**ELEC 204 Digital Design Project Report**

**Medication Reminder**

Enes Kaya 69095

Deniz Yılmaz 69743

Date: 01/10/2021

1. **Introduction and objectives**

The objective of our project is to implement an FPGA assistant which reminds patients to take medications and counts the number of medications taken. The medication reminder offers a convenient way to set an alarm. The user gives only the information of a time interval which a medication needs to be taken. The medication reminder saves this data to keep reminding at the end of each time interval so that the user does not need to enter the time interval again and again. Additionally, the counter increments at the end of each time interval, which reminds the patient the number of medications taken to avoid overuse of a medication. We used the seven-segment display, LED’s, switches and buttons to be able to implement these functions.

1. **Methods**

**CLOCK\_GENERATOR.vhd**

In this code, we generate 2 clocks for timing and seven-segment part.

|  |  |
| --- | --- |
| **Inputs** | **Description** |
| MCLK | the master clock. |

|  |  |
| --- | --- |
| **Outputs** | **Description** |
| SECONDCLK | the clock for the timer, so its period is 1 second. |
| HUNDREDHZCLOCK | the clock for the seven-segment display. |

The ‘COUNTER’ increases by 1 for each rising edge of MCLK. When it is smaller than 50000, ‘temp’ is 1. When it reaches 50000, ‘temp’ is 0 and ‘COUNTER’ again becomes 0. This means that we divide our master clock into 50000. For example, if our master clock 100 MHz=100\*10^6 Hz, we obtain 2000 ((100\*10^6) / 50000) and our new clock has 2000 rising edges in 1 second so our new clock’s frequency is 2000 Hz.

The ‘COUNTERS’ increases by 1 for each rising edge of MCLK. When it is smaller than 100000000, ‘temps’ is 1. When it reaches 100000000, ‘temps’ is 0 and ‘COUNTERS’ again becomes 0. This means that we divide our master clock into 100000000. It corresponds to 1 second between each rising edge of temps, if our master clock 100 MHz=100\*10^6 Hz.

At the end, we assign temps to secondclk and assign temp to hundredhzclock.

**SEVSEG\_DRIVER.vhd**

In this module, we determined which digit of seven-segment display shows which value, so we assigned the values to the digits.

|  |  |
| --- | --- |
| **Inputs** | **Description** |
| CLK | 1-bit clock input which indicates our clock for the operations in our seven segment display codes. It actually equals to ‘HUNDREDHZCLOCK’. We generated this clock in the clock generator module. |
| D1 | 4-bit input which indicates the data displayed on the leftmost seven-segment display. This data shows tens place of minute value. |
| D2 | 4-bit input which shows ones place of minute value. |
| D3 | 4-bit input which shows tens place of second value. |
| D4 | 4-bit input which shows ones place of second value. |
| D5 | 4-bit input which indicates the data displayed on seven-segment display before the rightmost one. This data shows tens place of medication counter |
| D6 | 4-bit input which indicates the data displayed on the rightmost seven-segment display. This data shows ones place of medication counter. |

On the seven-segment display, the order is: D1 D2 D3 D4 - - D5 D6

|  |  |
| --- | --- |
| **Outputs** | **Description** |
| SEV\_SEG\_DRIVER | It has 8 bits because there are 8 seven-segment display, but we used only 6 of them. When one of the bits is 0 and the others is 1, this means that this seven-segment is used on the FPGA board. For example, when SEV\_SEG\_DRIVER is 01111111, the leftmost seven-segment display shows the value which we assigned. |
| SEV\_SEG\_DATA | It contains 4 bits because our data (D1, D2, D3, D4, D5, D6) also have 4 bits. SEV\_SEG\_DATA indicates the value which is showed on the seven-segment display. |

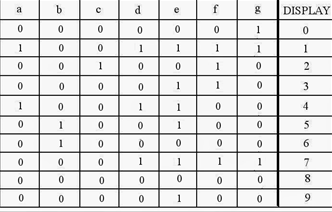
**SEVSEG\_DECODER.vhd**

In this module, we determine which segment will be light in order to show the desired digit on the display. There are 8 seven-segments and each seven-segment contains 7 segments (led). When we want to show a digit on a seven-segment, we should make the some segments be light in order to display that digit.

|  |  |
| --- | --- |
| **Inputs** | **Description** |
| INPUT | It equals SEV\_SEG\_DATA from SEVSEG\_DRIVER.vhd because we link this data to one of the 8 seven-segments. |

|  |  |
| --- | --- |
| **Outputs** | **Description** |
| SEVSEG\_BUS | It has 7 bits to indicate each 7 segments. When the bit is 0, this segment will be light. We use the truth table to be able to display each digit. |

Truth table:

** [3]**

**Timer.vhd**

The main operations are performed in this module.

|  |  |
| --- | --- |
| **Inputs** | **Description** |
| SCLK | 1-bit clock input for 1 second. |
| set\_mode | 1-bit switch input. When it is turned on, the time interval for medications can be entered. |
| start | 1-bit switch input. When it is turned on, countdown starts. When countdown is over and the alarm LED is turned on, it is turned back off to indicate that the medication is taken. |
| SET\_SELECT | 2-bit switch input. While setting an alarm, the digit to be incremented is specified with SET\_SELECT switches. |
| RST | 1-bit button input to reset the timer. It also returns the medication counter to the initial value 1. |
| PLUSONE | 1-bit button input which is used to increment the selected digit on the seven-segment display. |

|  |  |
| --- | --- |
| **Outputs** | **Description** |
| led\_digit | 4-bit LED output which indicates the current digit being incremented. |
| alarm | 1-bit LED output which shows that it is time to take a medication. |
| counter | 8-bit output which counts the number of medications taken. |
| digitone, digittwo, digitthree, digitfour | 4-bit outputs which hold the value of four timer digits on the seven-segment display. |

We defined the 4-bit intermediate signals r1, r2, r3 and r4 to send the current digit value to the outputs digitone, digittwo, digitthree, digitfour. We defined the 4-bit intermediate signals d1, d2, d3 and d4 to store the input of time interval entered by the user. In this way, the interval can be used again and the user does not need to give this information again. The 4-bit intermediate signals s, s1, m, and m1 are the values of second and minute digits. Countdown operations are performed over these values. The 1-bit intermediate signal for\_alarm gets the values 1 or 0 in the different conditions and it is assigned to the output alarm. The 4-bit intermediate signals counter1 and counter2 are the ‘ones digit’ and ‘tens digit’ of medication counter correspondingly. Both are assigned to a half of the output counter.

In the first if statement, when the input set\_mode is made 1 by the user, the time interval of medications can be set. A digit is selected with the SET\_SELECT input and the selected digit is incremented in each rising edge of the input PLUSONE. As the signals r2 and r4 corresponds the ‘ones digit’ of second and minute, they return to 0 after 9. The signals r1 and r3 return to 0 after 5. Once the signals r1, r2, r3 and r4 are specified by the user, they are stored into the signals d1, d2, d3 and d4 for the next use.

If the set\_mode switch is off and the start switch is on, the countdown signals m1, m, s1 and s are assigned to the digit value signals r1, r2, r3 and r3 correspondingly. These m1, m, s1 and s signals decrement with a hierarchy in each rising edge of the SCLK input. s decrements until it is equal to “0000”. When it is “0000”, s1 decrements once and s turns to “1001” which is decimallyequal to 9. If s1 is also “0000”, then m decrements once and s1 turns to “0101” which is decimally equal to 5. We applied this logic for the decrements of the minute and second digits. Once all m1, m, s1 and s get the value “0000”, the signal for\_alarm gets the value ‘1’ and countdown is over. At his moment, if the start switch is turned off, the signal for\_alarm gets the value ‘0’ and the signals m1, m, s1 and s are updated with the initial stored information of the signals d1, d2, d3, d4. If both set\_mode and start switches are off, a new time interval can be inputted.

Besides, for each rising edge of the 'for\_alarm' signal, counter1 (and if needed counter2) increments. With the reset button, the user can reset the timer digits and return the counter1 to “0001”, the counter2 to “0000”. Additionally, one of the led\_digit bits turns on with the corresponding SET\_SELECT combination. At the end of the Timer module, the outputs get the value of the related intermediate signals after the above operations are performed.

**MAIN.vhd**

In the main module, we combined the components clock\_generator, driver, decoder and timer using port maps. They feed each other with their inputs and outputs as explained above for each module.

In addition, we defined the intermediate signals to make the connections between the port maps. We used the wire\_hundredhz\_clock to assign the 100-Hz clock output of the clock\_generator module to the clock input of the driver module. We defined the wire\_secondclock signal which we used to assign the second clock output of the clock\_generator module to the clock input of the Timer module. The intermediate signal wire\_sevseg\_data takes the output sev\_seg\_data of the driver module and assigns it to the input of the decoder module. With the intermediate signals minute1, minute, second1 and second, the timer digits from the Timer module are sent to the inputs D1, D2, D3 and D4 of the driver module. Lastly, we used the intermediate signal counter to assign the counter output of the Timer module to the input D5 and D6 of the driver module.

Final outputs are LED\_digit, ALARM, SEVSEG\_DATA and SEVSEG\_CONTROL. LED\_digit indicates the current digit being incremented. ALARM shows that it is time to take a medication. SEVSEG\_DATA and SEVSEG\_CONTROL are used to specify the value and leds of seven-segment display digits.

**main.sim.vhd**

This is our simulation code. We need at least 1 second to show that the alarm is ringing after 1 second. In other words, we set the time to 00.01 to be able to show in the simulation timing diagram. The master clock period is 10ns because we want it to be 100 MHz.

First, we reset all the values. After 10ns, SET\_MODE and PLUS\_ONE is 1 because we want to set the time and now the time is set to 00.01. After 10ns, START is 1 because we want to start countdown. After 1 second, the countdown is over and the time is 00.00 and “alarm” becomes 1.

**MAIN.ucf**

In this file, we assigned our inputs and outputs to pins. MCLK is already assigned to P40 in the board. “START” input is a switch and we assigned it to P78. “SET\_MODE” input is a switch and we assigned it to P15. “RESET” input is a button and we assigned it to P37. “PLUS\_ONE” input is a button and we assigned it to P36. “SET\_SELECT[0]” input and “SET\_SELECT[1]” input are switches and we assigned them to P12 and P5, respectively. The output alarm is a led and we assigned it to P86. "LED\_digit[0]", "LED\_digit[1]", "LED\_digit[2]", "LED\_digit[3]" are outputs and we assigned them to P16, P13, P6, P3, respectively. Finally, we assigned each bit of “SEVSEG\_CONTROL” to each digit of the seven-segment display and we assigned each bit of “SEVSEG\_DATA” to each segment.

When we push the switches and buttons, their associated signal becomes 1. Leds' corresponding signal becomes 1 when Leds are light.

1. **Problems encountered, errors and warnings resolved**

One of us had the red FPGA board and the other had the black one, so the seven-segment control locations were slightly different. We noticed this and made the code compatible for both FPGA boards.

Our FPGA boards could not be recognized by our computers in our first attempts. It was taking some time for FPGA boards to be recognized, maybe it is because they are old. We often forgot to put semicolons at the end of each line or forgot to put some signs such as > or parentheses so We encountered many time this error ‘Syntax error near " : " ’ ‘Syntax error near "or " ’ ‘Syntax error near "and " ’…

There is a limitation: We cannot easily show our code in the simulation timing diagram because we need at least 1 second to show the changes in the alarm signal. It takes a long time to wait for the simulation timing diagram, which is extended to 1 second. Therefore, it is very difficult to simulate our code to see if the alarm signal is 1 when the countdown is over. Therefore, on the test bench, we simulated for when the clock is set to 00.01.

1. **For improvement**

We could use a buzzer to indicate the alarm.

We could increase the number of digits which medication counter contains. We only used 2 digits to indicate the medication number.

We could add hours for time setting. We used just seconds and minutes to easily display it on the FPGA board.

1. **Conclusion**

Before starting the project, we examined the previous Alarm System project [1] and decided to develop the Medication Reminder. Our project, with its advanced and useful features, differs from the Alarm System project. We got help from this project just to be able to take a time interval from the user. However, our project offers more additional features. The Medication Reminder can save the alarm information and repeat the process. In addition, the Medication Reminder has a countdown for an alarm, differently from the Alarm System. The Medication Reminder also constantly gives the information of the number of taken medications.

Even though the FPGA boards were delivered to us late, we got much more familiar with the boards thanks to the project. Collaborating online was a challenge for us, but we overcame it by spending more time on the project.

Additionally, it is possible to improve the Medication Reminder further by connecting a buzzer to the FPGA board.

**References**

1. <https://drive.google.com/drive/folders/0B0QTMepM81hdVEpMQU1wcnMxQU0>
2. Eduvance (2016, November 17). ‘VHDL Lecture 23 Lab 8 - Clock Dividers and Counters’ [Video file]. Retrieved from : <https://www.youtube.com/watch?v=2YbelsD79Q4&list=PLZv8x7uxq5XY-IQfQFb6mC6OXzz0h8ceF&index=16>
3. Realfinetime (2021, January 14). FUNNY ELECTRONICS, ‘Control Common Anode Seven Segment Display Using Arduino and CD4543B’. Retrieved from: <http://www.learnerswings.com/2014/04/control-common-anode-seven-segment.html>

**Appendix 1. Lab source code**

Main.vhd code is shown below.

|  |
| --- |
| ----------------------------------------------------------------------------------  -- Company:  -- Engineer:  --  -- Create Date: 14:35:47 01/09/2021  -- Design Name:  -- Module Name: MAIN - Behavioral  -- Project Name:  -- Target Devices:  -- Tool versions:  -- Description:  --  -- Dependencies:  --  -- Revision:  -- Revision 0.01 - File Created  -- Additional Comments:  --  ----------------------------------------------------------------------------------  library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  entity MAIN is  PORT ( SET\_SELECT : IN STD\_LOGIC\_VECTOR (1 DOWNTO 0);  MCLK : IN STD\_LOGIC;  SET\_MODE : IN STD\_LOGIC;  RESET : IN STD\_LOGIC;  PLUS\_ONE : IN STD\_LOGIC;  START :IN STD\_LOGIC;    LED\_DIGIT : out std\_logic\_vector(3 downto 0);  ALARM : OUT STD\_LOGIC ;  SEVSEG\_DATA : OUT STD\_LOGIC\_VECTOR (6 DOWNTO 0);  SEVSEG\_CONTROL : OUT STD\_LOGIC\_VECTOR (7 DOWNTO 0));  end MAIN;  architecture Behavioral of MAIN is  --intermediate signals  signal WIRE\_HUNDREDHZ\_CLOCK : std\_logic;  signal WIRE\_SEVSEG\_DATA : std\_logic\_vector( 3 downto 0);  signal WIRE\_SECONDCLOCK : std\_logic;  signal minute : std\_logic\_vector( 3 downto 0);  signal minute1 : std\_logic\_vector( 3 downto 0);  signal second : std\_logic\_vector( 3 downto 0);  signal second1 : std\_logic\_vector( 3 downto 0);  signal COUNTER : std\_logic\_vector (7 downto 0);  BEGIN  --ADD CLOCK GENERATOR  CLOCK\_GENERATOR : entity WORK.CLOCK\_GENERATOR PORT MAP(  MCLK => MCLK,    HUNDREDHZCLOCK => WIRE\_HUNDREDHZ\_CLOCK,  SECONDCLK => WIRE\_SECONDCLOCK  );  --ADD DRIVER  DRIVER : ENTITY WORK.SEVSEG\_DRIVER PORT MAP(  D1 => minute1,  D2 => minute,  D3 => second1,  D4 => second,  D5 => COUNTER(7 downto 4),  D6=> COUNTER (3 downto 0),  CLK => WIRE\_HUNDREDHZ\_CLOCK,  SEV\_SEG\_DATA => WIRE\_SEVSEG\_DATA,  SEV\_SEG\_DRIVER => SEVSEG\_CONTROL  );  --ADD DECODER  DECODER : ENTITY WORK.SEVSEG\_DECODER PORT MAP(  INPUT => WIRE\_SEVSEG\_DATA,  SEVSEG\_BUS => SEVSEG\_DATA  );  --ADD Timer  TIMER : ENTITY WORK.Timer PORT MAP(  RST => RESET,  set\_mode => SET\_MODE,  PLUSONE=> PLUS\_ONE,  SET\_SELECT => SET\_SELECT,  start => START,    digitfour =>second ,  digitthree =>second1,  digittwo=>minute ,  digitone =>minute1,  SCLK => WIRE\_SECONDCLOCK,  alarm => ALARM,  led\_digit => LED\_DIGIT,  counter => COUNTER  );  END BEHAVIORAL; |

Timer.vhd is shown below

|  |
| --- |
| ----------------------------------------------------------------------------------  -- Company:  -- Engineer:  --  -- Create Date: 14:42:57 01/09/2021  -- Design Name:  -- Module Name: Time - Behavioral  -- Project Name:  -- Target Devices:  -- Tool versions:  -- Description:  --  -- Dependencies:  --  -- Revision:  -- Revision 0.01 - File Created  -- Additional Comments:  --  ----------------------------------------------------------------------------------  library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  USE IEEE.STD\_LOGIC\_ARITH.ALL;  USE IEEE.STD\_LOGIC\_UNSIGNED.ALL;  USE IEEE.NUMERIC\_STD.ALL;  entity Timer is  Port (SCLK: IN std\_logic; --clock input  set\_mode : IN STD\_LOGIC; --the mode to set an alarm  start : in std\_logic; --the switch to start countdown  RST : IN STD\_LOGIC; -----the button to reset the timer and medication counter  SET\_SELECT : IN STD\_LOGIC\_VECTOR(1 downto 0); --to select a digit being incremented while setting an alarm  PLUSONE : IN STD\_LOGIC; --to increment the selected digit    led\_digit : out std\_logic\_vector(3 downto 0); --indicates the current digit being incremented  alarm: out std\_logic ; --LED output showing that it is time to take a medication  counter: out std\_logic\_vector(7 downto 0); --counts the number of medications taken      digitone : out std\_logic\_vector(3 downto 0) ; --the digits used to display timer  digittwo : out std\_logic\_vector(3 downto 0) ;  digitthree : out std\_logic\_vector(3 downto 0);  digitfour : out std\_logic\_vector(3 downto 0)    );  end Timer;  architecture Behavioral of Timer is  --intermediate signals  signal r1 : std\_logic\_vector(3 downto 0):= "0000";  SIGNAL r2 : STD\_LOGIC\_VECTOR(3 DOWNTO 0) := "0000";  SIGNAL r3 : STD\_LOGIC\_VECTOR(3 DOWNTO 0) := "0000";  SIGNAL r4 : STD\_LOGIC\_VECTOR(3 DOWNTO 0) := "0000";    signal d1 : std\_logic\_vector(3 downto 0):= "0000";  SIGNAL d2 : STD\_LOGIC\_VECTOR(3 DOWNTO 0) := "0000";  SIGNAL d3 : STD\_LOGIC\_VECTOR(3 DOWNTO 0) := "0000";  SIGNAL d4 : STD\_LOGIC\_VECTOR(3 DOWNTO 0) := "0000";    SIGNAL s : STD\_LOGIC\_VECTOR(3 DOWNTO 0):= "0000";  SIGNAL s1 : STD\_LOGIC\_VECTOR(3 DOWNTO 0) := "0000";  SIGNAL m : STD\_LOGIC\_VECTOR(3 DOWNTO 0) := "0000";  SIGNAL m1 : STD\_LOGIC\_VECTOR(3 DOWNTO 0):= "0000" ;    signal for\_alarm : std\_logic := '0';  signal counter1 : std\_logic\_vector (3 downto 0) := "0001";  signal counter2 : std\_logic\_vector (3 downto 0) := "0000";    begin  PROCESS\_SCLK : PROCESS(SCLK,RST)  begin  --minute and second digits are set here  if (set\_mode = '1') then  if (SET\_SELECT = "00" and rising\_edge(PLUSONE)) then  r4 <= r4 + 1 ;  if r4 = 9 then  r4 <= "0000";  end if;  end if;  if (SET\_SELECT = "01" and rising\_edge(PLUSONE)) then  r3 <= r3 + 1 ;  if r3 = 5 then  r3 <= "0000";  end if;  end if;  if (SET\_SELECT = "10" and rising\_edge(PLUSONE)) then  r2 <= r2 + 1 ;  if r2 = 9 then  r2 <= "0000";  end if;  end if;  if (SET\_SELECT = "11" and rising\_edge(PLUSONE)) then  r1 <= r1 + 1 ;  if r1 = 5 then  r1 <= "0000";  end if;  end if;    --the time period input is saved  d1<= r1;  d2<= r2;  d3<= r3;  d4<= r4;    --the second and minute digits are assigned to r1,r2,r3,r4  elsif (set\_mode='0' and start = '1') then  r1<= m1;  r2<= m;  r3<= s1;  r4<= s;  end if;    if(rising\_edge(SCLK))then  if(set\_mode='0' and start = '1') then --countdown starts  if(s >= "0000") then --decrements of the digits are defined  s <= s - 1;  if(s = "0000") then  s<= "1001";  s1<= s1 - 1;  if(s1 = "0000") then  m <= m - 1;  s1 <= "0101";  if(m = "0000") then  m <= "1001";  m1 <= m1 - 1;  if(m1 = "0000") then  m1 <= "0101";  end if;  end if;  end if;  end if;  end if;  --when countdown is over, digits are reset and the LED alarm is set to '1'.  if(s = "0000" and s1 = "0000" and m = "0000" and m1 = "0000" ) then  s <= "0000" ;  s1 <= "0000";  m <= "0000" ;  m1 <= "0000";  for\_alarm <= '1';  end if;    elsif (set\_mode='0' and start='0') then --once the 'start' switch is off, the LED alarm is reset  for\_alarm <= '0';  m1<= d1; --and digits are converted into the saved digits (initial state)  m<= d2;  s1<= d3;  s<= d4;    elsif (set\_mode='1' and start='0') then --when the 'mode' switch is on, a new time interval can be inputted  m1<= r1;  m<= r2;  s1<= r3;  s<= r4;  end if;  end if;    --for each rising edge of the 'for\_alarm' signal, medication counter will increment  if rising\_edge (for\_alarm) then  counter1 <= counter1 + 1 ;  if(counter1 = "1001") then --our counter has two digits  counter2 <= counter2 + 1 ;  counter1 <= "0000" ;  end if;  end if;    --with reset button, everything on the sevensegment will be reset  if RST = '1' then  r1<="0000";  r2<="0000";  r3<="0000";  r4<="0000";  counter1 <= "0001";  counter2 <= "0000";  end if;  end process;  --the LED bus shows the selected digit while setting an alarm  WITH SET\_SELECT SELECT LED\_DIGIT <=  "0001" when "00", --led 16  "0010" when "01", --led 13  "0100" when "10", --led 6  "1000" when "11", --led 3  "0000" WHEN OTHERS;  --the intermediate signals are assigned into the related outputs  digitone <=r1 ;  digittwo <=r2;  digitthree<=r3 ;  digitfour <=r4;  alarm <= for\_alarm;    counter(7 downto 4) <= counter2;  counter(3 downto 0) <= counter1;  end Behavioral; |

CLOCK\_GENERATOR.vhd is shown below

|  |
| --- |
| ----------------------------------------------------------------------------------  -- Company:  -- Engineer:  --  -- Create Date: 14:39:27 01/09/2021  -- Design Name:  -- Module Name: CLOCK\_GENERATOR - Behavioral  -- Project Name:  -- Target Devices:  -- Tool versions:  -- Description:  --  -- Dependencies:  --  -- Revision:  -- Revision 0.01 - File Created  -- Additional Comments:  --  ----------------------------------------------------------------------------------  library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  USE IEEE.STD\_LOGIC\_ARITH.ALL;  USE IEEE.STD\_LOGIC\_UNSIGNED.ALL;  entity CLOCK\_GENERATOR is  Port ( MCLK : in STD\_LOGIC;  SECONDCLK : out STD\_LOGIC;  HUNDREDHZCLOCK : out STD\_LOGIC);    end CLOCK\_GENERATOR;  architecture Behavioral of CLOCK\_GENERATOR is  SIGNAL COUNTER : integer := 0;  SIGNAL COUNTERS : integer := 0;  signal temp : std\_logic := '0';  signal temps: std\_logic := '0';  begin  CLK\_PROCESS: PROCESS(MCLK)  BEGIN  if rising\_edge(mclk) then  counter <= counter + 1;  if counter<50000 then  temp <= '1';  else  temp <= '0';  counter <= 0;  end if ;  end if;  if rising\_edge(mclk) then  counters <= counters + 1;  if counters<100000000 then  temps <= '1';  else  temps <= '0';  counters <= 0;  end if;  end if;    END PROCESS;    hundredhzclock <= temp;  secondclk <= temps ;  end Behavioral; |

SEVSEG\_DRIVER.vhd is shown below

|  |
| --- |
| ----------------------------------------------------------------------------------  -- Company:  -- Engineer:  --  -- Create Date: 14:40:54 01/09/2021  -- Design Name:  -- Module Name: SEVSEG\_DRIVER - Behavioral  -- Project Name:  -- Target Devices:  -- Tool versions:  -- Description:  --  -- Dependencies:  --  -- Revision:  -- Revision 0.01 - File Created  -- Additional Comments:  --  ----------------------------------------------------------------------------------  library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  USE IEEE.STD\_LOGIC\_ARITH.ALL;  USE IEEE.STD\_LOGIC\_UNSIGNED.ALL;  USE IEEE.NUMERIC\_STD.ALL;  entity SEVSEG\_DRIVER is  PORT (  D4 : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);  D3 : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);  D2 : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);  D1 : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);  D5 : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);  D6 : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);  CLK : IN STD\_LOGIC;  SEV\_SEG\_DATA : OUT STD\_LOGIC\_VECTOR (3 DOWNTO 0);  SEV\_SEG\_DRIVER : OUT STD\_LOGIC\_VECTOR (7 DOWNTO 0)  );  END SEVSEG\_DRIVER;  ARCHITECTURE BEHAVIORAL OF SEVSEG\_DRIVER IS  SIGNAL COUNTER : STD\_LOGIC\_VECTOR(2 DOWNTO 0) := "000";  BEGIN  --INCREMENT COUNTER  PROCESS\_CLK : PROCESS(CLK)  BEGIN  IF(CLK'EVENT AND CLK = '1') THEN  COUNTER <= COUNTER + 1;  IF(COUNTER = "101" )THEN  COUNTER <= "000";  END IF;  END IF;  END PROCESS;  -- SEV\_SEG DATA  WITH COUNTER SELECT SEV\_SEG\_DATA <=  D1 WHEN "000",  D2 WHEN "001",  D3 WHEN "010",  D4 WHEN "011",  D5 WHEN "100",  D6 WHEN "101",  "1001" WHEN OTHERS;  --DATA END  --SEV\_SEG\_CONTROLLER  WITH COUNTER SELECT SEV\_SEG\_DRIVER <=  "01111111" WHEN "000",  "10111111" WHEN "001",  "11011111" WHEN "010",  "11101111" WHEN "011",  "11111101" WHEN "100",  "11111110" WHEN "101",  "00001111" WHEN OTHERS;  --SEV\_SEG CONTROLLER END  END BEHAVIORAL; |

SEVSEG\_DECODER.vhd is shown below

|  |
| --- |
| ----------------------------------------------------------------------------------  -- Company:  -- Engineer:  --  -- Create Date: 14:41:51 01/09/2021  -- Design Name:  -- Module Name: SEVSEG\_DECODER - Behavioral  -- Project Name:  -- Target Devices:  -- Tool versions:  -- Description:  --  -- Dependencies:  --  -- Revision:  -- Revision 0.01 - File Created  -- Additional Comments:  --  ----------------------------------------------------------------------------------  library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  entity SEVSEG\_DECODER is  PORT ( INPUT : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);  SEVSEG\_BUS : OUT STD\_LOGIC\_VECTOR (6 DOWNTO 0));  END SEVSEG\_DECODER;  ARCHITECTURE BEHAVIORAL OF SEVSEG\_DECODER IS  BEGIN  WITH INPUT SELECT SEVSEG\_BUS <=  "0000001" when "0000", --0  "1001111" when "0001", --1  "0010010" when "0010", --2  "0000110" when "0011", --3  "1001100" when "0100", --4  "0100100" when "0101", --5  "0100000" when "0110", --6  "0001111" when "0111", --7  "0000000" when "1000", --8  "0000100" when "1001", --9  "0000100" WHEN OTHERS;  END BEHAVIORAL; |

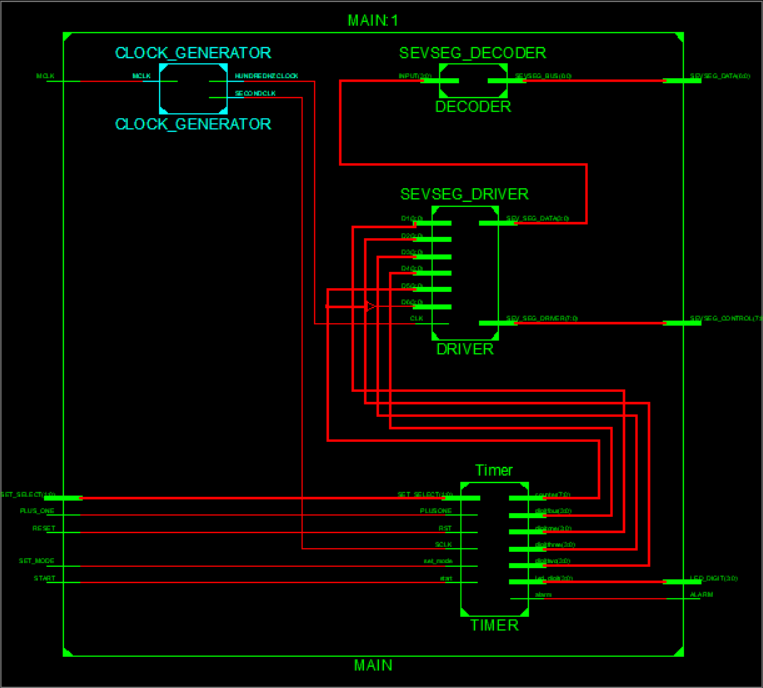
Main.ucf is shown below (for red FPGA board)

|  |
| --- |
| NET "SEVSEG\_CONTROL[7]" LOC = P60;  NET "SEVSEG\_CONTROL[6]" LOC = P61;  NET "SEVSEG\_CONTROL[5]" LOC = P59;  NET "SEVSEG\_CONTROL[4]" LOC = P57;  NET "SEVSEG\_CONTROL[3]" LOC = P52;  NET "SEVSEG\_CONTROL[2]" LOC = P56;  NET "SEVSEG\_CONTROL[1]" LOC = P50;  NET "SEVSEG\_CONTROL[0]" LOC = P49;  NET "SEVSEG\_DATA[6]" LOC = P71;  NET "SEVSEG\_DATA[5]" LOC = P62;  NET "SEVSEG\_DATA[4]" LOC = P65;  NET "SEVSEG\_DATA[3]" LOC = P72;  NET "SEVSEG\_DATA[2]" LOC = P73;  NET "SEVSEG\_DATA[1]" LOC = P98;  NET "SEVSEG\_DATA[0]" LOC = P64;  NET "MCLK" LOC = P40;  NET "START" LOC = P78;  NET "SET\_MODE" LOC = P15;  NET "RESET" LOC = P37;  NET "PLUS\_ONE" LOC = P36;  NET "SET\_SELECT[0]" LOC = P12;  NET "SET\_SELECT[1]" LOC = P5;  NET "ALARM" LOC = P86;  NET "LED\_DIGIT[0]" LOC = P16;  NET "LED\_DIGIT[1]" LOC = P13;  NET "LED\_DIGIT[2]" LOC = P6;  NET "LED\_DIGIT[3]" LOC = P3; |

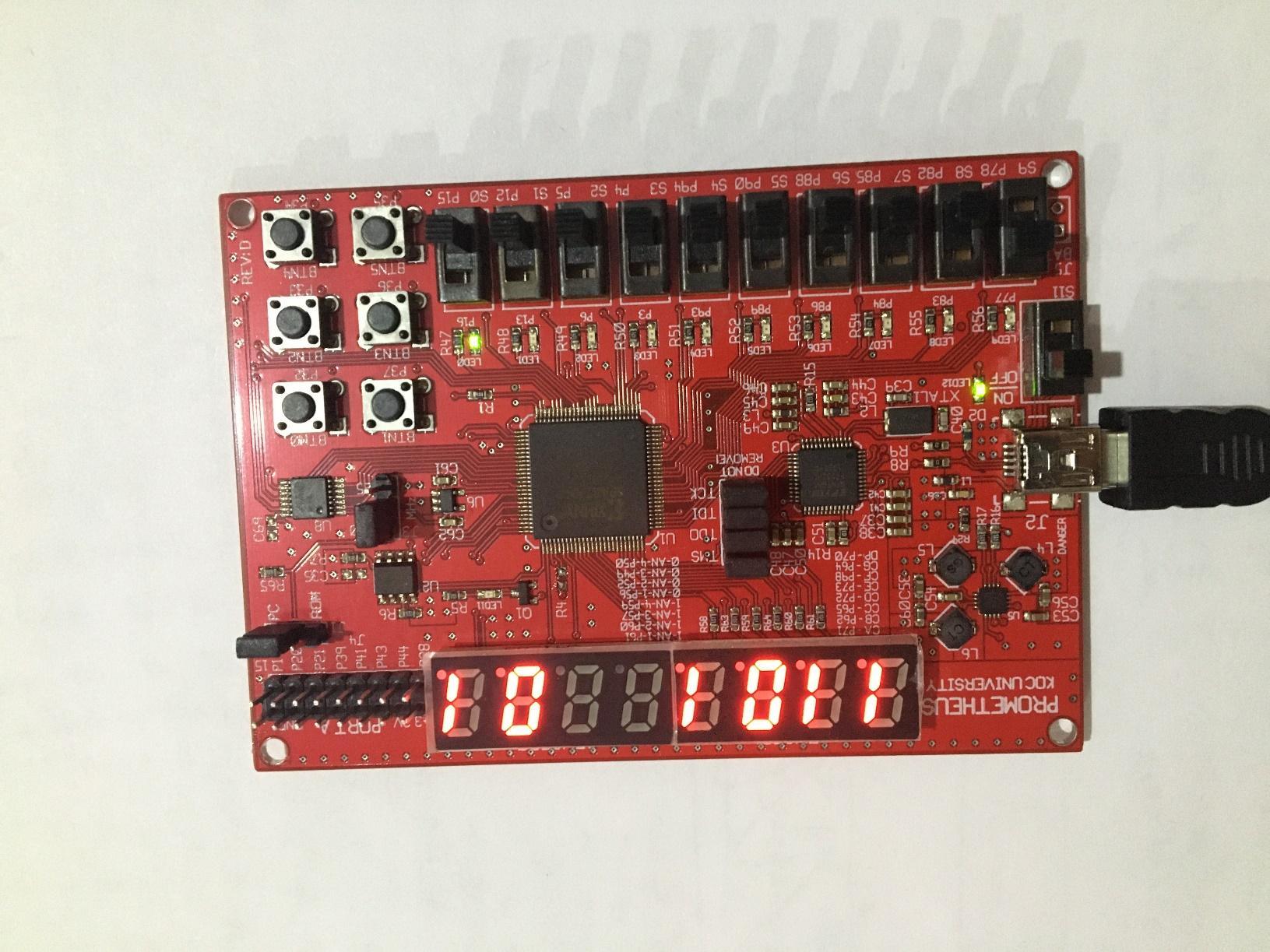
main.sim is shown below

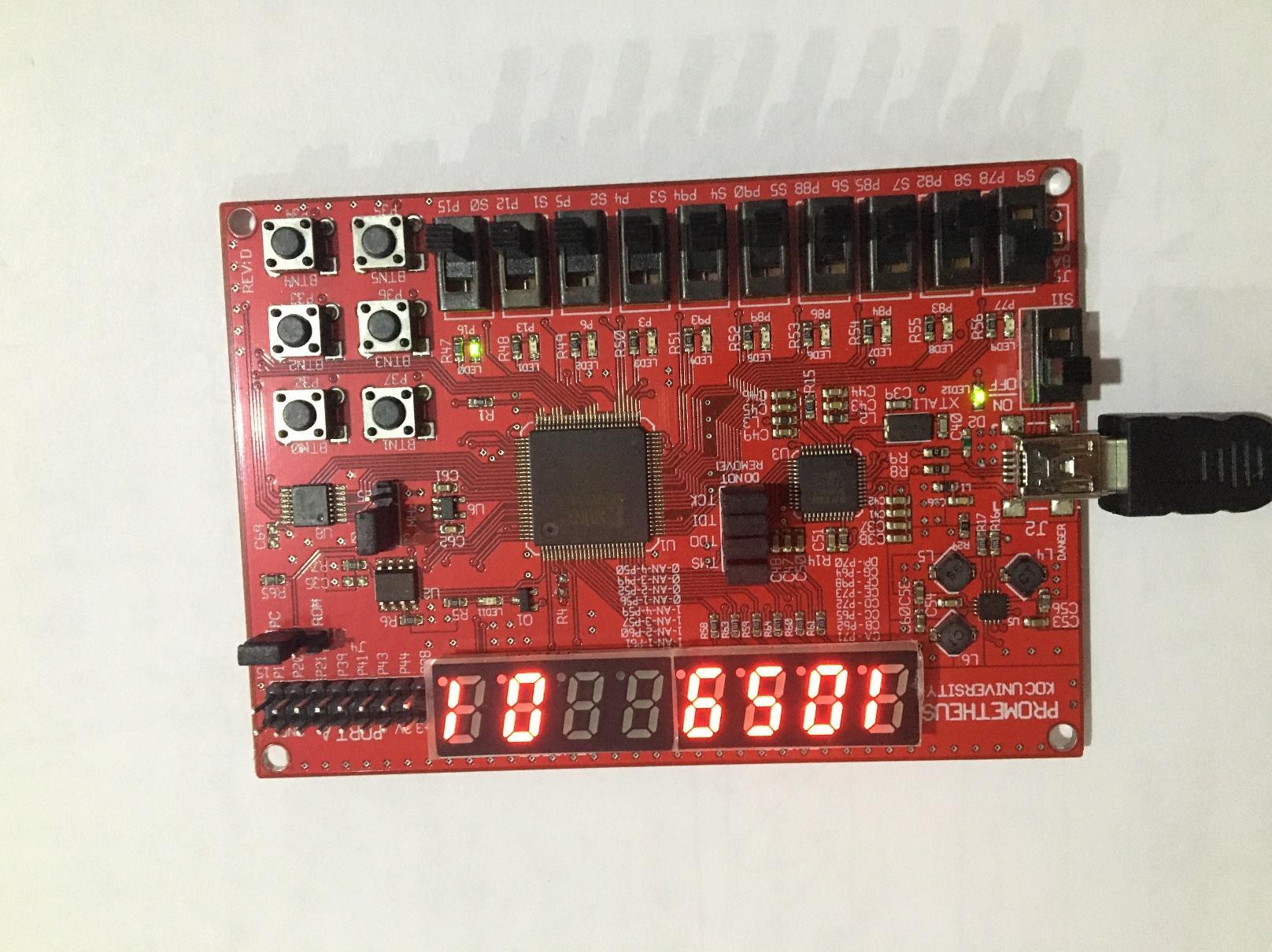
|  |
| --- |
| --------------------------------------------------------------------------------  -- Company:  -- Engineer:  --  -- Create Date: 23:58:44 01/09/2021  -- Design Name:  -- Module Name: C:/Users/Deniz/Desktop/Medication\_Reminder/main\_sim.vhd  -- Project Name: Medication\_Reminder  -- Target Device:  -- Tool versions:  -- Description:  --  -- VHDL Test Bench Created by ISE for module: MAIN  --  -- Dependencies:  --  -- Revision:  -- Revision 0.01 - File Created  -- Additional Comments:  --  -- Notes:  -- This testbench has been automatically generated using types std\_logic and  -- std\_logic\_vector for the ports of the unit under test. Xilinx recommends  -- that these types always be used for the top-level I/O of a design in order  -- to guarantee that the testbench will bind correctly to the post-implementation  -- simulation model.  --------------------------------------------------------------------------------  LIBRARY ieee;  USE ieee.std\_logic\_1164.ALL;    ENTITY main\_sim IS  END main\_sim;    ARCHITECTURE behavior OF main\_sim IS    -- Component Declaration for the Unit Under Test (UUT)    COMPONENT MAIN  PORT(  SET\_SELECT : IN std\_logic\_vector(1 downto 0);  MCLK : IN std\_logic;  SET\_MODE : IN std\_logic;  RESET : IN std\_logic;  PLUS\_ONE : IN std\_logic;  START : IN std\_logic;  LED\_digit : OUT std\_logic\_vector(3 downto 0);  ALARM : OUT std\_logic;  SEVSEG\_DATA : OUT std\_logic\_vector(6 downto 0);  SEVSEG\_CONTROL : OUT std\_logic\_vector(7 downto 0)  );  END COMPONENT;    --Inputs  signal SET\_SELECT : std\_logic\_vector(1 downto 0) := (others => '0');  signal MCLK : std\_logic := '0';  signal SET\_MODE : std\_logic := '0';  signal RESET : std\_logic := '0';  signal PLUS\_ONE : std\_logic := '0';  signal START : std\_logic := '0';  --Outputs  signal LED\_digit : std\_logic\_vector(3 downto 0);  signal ALARM : std\_logic;  signal SEVSEG\_DATA : std\_logic\_vector(6 downto 0);  signal SEVSEG\_CONTROL : std\_logic\_vector(7 downto 0);  -- Clock period definitions  constant MCLK\_period : time := 10 ns;    BEGIN    -- Instantiate the Unit Under Test (UUT)  uut: MAIN PORT MAP (  SET\_SELECT => SET\_SELECT,  MCLK => MCLK,  SET\_MODE => SET\_MODE,  RESET => RESET,  PLUS\_ONE => PLUS\_ONE,  START => START,  LED\_digit => LED\_digit,  ALARM => ALARM,  SEVSEG\_DATA => SEVSEG\_DATA,  SEVSEG\_CONTROL => SEVSEG\_CONTROL  );  -- Clock process definitions  MCLK\_process :process  begin  MCLK <= '0';  wait for MCLK\_period/2;  MCLK <= '1';  wait for MCLK\_period/2;  end process;    -- Stimulus process  stim\_proc: process  begin  SET\_MODE <= '0';SET\_SELECT <= "00"; PLUS\_ONE <= '0'; RESET <= '1'; START <= '0';  wait for MCLK\_period ;  SET\_MODE <= '1';SET\_SELECT <= "00"; PLUS\_ONE <= '1'; RESET <= '0'; START <= '0';  wait for MCLK\_period;  SET\_MODE <= '0';SET\_SELECT <= "00"; PLUS\_ONE <= '0'; RESET <= '0'; START <= '1';  wait for MCLK\_period\*100000000;  SET\_MODE <= '0';SET\_SELECT <= "00"; PLUS\_ONE <= '0'; RESET <= '0'; START <= '1';  -- insert stimulus here  wait;  end process;  END; |

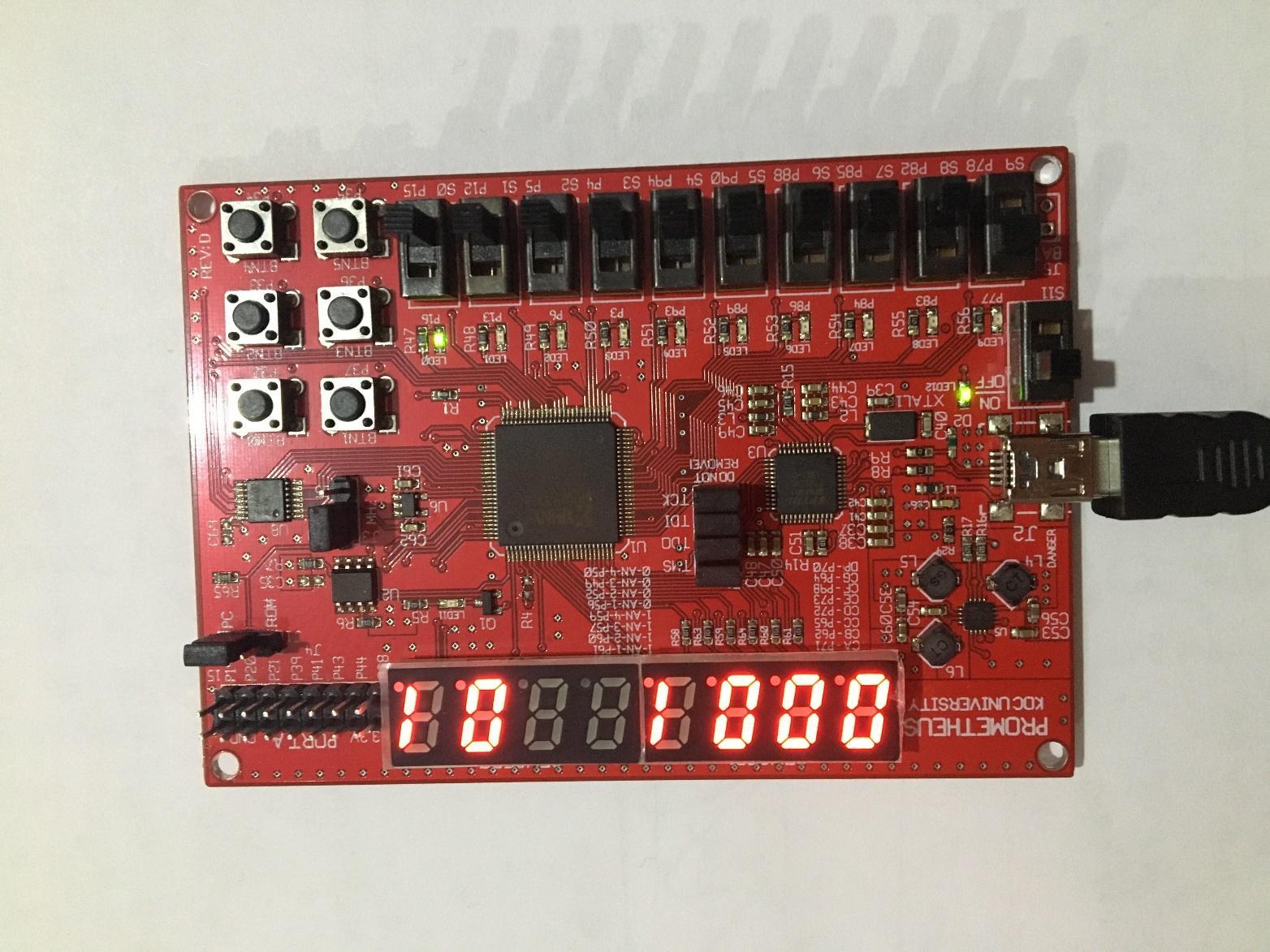
**Appendix 2. RTL schematics**

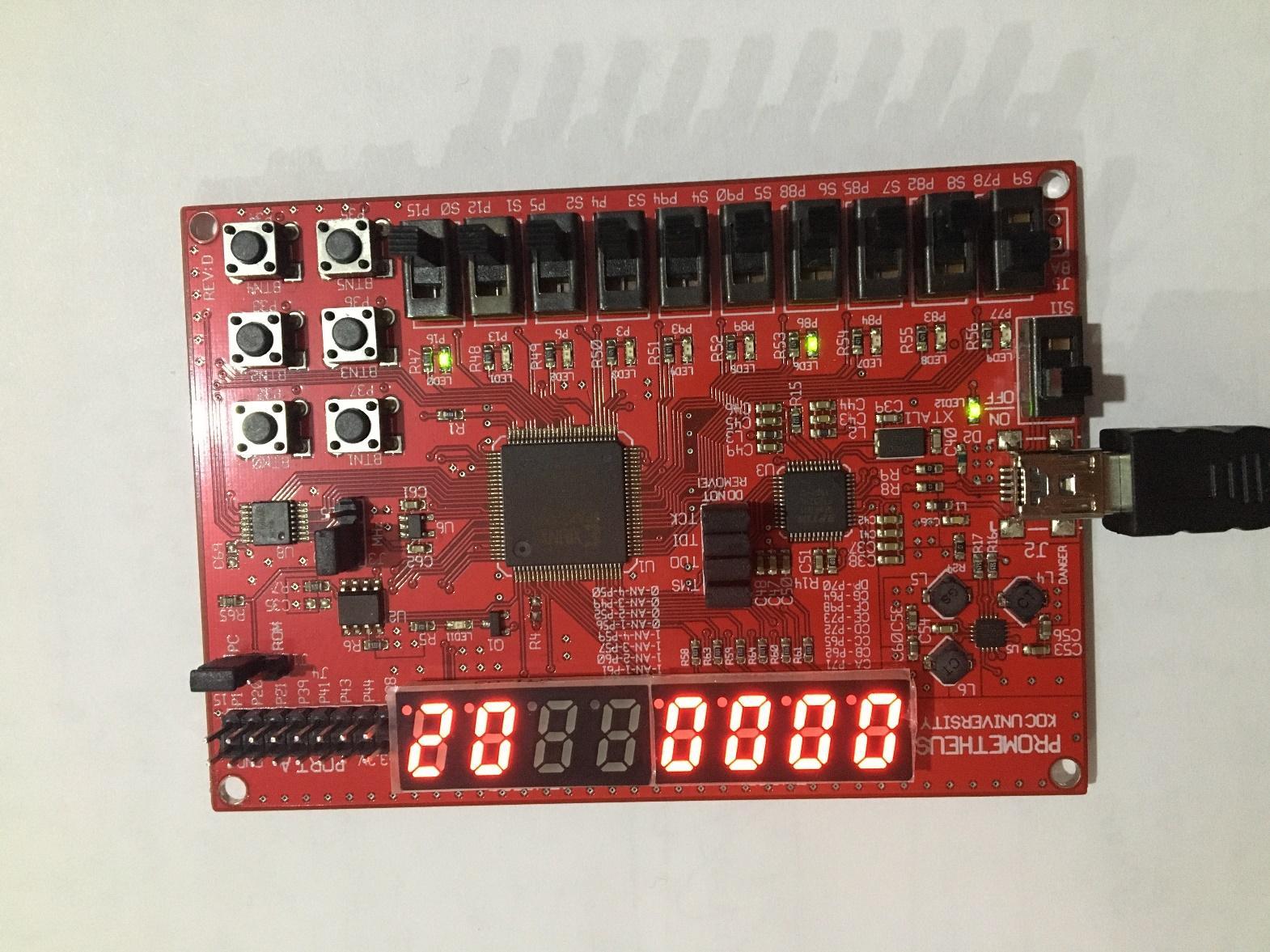
****

**Appendix 3. FPGA Board photos showing working code**

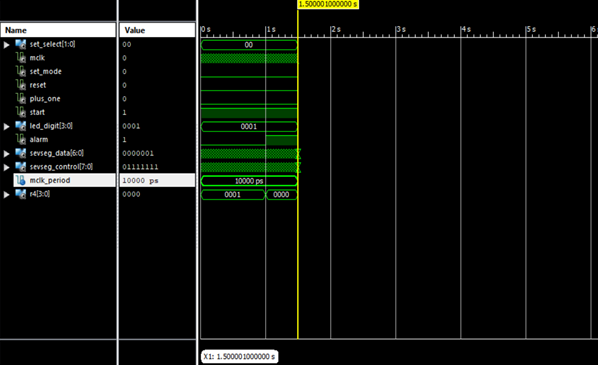
****

****

****

****

**Appendix 4. Simulation timing diagram**

****