seconds: std_logic_vector(5 downto 0);
    signal LED_seconds_countdown: std_logic_vector(5 downto 0);

    Component clock_structure_TB
```

```
        Port(
            clk: in std_logic;
            reset_button: in std_logic;
            up_button: in std_logic;
            down_button: in std_logic;
            confirm_button: in std_logic;
            set_button: in std_logic;
            min_hr_switch: in std_logic_vector(1 downto 0);
            SEVSEG: out std_logic_vector(7 downto 0);
            ANODE: out std_logic_vector(7 downto 0);
            --led status (RGB)
            LED_status: out std_logic_vector(2 downto 0);
          --second display for LEDS
            LED_seconds: out std_logic_vector(5 downto 0);
            LED_seconds_countdown: out std_logic_vector(5 downto 0)
        );
    end Component;

begin
    --wiring TB signals to their I/O's
    DUT: clock_structure_TB Port Map
        (
            clk => clk,
            reset_button => reset_button,
            up_button => up_button,
            down_button => down_button,
            confirm_button => confirm_button,
            set_button => set_button,
            min_hr_switch => min_hr_switch,
            SEVSEG => SEVSEG,
            ANODE => ANODE,
            LED_status => LED_status,
```

```
        LED_seconds => LED_seconds,
        LED_seconds_countdown => LED_seconds_countdown
    );


--setting up clock for simulation
clock: Process
    begin
        clk <= '1';
        wait for 5 ns;
        clk <= '0';
        wait for 5 ns;
end Process;


--process for running through test signals
Stimulus: Process
    begin
        reset_button <= '0';
        up_button <= '0';
        down_button <= '0';
        confirm_button <= '0';
        set_button <= '0';
        min_hr_switch <= "00";
        wait for 60 ns;
        set_button <= '1';
        wait for 10 ns;
        set_button <= '0';
        wait for 50 ns;
        Up_button <= '1';
        wait for 10 ns;
        Up_button <= '0';
```

```
        wait for 100 ns;
        down_button <= '1';
        wait for 20 ns;
        down_button <= '0';
        wait for 600ns;
        reset_button <= '1';
        wait for 10 ns;
        reset_button <= '0';
        wait for 5000ns;
    end Process;
end Behavioral;
```

FLAGS:

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

entity flags_TB is
    Port(
        reset_button: in std_logic;
        up_button: in std_logic;
        down_button: in std_logic;
        confirm_button: in std_logic;
        set_button: in std_logic;
        min_hr_switch: in std_logic_vector(1 downto 0);
        reset_flag: out std_logic;
        up_flag: out std_logic;
        down_flag: out std_logic;
        set_flag: out std_logic;
        confirm_flag: out std_logic;
        min_hr_flag: out std_logic_vector(1 downto 0)
    );
end flags_TB;
```

```
architecture Behavioral of flags_TB is

begin

    --setting the flags based off inputs
    reset_flag <= '1' when reset_button = '1' else '0';
    up_flag <= '1' when up_button = '1' else '0';
    down_flag <= '1' when down_button = '1' else '0';
    confirm_flag <= '1' when confirm_button = '1' else '0';
    set_flag <= '1' when set_button = '1' else '0';

    min_hr_flag <= "00" when min_hr_switch = "00" else
                   "01" when min_hr_switch = "01" else
                   "10" when min_hr_switch = "10" else
                   "11";
end Behavioral;
```

FSM:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;



entity state_machine_clock_TB is
    Port(
        --clk input for hardware
        clk: in std_logic;
        --flag inputs
        reset_flag: in std_logic;
        set_flag: in std_logic;
        confirm_flag: in std_logic;
      --led status (RGB)
        LED_status: out std_logic_vector(2 downto 0);
      --second display for LEDS
```

```
        LED_seconds: out std_logic_vector(5 downto 0);
        LED_seconds_countdown: out std_logic_vector(5 downto 0);
        state_output: out Integer Range 0 to 2
    );
end state_machine_clock_TB;



architecture Behavioral of state_machine_clock_TB is

    --signal  for states
    signal state: Integer range 0 to 2;
    signal next_state: Integer range 0 to 2;

    --signal for making a slow clock that is 1 Hz
    signal clkCnt: Integer := 0;
    signal slowClk: std_logic;

    --signal for seconds counter
    signal seconds: std_logic_vector(5 downto 0);

    --signal for two second delay in reset state
    signal two_seconds: Integer := 0;

    --signal for 20 seconds delay in set/change state
    signal twenty_seconds: std_logic_vector(5 downto 0);

begin

  --clk scaler with 100 MHz clk (make 1 Hz clock)
  second_Prescaler: Process(clk)
   begin
    if rising_edge(clk) then
        if (clkCnt = 1) then -- if counts up to 100 million
            clkCnt <= 0;    -- when one second passes reset counter
        else
            clkCnt <= clkCnt + 1; --incrementing counter to simulate seconds
```

```
    end if;
  end if;
end process second_Prescaler;


 --process that counts the seconds of each minute
 seconds_counter: Process(clk, slowClk, state)
    begin
        if(rising_edge(clk)) then
            if(slowClk = '1') then
                --reason for being 59 is because of how statement is written
                if(seconds ="111011") then    --reset back to 0 when 60 seconds pass
                    seconds <= "000000";
                else
                    seconds <= seconds + "000001"; --increment seconds counter
                end if;
            end if;
        end if;

        --reseting seconds off reset input
        if(state = 0 or state = 2) then
            seconds <= "000000";
        end if;
 end Process seconds_counter;

 --process that counts 2 seconds
 two_second_delay: Process(clk, slowClk, state)
    begin
        if(rising_edge(clk)) then
            if(slowClk = '1' and state = 0) then
                --incrementing by 1 second
                two_seconds <= two_seconds + 1;
            end if;
        end if;

        --reseting seconds after done with reset state
```

```
        if(state = 1 or state = 2) then
            two_seconds <= 0;
        end if;
 end Process two_second_delay;

--process that counts to 20 seconds
twenty_second_delay: Process(clk, slowClk, state)
    begin
        if(rising_edge(clk)) then
            if(slowClk = '1' and state = 2) then
                --incrementing by 1 second
                twenty_seconds <= twenty_seconds + "000001";
            end if;
        end if;

        --reseting seconds after done with reset state
        if(state = 0 or state = 1) then
            twenty_seconds <= "000000";
        end if;
 end Process twenty_second_delay;

--Process that will tranistion the state to the nextstate
--using slowClk to act as a slight debounce
transition_next_state: Process(clk)
    begin
        if(rising_edge(clk)) then
            if(slowClk = '1') then
                state <= next_state;
            end if;
        end if;
 end Process transition_next_state;

--Process for entering next state
next_state_condition: Process(clk)
    begin
        if(rising_edge(clk)) then
```

```
            if(reset_flag = '1') then
                next_state <= 0;
            end if;

            if(two_seconds = 1 and state = 0) then
                next_state <= 1;
            end if;

            if(set_flag = '1') then
                next_state <= 2;
            end if;

            if(state = 2 and (confirm_flag = '1' or twenty_seconds = "010100")) then
                next_state <= 1;
            end if;
        end if;
    end Process;

  --process for state outputs
  states: Process(state)
    begin
        case state is
            when 0 => --reset state
                LED_status <= "001"; --outputting blue (RGB)

            when 1 => --clock running state
                LED_status <= "000"; --outputting black (RGB)

            when 2 =>
                LED_status <= "100"; --outputttting red (RGB)
            end case;
  end Process states;


--Concurrent Logic CODE BELOW:
```

```
    --this would be the slowclk that converts 100Mhz to 1Hz
    slowClk <= '1' when clkCnt = 1 else '0';

    --setting the output for the seconds counter to be displayed
    LED_seconds <= seconds;

    --setting the set/change counter
    LED_seconds_countdown <= twenty_seconds;

    --setting the state output for transfering to other entity
    state_output <= state;

end Behavioral;
```

Time Calc:

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

entity time_TB is
    Port(
        clk: in std_logic;
        state: in Integer Range 0 to 2;
        up_flag: in std_logic;
        down_flag: in std_logic;
        min_hr_flag: in std_logic_vector(1 downto 0);
        clk_display_min_1: out std_logic_vector(3 downto 0);
        clk_display_min_2: out std_logic_vector(3 downto 0);
        clk_display_hr_1: out std_logic_vector(3 downto 0);
        clk_display_hr_2: out std_logic_vector(3 downto 0)
        );
end time_TB;

architecture Behavioral of time_TB is
  --signals for the entity
```

```
    --signal for making a slow clock that is 1 Hz
    signal clkCnt: Integer := 0;
    signal slowClk: std_logic;

   --signal for min clock(needed to use for transition from 60 seconds to 1 min)
   signal min_clk: std_logic;

   --signal for hour clock
   signal hour_clk: std_logic;

   --signal for min_clk process(will be used as a counter inside the process)
   signal minCount: Integer := 0;

    --signal for minutes digit counter
    signal minute_1: Integer range 0 to 10;
    signal minute_2: Integer range 0 to 10;

    --signal for hours digit counter
    signal hour_1: Integer range 0 to 10;
    signal hour_2: Integer range 0 to 10;

   --signal for debounce counter
   signal debounce_counter: Integer := 0;
   signal debounce_clk: std_logic;

   --signals for up button and down button for minute
   signal up_minute: std_logic;
   signal down_minute: std_logic;

   --signals for up button and down button for minute
   signal up_hour: std_logic;
   signal down_hour: std_logic;

begin

  --clk scaler with 100 MHz clk (make 1 Hz clock)
```

```
Prescaler: Process(clk)
 begin
  if rising_edge(clk) then
     if (clkCnt = 1) then -- if counts up to 100 million
        clkCnt <= 0;    -- when one second passes reset counter
     else
        clkCnt <= clkCnt + 1; --incrementing counter to simulate seconds
     end if;
  end if;
end process Prescaler;

--triggering 1 min clk cycle (making it so it trigger for a single clock rising)
min_Prescaler: Process(clk)
 begin
  if rising_edge(clk) then
     if(slowClk = '1') then
        --reason for the minus 1 is because there's a 1 second delay otheriwse
        if (minCount = 60 - 1) then -- 60 seconds in 1 in minute
           minCount <= 0;    -- when one hour passes reset counter
        else
           minCount <= minCount + 1; --incrementing counter to simulate seconds
        end if;
     end if;
  end if;

  --reseting minute counter upon reset state because of the two second delay (will make an error otherwise)
  if(state = 0 or state = 2) then
     minCount <= 0;
  end if;
end process min_Prescaler;

  --process for debounce delay on up and down buttons
  debounce_delay: Process(clk)
     begin
        if(rising_edge(clk)) then
           if(debounce_counter = 1) then
```

```
            debounce_counter <= 0;
            debounce_clk <= '1';
        else
            debounce_counter <= debounce_counter + 1;
            debounce_clk <= '0';
        end if;
    end if;
end Process debounce_delay;



--process that counts and increments the minutes of the clock
mins_counter: Process(clk, slowClk, min_clk, state, up_minute, down_minute)
  begin
    --when 60 seconds increment 1 min
    if(rising_edge(clk)) then
      --requiring synchronization of different clocks
      hour_clk <= '0'; --setting hour_clk to zero
      if(slowClk = '1' and min_clk = '1') then
          --when hits 60 mins increment back
          --reason for number being 59 is because how statement is written
          if(minute_2 = 5 and minute_1 = 9) then
            minute_1 <= 0;
            minute_2 <= 0;
            hour_clk <= '1';
          elsif(minute_1 = 9) then    --when hits 10 then increment second digit
                minute_1 <= 0;
                minute_2 <= minute_2 + 1;
          else
            minute_1 <= minute_1 + 1;  --increment by 1 minute
          end if;
      end if;

    --condition that the minute/hour switch is set for minutes
    --if statement for incrementing by 1 when increase button is pressed
      if(up_minute = '1') then
          --when hits 60 mins increment back to 0
```

```
            --reason for number being 59 is because how statement is written
        if(minute_2 = 5 and minute_1 = 9) then
                minute_1 <= 0;
                minute_2 <= 0;
                hour_clk <= '1';
        elsif(minute_1 = 9) then    --when hits 10 then increment second digit
                minute_1 <= 0;
                minute_2 <= minute_2 + 1;
        else
            minute_1 <= minute_1 + 1;  --increment by 1 minute
        end if;
      end if;

    --if statement for decrementing by 1 when the decrease button is pressed
      if(down_minute = '1') then
            --when hits 0 decrements back to 59
            --reason for number being 59 is because how statement is written
        if(minute_1 = 0 and minute_2 = 0) then
                minute_2 <= 5;
                minute_1 <= 9;
        elsif(minute_1 = 0) then    --when hits 10 then increment second digit
                minute_1 <= 9;
                minute_2 <= minute_2 - 1;
        else
            minute_1 <= minute_1 - 1;  --increment by 1 minute
        end if;
      end if;

      --reseting minutes off reset input
      if(state = 0) then
        minute_1 <= 0;
        minute_2 <= 0;
      end if;
    end if;

end Process mins_counter;
```

```
--process that counters and increments the hours of the clock
 hours_counter: Process(clk, hour_clk, state, up_hour, down_hour)
   begin
      --when 60 minutes increment 1 hour
     if(rising_edge(clk)) then
     --synchronization of different clocks to make sure it triggers correctly
         if(hour_clk = '1') then
                  --when reach number ten then increment second digit
                  if(hour_1 = 9) then
                     hour_1 <= 0;
                     hour_2 <= hour_2 + 1;

                  --when hits 24 hours then reset back to 0
                  elsif(hour_2 = 2 and hour_1 = 3) then
                     hour_1 <= 0;
                     hour_2 <= 0;

                  --increment normally
                  else
                     hour_1 <= hour_1 + 1;
                  end if;
         end if;

         --if the min/hour switch is set for hour
         --if statement for incrementing by 1 when increase button is pressed
         if(up_hour = '1') then
                  --when reach number ten then increment second digit
                  if(hour_1 = 9) then
                     hour_1 <= 0;
                     hour_2 <= hour_2 + 1;

                  --when hits 24 hours then reset back to 0
                  elsif(hour_2 = 2 and hour_1 = 3) then
                     hour_1 <= 0;
                     hour_2 <= 0;
```

```
                --increment normally
                else
                    hour_1 <= hour_1 + 1;
                end if;

        end if;

        --if statement for decrementing by 1 when the decrease button is pressed
        if(down_hour = '1') then

                --when reach number 9 then decrement second digit
                if(hour_1 = 0 and hour_2 /= 0) then
                    hour_1 <= 9;
                    hour_2 <= hour_2 - 1;

                --when hits 0 hours then reset back to 23
                elsif(hour_1 = 0 and hour_2 = 0) then
                    hour_2 <= 2;
                    hour_1 <= 3;

                --decrement normally
                else
                    hour_1 <= hour_1 - 1;
                end if;
        end if;

        --reseting seconds off reset input
        if(state = 0) then
            hour_1 <= 0;
            hour_2 <= 0;

        end if;
    end if;
end Process hours_counter;
```

```
--Concurrent Logic CODE:
  --this would be the slowclk that converts 100Mhz to 1Hz
  slowClk <= '1' when clkCnt = 1 else '0';

  --this wouild be the minuites that triggers off slowClk
  min_clk <= '1' when minCount = 60 - 1 else '0';


  --setting conditions for up/down button for mins
  up_minute <= '1' when state = 2 and up_flag = '1' and min_hr_flag = "00" and debounce_clk = '1' else '0';
  down_minute <= '1'when state = 2 and down_flag = '1' and min_hr_flag = "00" and debounce_clk = '1' else '0';

  --setting conditions for up/down button for mins
  up_hour <= '1' when state = 2 and up_flag = '1' and min_hr_flag = "01" and debounce_clk = '1' else '0';
  down_hour <= '1' when state = 2 and down_flag = '1' and min_hr_flag = "01" and debounce_clk = '1' else '0';


  --setting the output for the minutes counter to be displayed
  clk_display_min_1 <= "0000" when minute_1 = 0 else
                       "0001" when minute_1 = 1 else
                       "0010" when minute_1 = 2 else
                       "0011" when minute_1 = 3 else
                       "0100" when minute_1 = 4 else
                       "0101" when minute_1 = 5 else
                       "0110" when minute_1 = 6 else
                       "0111" when minute_1 = 7 else
                       "1000" when minute_1 = 8 else
                       "1001" when minute_1 = 9 else
                       "1111"; --F as in error

  clk_display_min_2 <= "0000" when minute_2 = 0 else
                       "0001" when minute_2 = 1 else
                       "0010" when minute_2 = 2 else
                       "0011" when minute_2 = 3 else
                       "0100" when minute_2 = 4 else
                       "0101" when minute_2 = 5 else
```

```
                "0110" when minute_2 = 6 else
                "0111" when minute_2 = 7 else
                "1000" when minute_2 = 8 else
                "1001" when minute_2 = 9 else
                "1111"; --F as in error

    --setting the output for the hours counter to be displayed
    clk_display_hr_1 <= "0000" when hour_1 = 0 else
                "0001" when hour_1 = 1 else
                "0010" when hour_1 = 2 else
                "0011" when hour_1 = 3 else
                "0100" when hour_1 = 4 else
                "0101" when hour_1 = 5 else
                "0110" when hour_1 = 6 else
                "0111" when hour_1 = 7 else
                "1000" when hour_1 = 8 else
                "1001" when hour_1 = 9 else
                "1111"; --F as in error

    clk_display_hr_2 <= "0000" when hour_2 = 0 else
                "0001" when hour_2 = 1 else
                "0010" when hour_2 = 2 else
                "0011" when hour_2 = 3 else
                "0100" when hour_2 = 4 else
                "0101" when hour_2 = 5 else
                "0110" when hour_2 = 6 else
                "0111" when hour_2 = 7 else
                "1000" when hour_2 = 8 else
                "1001" when hour_2 = 9 else
                "1111"; --F as in error
end Behavioral;
```

SEVSEG:

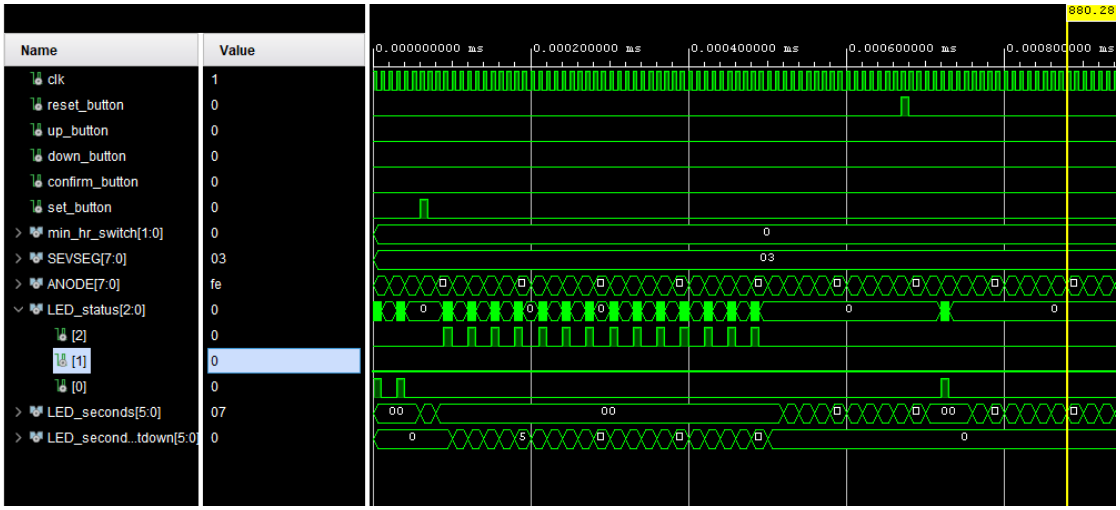
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity sevseg_TB is
    Port(
        clk: in std_logic;
        clk_display_min_1: in std_logic_vector(3 downto 0);
        clk_display_min_2: in std_logic_vector(3 downto 0);
        clk_display_hr_1: in std_logic_vector(3 downto 0);
        clk_display_hr_2: in std_logic_vector(3 downto 0);
        SEVSEG: out std_logic_vector(7 downto 0);
        ANODE: out std_logic_vector(7 downto 0)
    );
end sevseg_TB;

architecture Behavioral of sevseg_TB is
    --signal for display
    signal code: std_logic_vector(3 downto 0);

    --creating a type for the states
    type eg_state_type is (s0, s1, s2, s3);

    --signals for states
    signal state_reg, state_next: eg_state_type;    --for FSM

begin
    --internal clock is at 100MHZ
    --wanna refresh at 60 Hz which is 0.016 sec
    -- there are 4 sevseg therefore we want each on for 0.004 sec
    -- 0.004 sec = 250 HZ
    -- 100MHZ/250 HZ = 400,000
    -- therefore have to count up to 400,000
    --for simulation change counter to 1
        timer: Process(clk)
            variable counter: integer := 0;          -- setting counter to implement a cycle
```

```
        begin
            if(rising_edge(clk)) then
                if(counter = 1) then   --CURRENTLY SET FOR 4 SEVSEGS
                    state_reg <= state_next;
                    counter := 0;
                else
                    counter := counter + 1;
                end if;
            end if;
    end Process timer;


--on simulation the AN will appear wrong because we have the LED refreshing right to left here and not basesd off numbers
--Process for switching the 7SEG Light (AN)
LED: Process(state_reg)
    begin
        case state_reg is
            when s0 => --for displaying first sevseg (clk min_1)
                code <= clk_display_min_1;
                state_next <= s1; -- go to next 7SEG
                ANODE <= "11111110";    --enabling first LED

            when s1 => --for displaying second sevseg (clk min_2)
                code <= clk_display_min_2;
                state_next <= s2; -- go to next 7SEG
                ANODE <= "11111101";    --enabling second LED

            when s2 => --for displaying third sevseg(clk hr_1)
                code <= clk_display_hr_1;
                state_next <= s3; -- go to next 7SEG
                ANODE <= "11111011";    --enabling third LED

            when s3 => --for displaying fourth sevseg(clk hr_2)
                code <= clk_display_hr_2;
                state_next <= s0; -- go back to first 7SEG
                ANODE <= "11110111";    --enabling fourth LED
        end case;
```

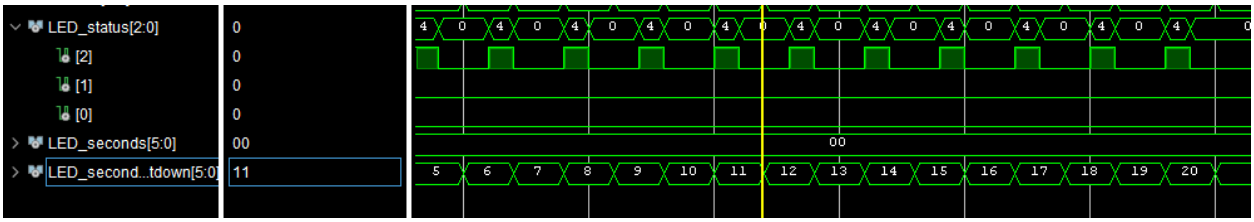
```
    end Process LED;


    --SEVSEG = | CA | CB | CC | CD | CE | CF | CG | DP
    SEVSEG <= "00000011" when code <= "0000" else   -- display 0
          "10011111" when code <= "0001" else   -- display 1
          "00100101" when code <= "0010" else   -- display 2
          "00001101" when code <= "0011" else   -- display 3
          "10011001" when code <= "0100" else   -- display 4
          "01001001" when code <= "0101" else   -- display 5
          "01000001" when code <= "0110" else   -- display 6
          "00011111" when code <= "0111" else   -- display 7
          "00000001" when code <= "1000" else   -- display 8
          "00001001" when code <= "1001" else   -- display 9
          "11111111";
end Behavioral;
```

5) What is the result waveform from your simulation? Based on this simulation, is your system implementation functioning as designed?

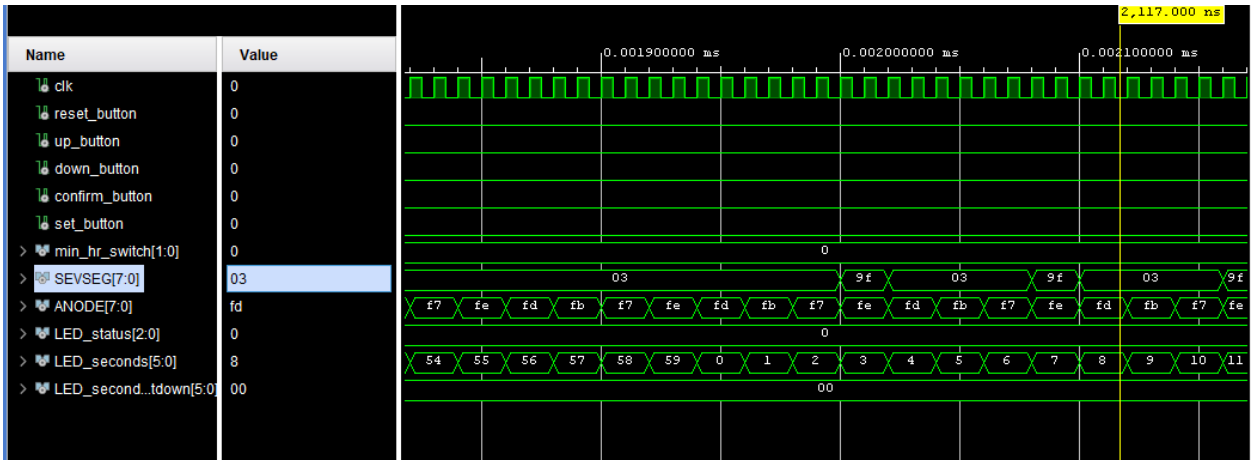
Here this waveform just shows the different states. The “LED status” shows how it enters the different states by changing the binary number for it (RGB). At first it enters the reset state and “LED status” is outputting “001” (Blue). When the “set button” is triggered it goes from “000” to outputting “100” (RED). After 20 seconds it will enter the running clock state and LED status will be “000” (black). I then trigger the reset button again which causes it to output “001” again. It then enters the normal clock state which causes the LED status to be “000”. I also have LED status not acting at 100% duty cycle to show that it can be implemented correctly. This is not10% duty cycle, but the numbers can be changed around for it too act properly when implemented onto the board. I just wanted to see if it functions.



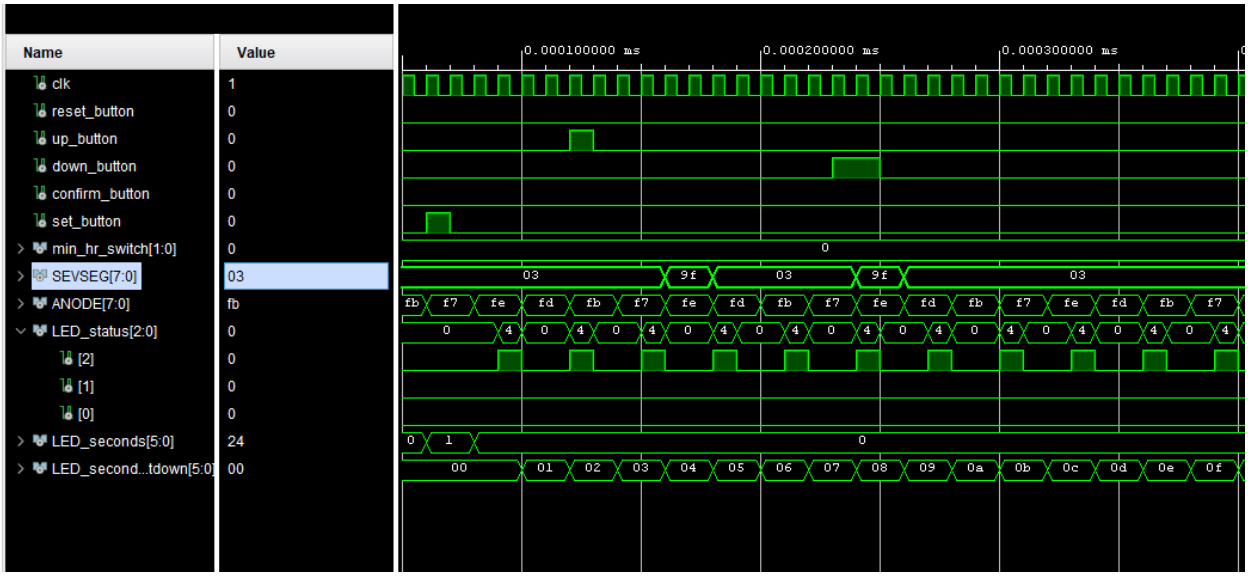
This waveform shows that the countdown works properly in set/change state where it will go up to twenty seconds and then exit to the running clock state.



This waveform shows 60 seconds passing. The SEVSEG goes from outputting constant x03 (‘0’ on the seven segment display) to x9f and x03. “x9f” is 1 on seven segment so it makes sense for when 60 seconds pass that the minute on the seven segment increases by 1.



This last waveform shows the UP and down button working when in the set/change state. Here we press the Up button and see that SEVSEG goes from outputting constant x03 to x9F and x03. This makes sense because the switch for changing the minute/hour is set for minute by being “00”. So when we increment once, one of the SEVSEG’s should start outputting 1 instead of constant zero which does occur with the “x9F”. Then when the down button is triggered it should go back to outputting a constant “x03” which is zero. This does occur showing it works.



After the QuestaSim simulation, please implement your digital clock on the Nexys 4 DDR board, and use the input/output setup as given above.

6) In this case, what is the constraint XDC file for this project?

```
## Clock signal

set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk }];
#IO_L12P_T1_MRCC_35 Sch=clk100mhz

create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {clk}];

set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets reset_button_IBUF]


## LEDs

set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { LED_seconds[0] }];
#IO_L18P_T2_A24_15 Sch=led[0]

set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports { LED_seconds[1] }];
#IO_L24P_T3_RS1_15 Sch=led[1]

set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 } [get_ports { LED_seconds[2] }];
#IO_L17N_T2_A25_15 Sch=led[2]

set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 } [get_ports { LED_seconds[3] }];
#IO_L8P_T1_D11_14 Sch=led[3]

set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { LED_seconds[4] }];
#IO_L7P_T1_D09_14 Sch=led[4]

set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { LED_seconds[5] }];
#IO_L18N_T2_A11_D27_14 Sch=led[5]

set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports {
LED_seconds_countdown[0] }]; #IO_L22P_T3_A05_D21_14 Sch=led[10]

set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports {
LED_seconds_countdown[1] }]; #IO_L15N_T2_DQS_DOUT_CSO_B_14 Sch=led[11]

set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports {
LED_seconds_countdown[2] }]; #IO_L16P_T2_CSI_B_14 Sch=led[12]

set_property -dict { PACKAGE_PIN V14   IOSTANDARD LVCMOS33 } [get_ports {
LED_seconds_countdown[3] }]; #IO_L22N_T3_A04_D20_14 Sch=led[13]

set_property -dict { PACKAGE_PIN V12   IOSTANDARD LVCMOS33 } [get_ports {
LED_seconds_countdown[4] }]; #IO_L20N_T3_A07_D23_14 Sch=led[14]

set_property -dict { PACKAGE_PIN V11   IOSTANDARD LVCMOS33 } [get_ports {
LED_seconds_countdown[5] }]; #IO_L21N_T3_DQS_A06_D22_14 Sch=led[15]
```

##Switches

set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { min_hr_switch[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]

set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 } [get_ports { min_hr_switch[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]

RGB LEDs

set_property -dict { PACKAGE_PIN R12 IOSTANDARD LVCMOS33 } [get_ports { LED_status[0] }]; #IO_L5P_T0_D06_14 Sch=led16_b

set_property -dict { PACKAGE_PIN M16 IOSTANDARD LVCMOS33 } [get_ports { LED_status[1] }]; #IO_L10P_T1_D14_14 Sch=led16_g

set_property -dict { PACKAGE_PIN N15 IOSTANDARD LVCMOS33 } [get_ports { LED_status[2] }]; #IO_L11P_T1_SRCC_14 Sch=led16_r

##7 segment display

set_property -dict { PACKAGE_PIN T10 IOSTANDARD LVCMOS33 } [get_ports { SEVSEG[7] }]; #IO_L24N_T3_A00_D16_14 Sch=ca

set_property -dict { PACKAGE_PIN R10 IOSTANDARD LVCMOS33 } [get_ports { SEVSEG[6] }]; #IO_25_14 Sch=cb

set_property -dict { PACKAGE_PIN K16 IOSTANDARD LVCMOS33 } [get_ports { SEVSEG[5] }]; #IO_25_15 Sch=cc

set_property -dict { PACKAGE_PIN K13 IOSTANDARD LVCMOS33 } [get_ports { SEVSEG[4] }]; #IO_L17P_T2_A26_15 Sch=cd

set_property -dict { PACKAGE_PIN P15 IOSTANDARD LVCMOS33 } [get_ports { SEVSEG[3] }]; #IO_L13P_T2_MRCC_14 Sch=ce

set_property -dict { PACKAGE_PIN T11 IOSTANDARD LVCMOS33 } [get_ports { SEVSEG[2] }]; #IO_L19P_T3_A10_D26_14 Sch=cf

set_property -dict { PACKAGE_PIN L18 IOSTANDARD LVCMOS33 } [get_ports { SEVSEG[1] }]; #IO_L4P_T0_D04_14 Sch=cg

set_property -dict { PACKAGE_PIN H15 IOSTANDARD LVCMOS33 } [get_ports { SEVSEG[0] }]; #IO_L19N_T3_A21_VREF_15 Sch=dp

set_property -dict { PACKAGE_PIN J17 IOSTANDARD LVCMOS33 } [get_ports { ANODE[0] }]; #IO_L23P_T3_FOE_B_15 Sch=an[0]

set_property -dict { PACKAGE_PIN J18 IOSTANDARD LVCMOS33 } [get_ports { ANODE[1] }]; #IO_L23N_T3_FWE_B_15 Sch=an[1]

set_property -dict { PACKAGE_PIN T9 IOSTANDARD LVCMOS33 } [get_ports { ANODE[2] }]; #IO_L24P_T3_A01_D17_14 Sch=an[2]

```
set_property -dict { PACKAGE_PIN J14   IOSTANDARD LVCMOS33 } [get_ports { ANODE[3] }];
#IO_L19P_T3_A22_15 Sch=an[3]

set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 } [get_ports { ANODE[4] }];
#IO_L8N_T1_D12_14 Sch=an[4]

set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS33 } [get_ports { ANODE[5] }];
#IO_L14P_T2_SRCC_14 Sch=an[5]

set_property -dict { PACKAGE_PIN K2    IOSTANDARD LVCMOS33 } [get_ports { ANODE[6] }];
#IO_L23P_T3_35 Sch=an[6]

set_property -dict { PACKAGE_PIN U13   IOSTANDARD LVCMOS33 } [get_ports { ANODE[7] }];
#IO_L23N_T3_A02_D18_14 Sch=an[7]


##Buttons

set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 } [get_ports { reset_button }];
#IO_L9P_T1_DQS_14 Sch=btnc

set_property -dict { PACKAGE_PIN M18   IOSTANDARD LVCMOS33 } [get_ports { up_button }];
#IO_L4N_T0_D05_14 Sch=btnu

set_property -dict { PACKAGE_PIN P17   IOSTANDARD LVCMOS33 } [get_ports { set_button }];
#IO_L12P_T1_MRCC_14 Sch=btnl

set_property -dict { PACKAGE_PIN M17   IOSTANDARD LVCMOS33 } [get_ports { confirm_button }];
#IO_L10N_T1_D15_14 Sch=btnr

set_property -dict { PACKAGE_PIN P18   IOSTANDARD LVCMOS33 } [get_ports { down_button }];
#IO_L9N_T1_DQS_D13_14 Sch=btnd
```

7) After implementing your digital design on the Nexys 4 board, please demonstrate it to the instructor; and create a short video demonstrating your test results. Please upload the short video as an attachment to this report in Titanium.

https://drive.google.com/file/d/1u6KpkYhk_dK3dioy82gu-9rTgS3nn9MW/view?usp=sharing