UPPSALA UNIVERSITY

Ms. Embedded Systems - 2022

Digital Electronics Design with VHDL

TEMPERATURE LOGGER

SUBHANG VEMPATI

UPPSALA UNIVERSITET

14-05-2022

1. Contents

2. ABSTRACT

The project task is to design a temperature logger which gathers the temperature using temperature sensor LM75.This is implemented using I2C communication between DE2-115 the temperature sensor.

3. INTRODUCTION

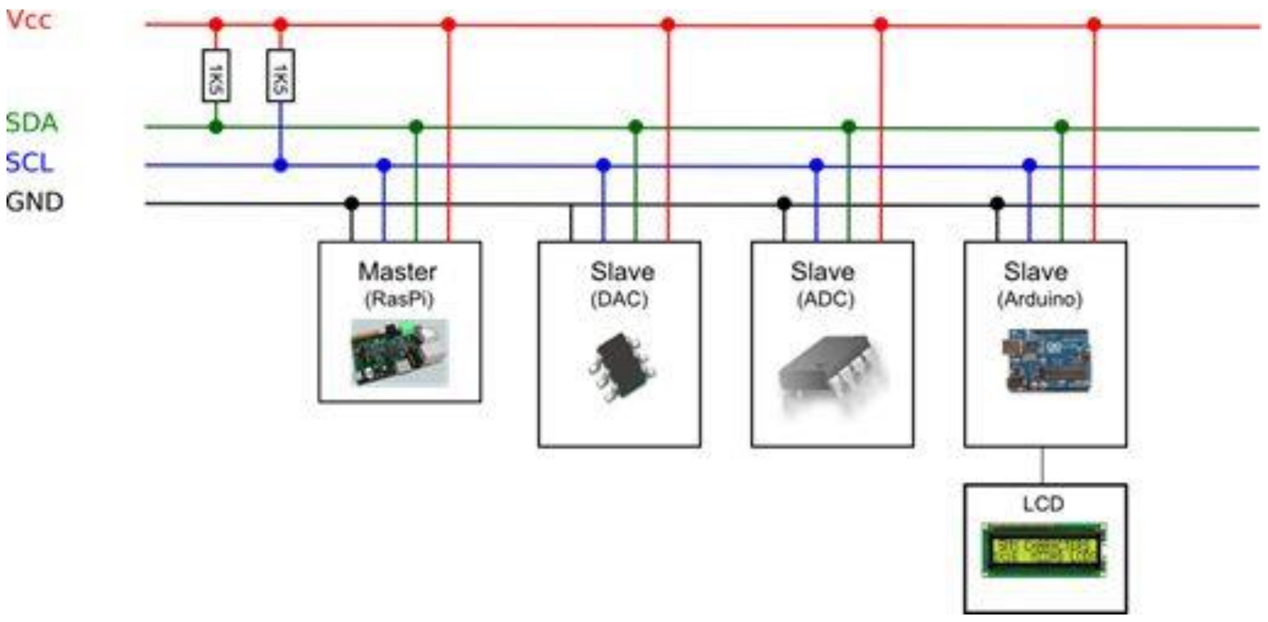


Figure: **Principle Drawing of I2C communication**

The basic principle of I2C communication is to write to or read from slave using 2 wires namely SDA (SERIAL DATA LINE) and SCL (SERIAL CLOCK LINE). The speciality of the I2C communication is there can N-number of masters and slaves in the circuit that can communicate between themselves. Here we are trying to fetch information from LM75 sensor which is working as a slave and display it on the LCD.

4. PROJECT DESCRIPTION

The process which is happening in the Temperature logger is explained below.

First, we create an I2C master which controls the temperature sensor. In the master we create a state machine which changes to the conditions which we need. The board DE2-115 provides us with a 50 MHz clock frequency which is way too high for the sensor to work with, so we design a clock divider which lowers the clock frequency and makes the master to work. For the LCD display we need a 400hz clock which is taken from the clock divider which we designed and finally I have taken an external serial clock signal which works with the serial data line, and this was implemented by taking an enable signal from the I2C master and giving it to a D flipflop which acts as a delayer. We have created a temperature register which divides most significant and least significant bits from data and sends it to a component which converts binary numbers into binary coded decimal numbers. This component uses a special algorithm called double dabble. Finally, LCD display is also implemented as a state machine in which each character is displayed is a state mentioned in the design.

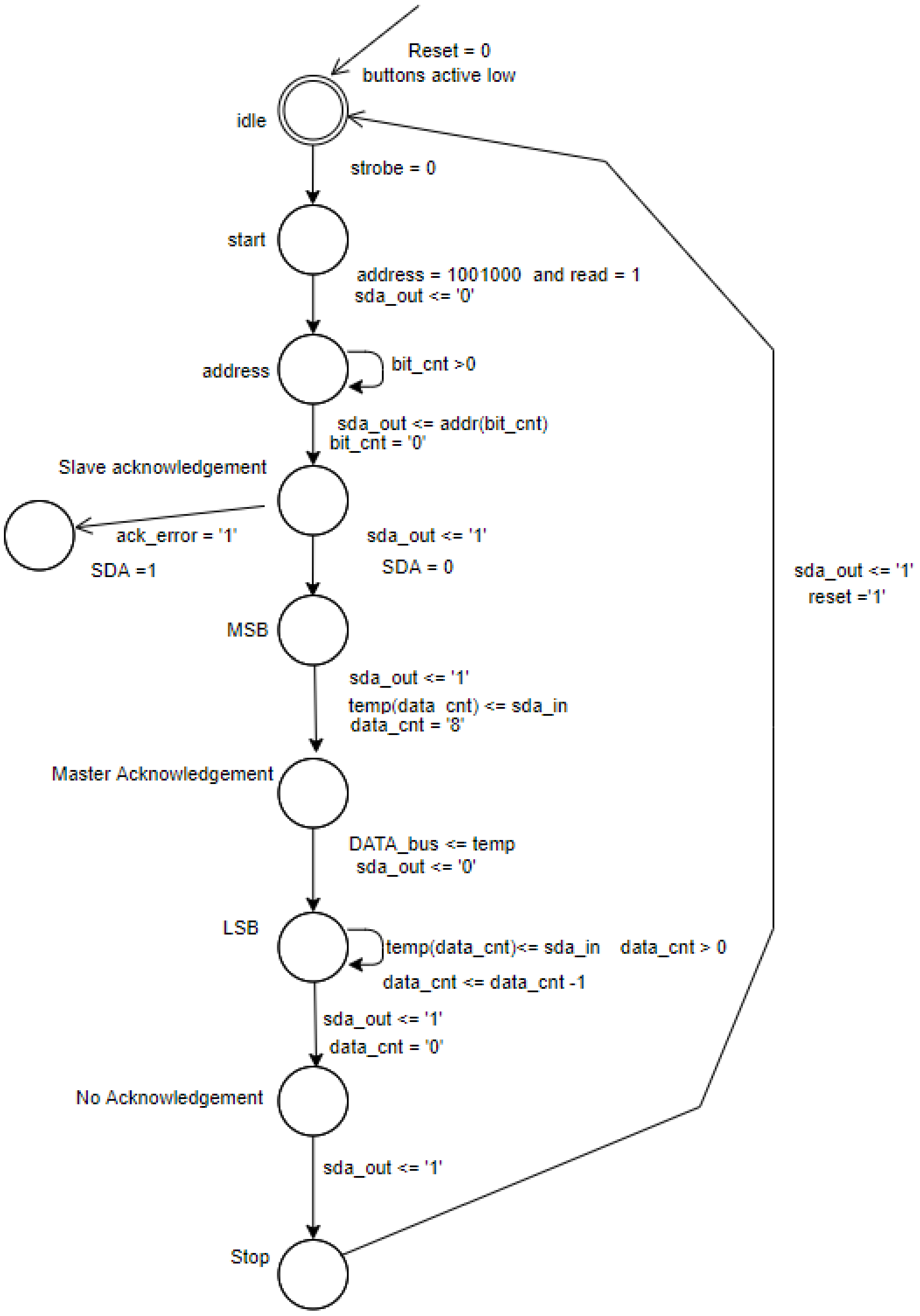


Figure: STATE MACHINE for I2C MASTER

5. IMPLEMENTATION AND SIMULATION

5.1 Clock Divider

CLOCK DIVIDER			
External signals	Port	Width (In bits)	Description
Clk_50	Input	1	Clock signal (50MHz)
Sclk_128	Output	1	Clock signal (200KHz)
Sclk_64	Output	1	Clock signal (100KHz)
Clk_400hz	Output	1	Clock signal (400 Hz)
Clk_strobe	Output	1	Clock signal (1 Hz)
Sample clock	Output	1	Clock signal (1 MHz)

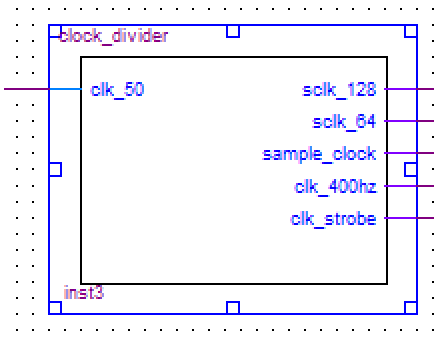


Figure: **clock divider BSF**

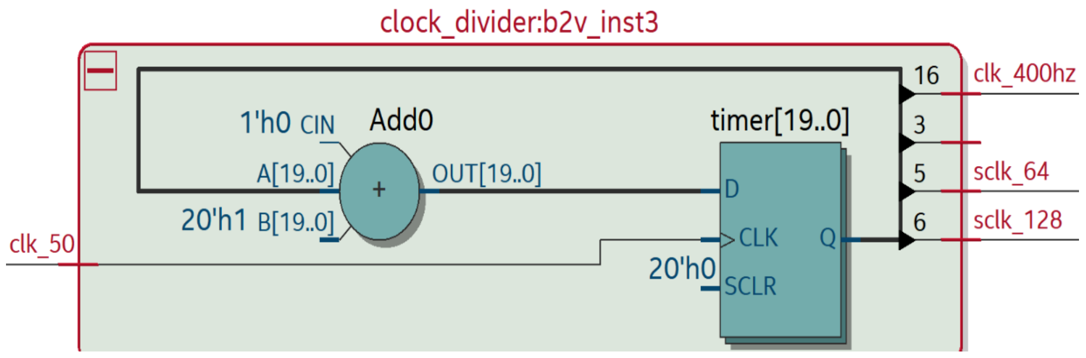


Figure: **clock divider RTL**

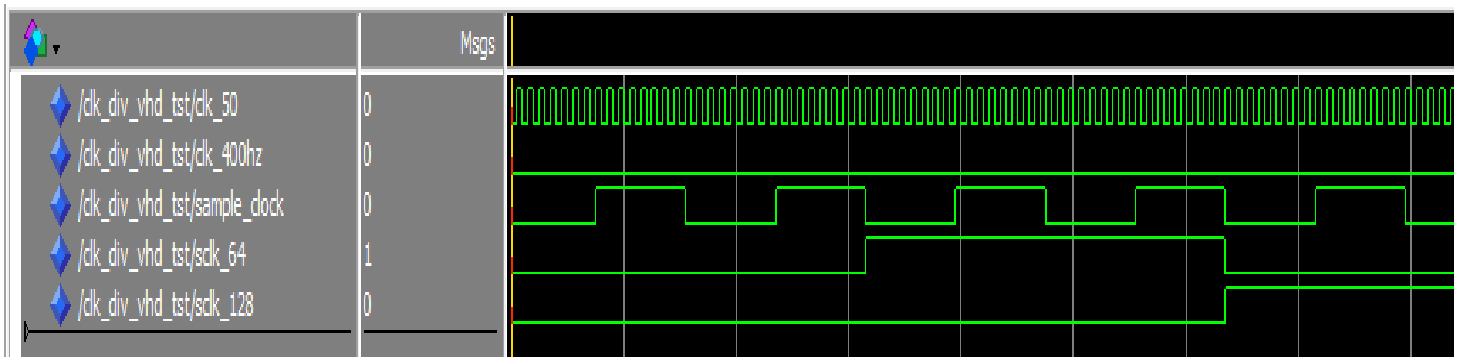

Msgs
/clk_div_vhd_tst/clk_50
/clk_div_vhd_tst/clk_400hz
/clk_div_vhd_tst/sample_clock
/clk_div_vhd_tst/sclk_64
/clk_div_vhd_tst/sclk_128
0
0
0
1
0

Figure: Clock 50 MHZ is given as input, and it is divided into various clock signals.

5.2 I2C MASTER

I/O Pins			
External signals	Port	Width (In bits)	Description
Clk	Input	1	Clock signal (200kHz)
Reset	Input	1	Global Reset signal from the press button
STROBE	Output	1	Strobe signal
Enable	Output	1	Enable signal
Data_bus	Output	16	16-bit instruction output

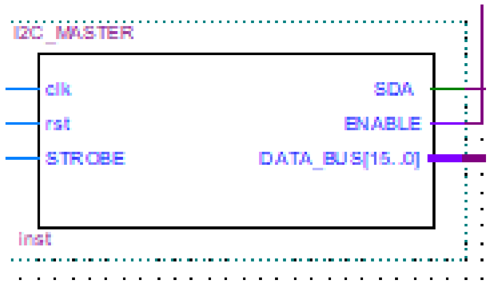


Figure: I2C BSF

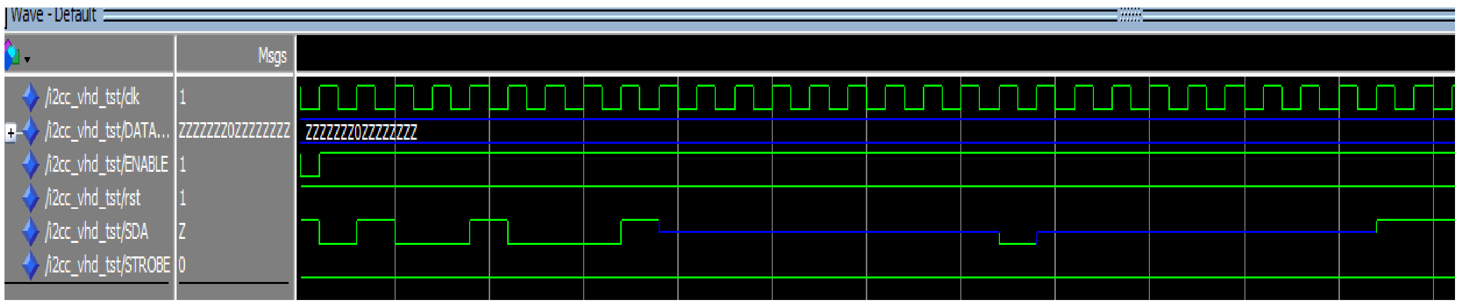


Figure: I2C Master simulation

As the SCL is external I have an Enable signal that controls the SCL, and the circuit has reset as '0' as the buttons pushed must be active low.

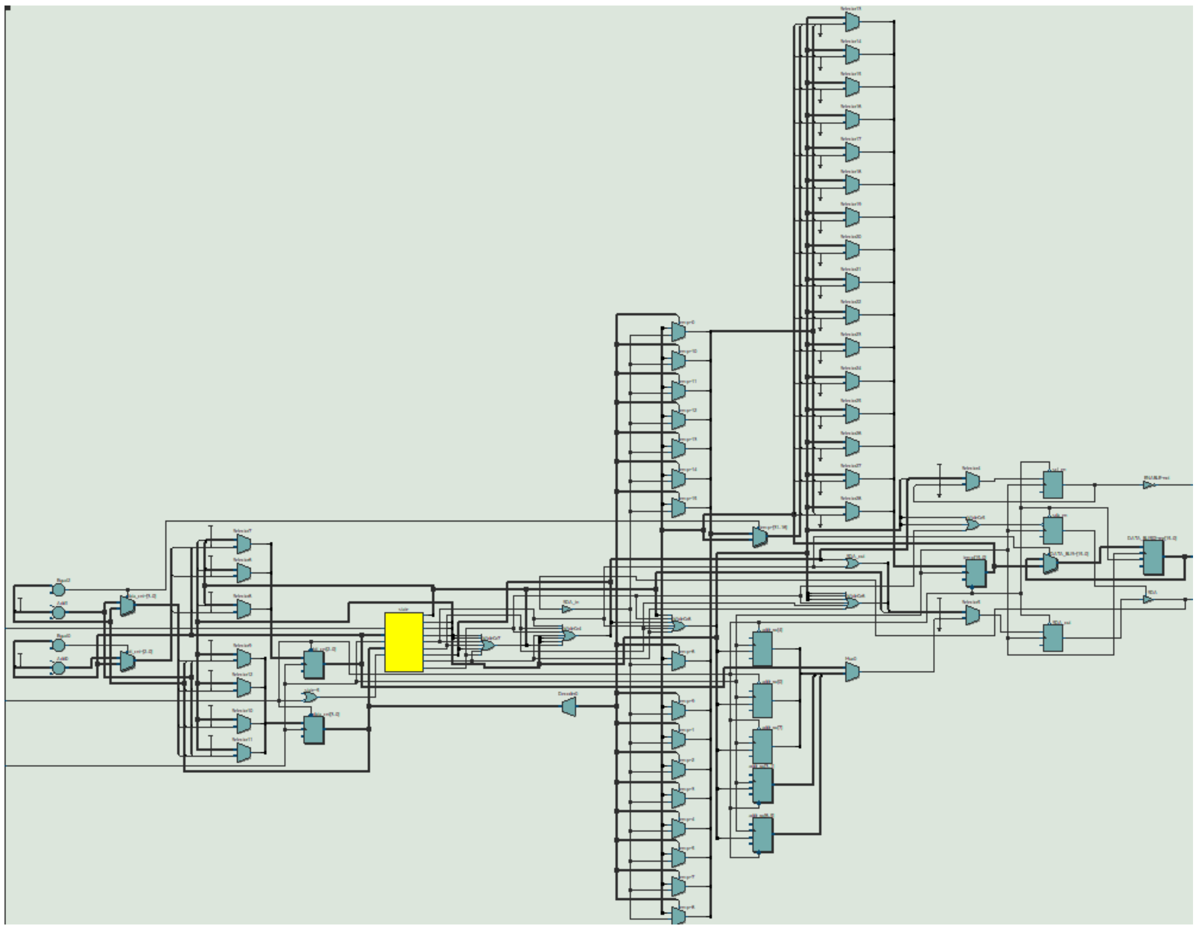


Figure: I2C MASTER RTL

I2C Master will be in read mode as it is reading the information given by the temperature sensor which is slave in the project. This communication is led by 2 wires namely SDA and SCL. I have implemented a I2C Master with an external serial clock. By implementing that way, we have less control over the data flow.

5.3 EXTERNAL SERIAL CLOCK

EXTERNAL SERIAL CLOCK			
External signals	Port	Width (In bits)	Description
Clock	Input	1	Clock signal (100 KHz)
Enable	Input	1	Enable signal
CLK_128	Input	1	Clock signal (200 KHz)
SCL	Output	1	Serial clock output

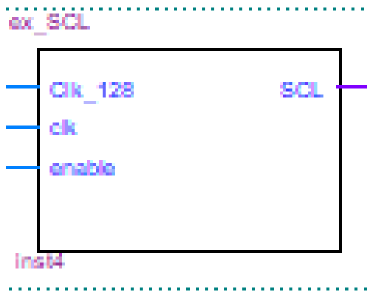


Figure: External Serial Clock BSF

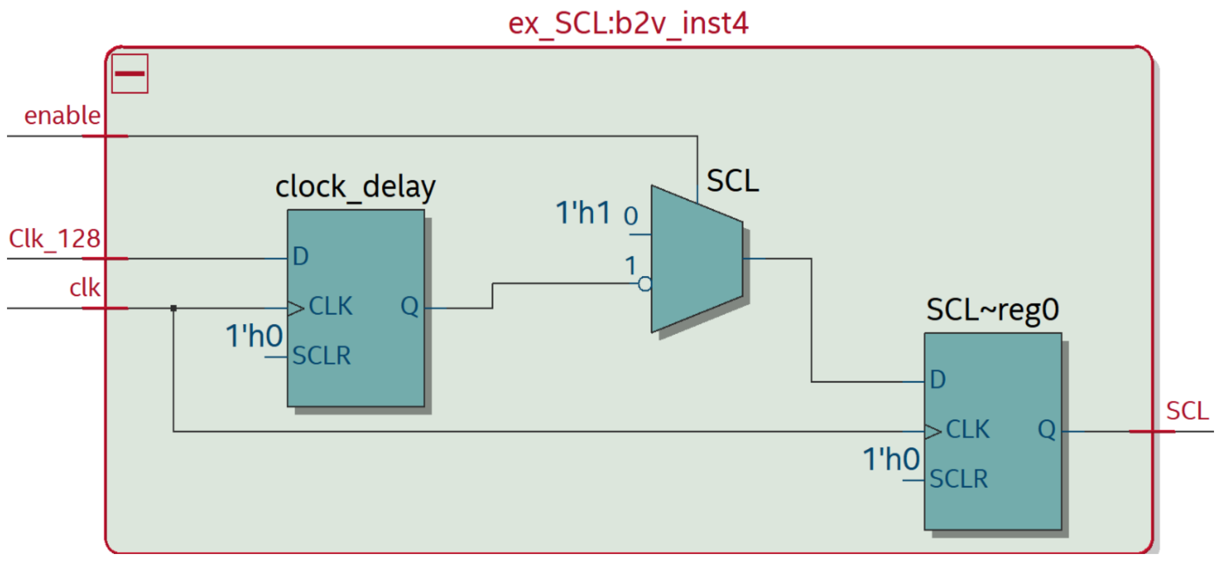


Figure: External Serial Clock RTL

5.4 TEMPERATURE REGISTER

Temperature Register			
External signals	Port	Width (In bits)	Description
Temp	Input	16	Temperature read from sensor by I2C MASTER
MSB	Output	8	Bits 15 down to 8 in the temperature
LSB	Output	1	Bit 7 in the temperature

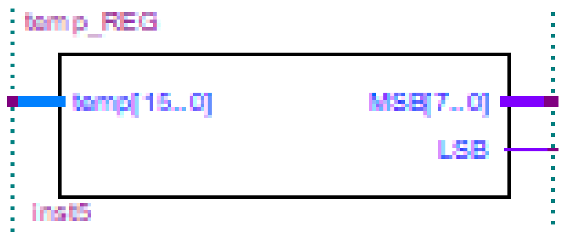


Figure: Temperature register BDF

5.5 Binary to BCD CONVERTER

PROGRAM COUNTER			
External signals	Port	Width (In bits)	Description
bin	Input	8	MSB from temperature register
BCD	Output	12	Converted bits of binary code

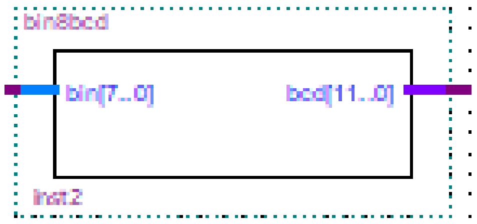


Figure: Binary to BCD converter BDF

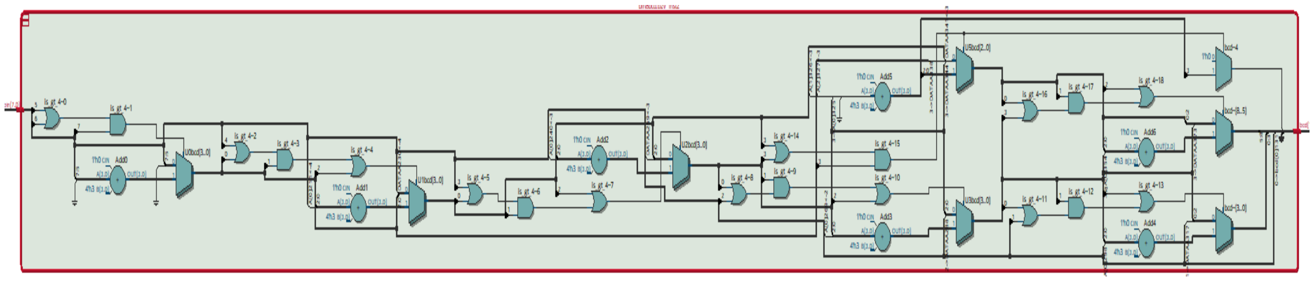
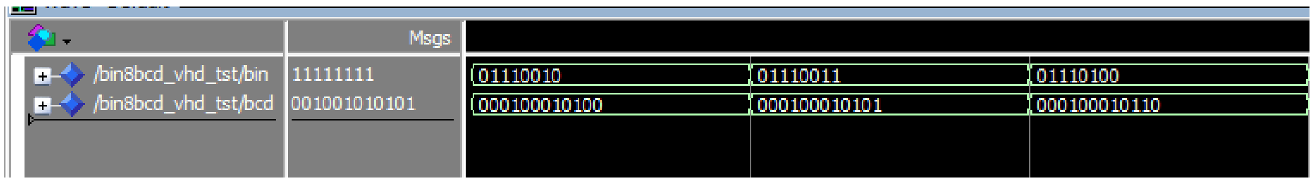


Figure: Binary to BCD converter RTL



	Msgs			
/bin8bcd_vhd_tst/bin	11111111	01110010	01110011	01110100
/bin8bcd_vhd_tst/bcd	001001010101	000100010100	000100010101	000100010110

Figure: Binary to BCD converter simulation

The converter uses a special algorithm called Double Dabble that helps in the conversion of the binary numbers into binary coded decimal numbers. What the algorithm does is first it takes the first bit of the binary number and checks if the digit is greater than 5 if its less than 5 it leaves the number as it eases and

shifts to right and checks that the numbers are greater than 5 it will add 3 to the number and the procedure continues until no digit is left.

5.6 LCD MODULE

LCD_MODULE			
External signals	Port	Width (In bits)	Description
Clock	Input	1	Clock signal (400 Hz)
Reset	Input	1	Reset
MSB	Input	12	BCD converted temperature bits
LSB	Input	1	LSB from temperature register
LCD_E	Output	1	Enable for LCD display
LCD_RS	Output	1	Register select for LCD Display
LCD_RW	Output	1	Read or Write condition select for LCD Display
LCD_BLON	Output	1	Backlight On
LCD_ON	Output	1	LCD Display On
LCD_DATA	Output	8	LCD Display bits

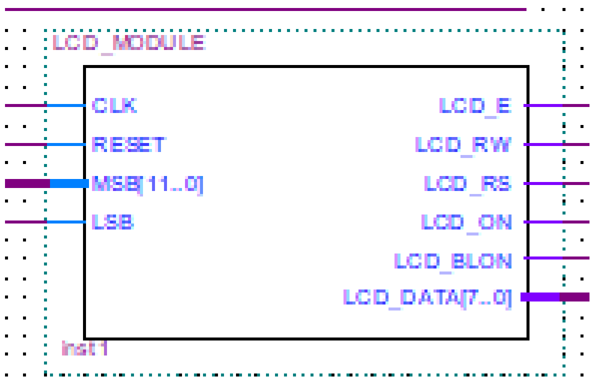


Figure: LCD Module BDF

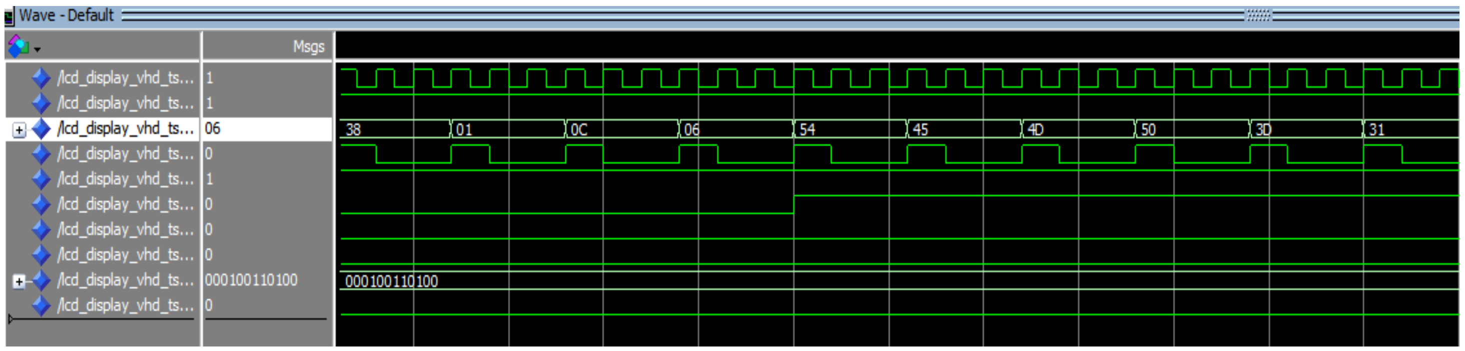


Figure LCD Display Simulation

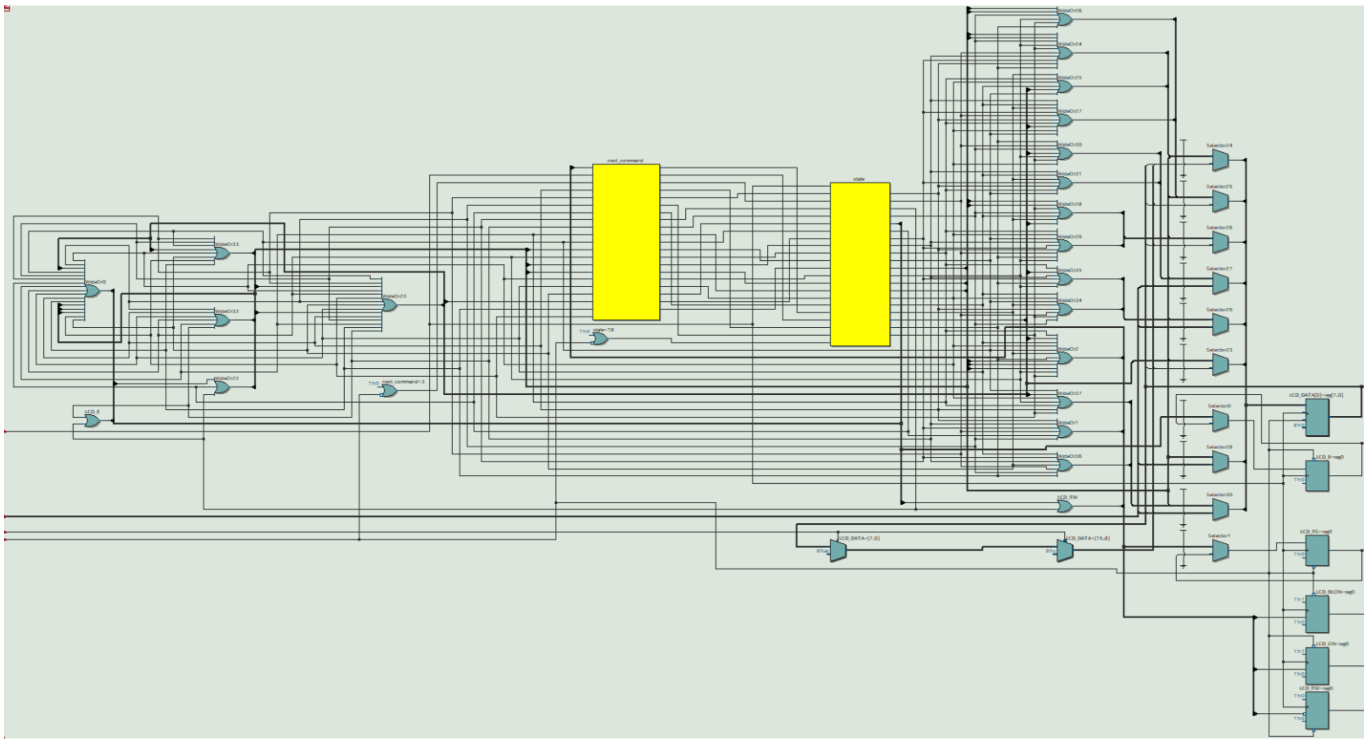


Figure: LCD Module RTL

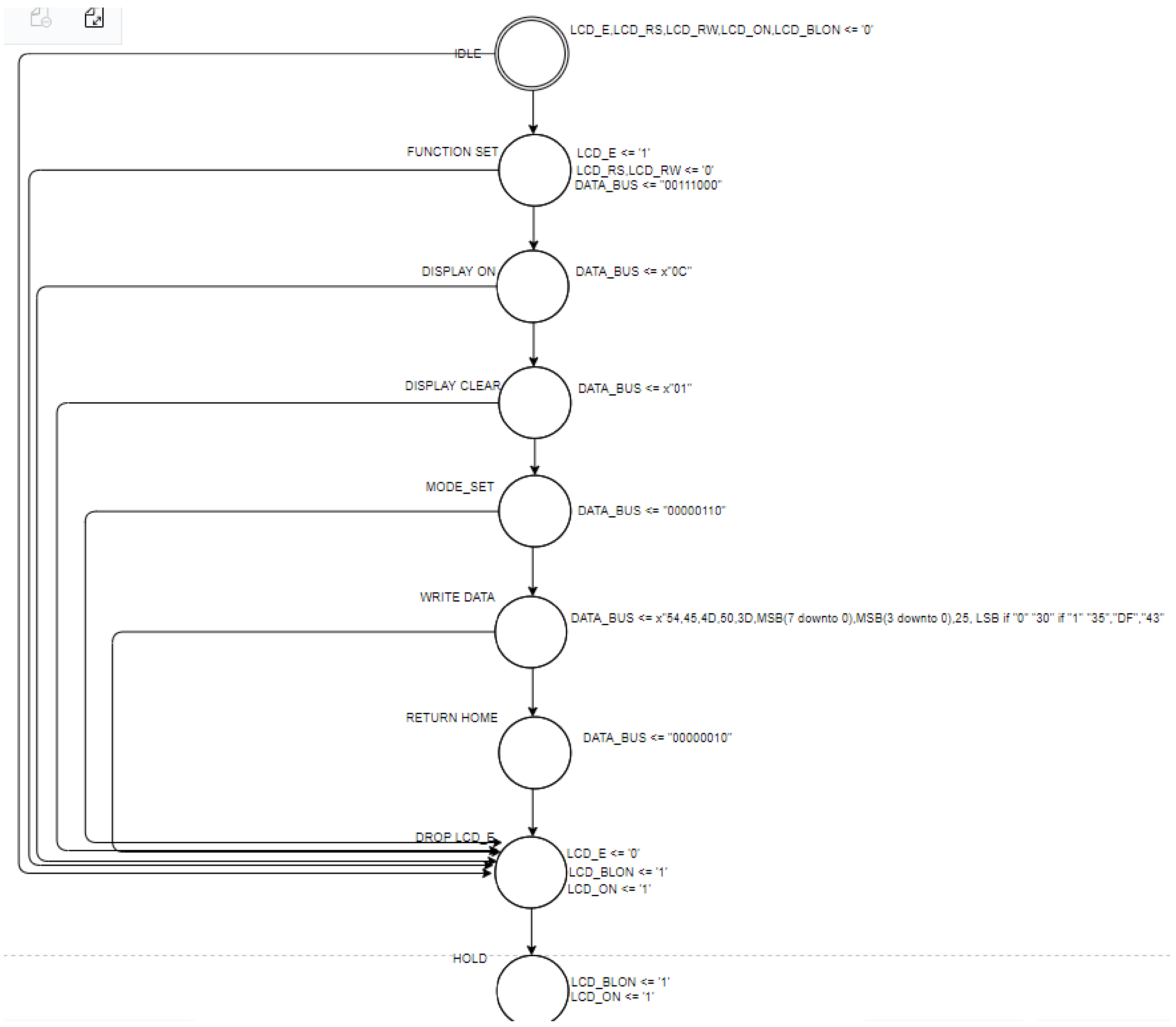


Figure: LCD state machine

5.7 COMPLETE DESIGN RTL

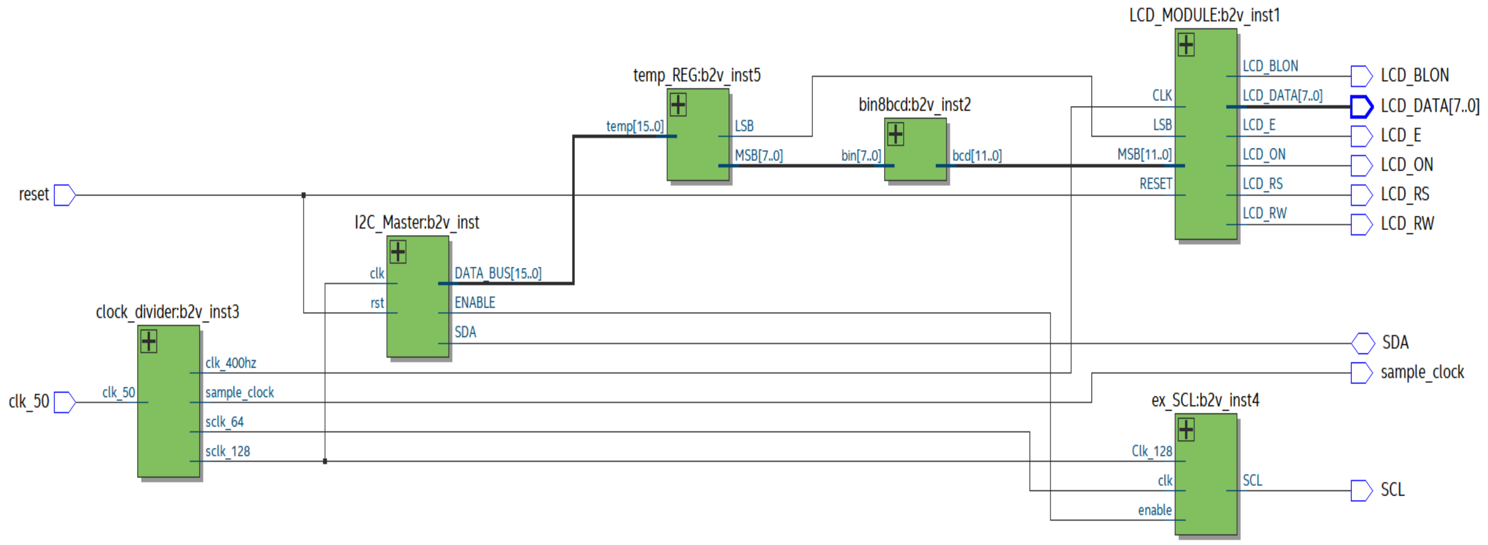


Figure: FULL DESIGN RTL

5.8 COMPLETE DESIGN BDF

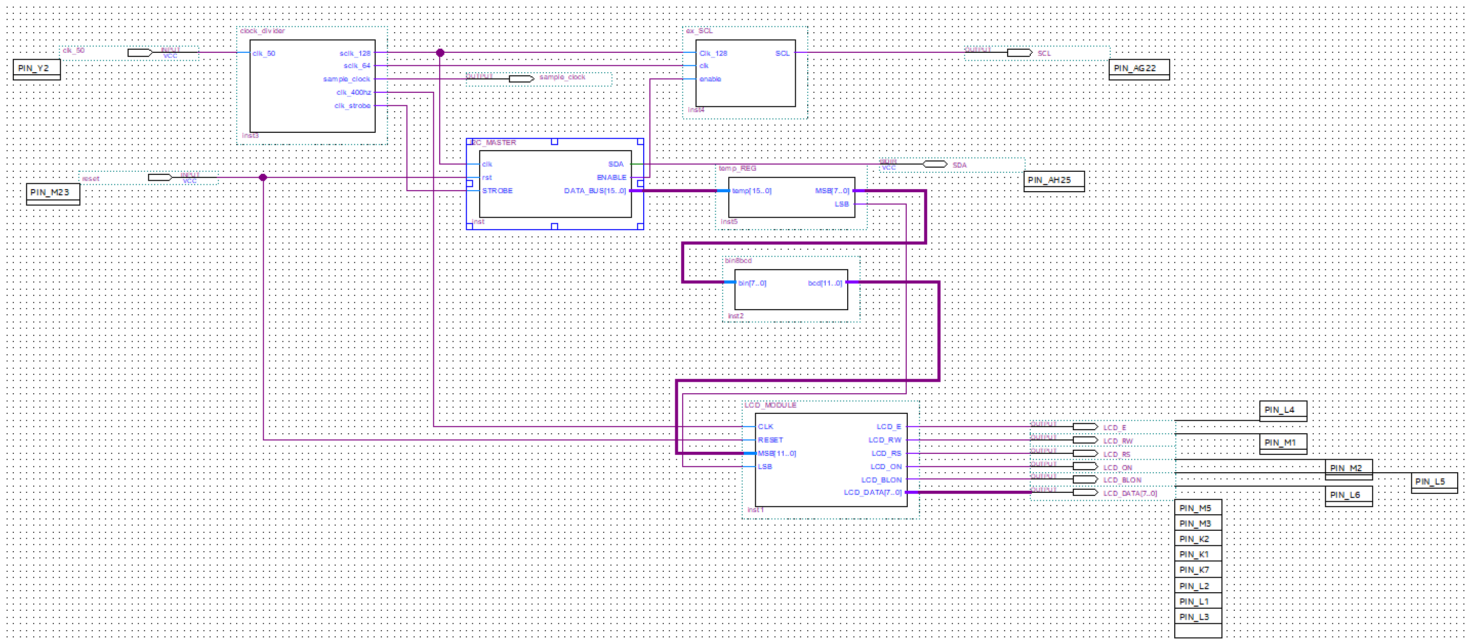


Figure: FULL DESIGN BDF

6. Temperature Logger working and Results

6.1 Temperature logger Working

The latest Temperature detected by LM75 temperature sensor is the output of the project. I2C master communicates with the slave which is temperature sensor in our case using the above-mentioned communication technique called I2C communication.

First, we create a I2C master with an external serial clock which communicates with slave using serial data line and external serial clock which is connected using an enable which is an output from the I2C master. The temperature detected is displayed on LCD display which is inbuilt in the DE2-115 board. This whole process is done using Quartus II compiler and have been coded using VHDL.

I2C master is a state machine which works according to i2c communication protocol. in I2C communication we can have multiple master and slave devices which are connected using 2 wires. Basically, the master communicates with slave by writing address of the slave to the SDA line. Slave responds to the master by giving an acknowledgement to the SDA line we call it the slave acknowledgement. After slave's response it start writing data to SDA line that master starts reading of. After reading significant data bits master send an acknowledgement and then slave sends the remaining data. When the data read is sufficient master sends a no acknowledgement which trigger the stop condition.

I2C master works on a clock signal which runs on the clock frequency of 200KHz, external serial clock runs on clock frequency of 100 KHZ and LCD display runs on clock frequency of 400Hz, but DE2-115 provides us with a clock with the frequency of 50MHz, so we use a clock divider which divides the clock and distributes the appropriate clock frequency for the respective component.

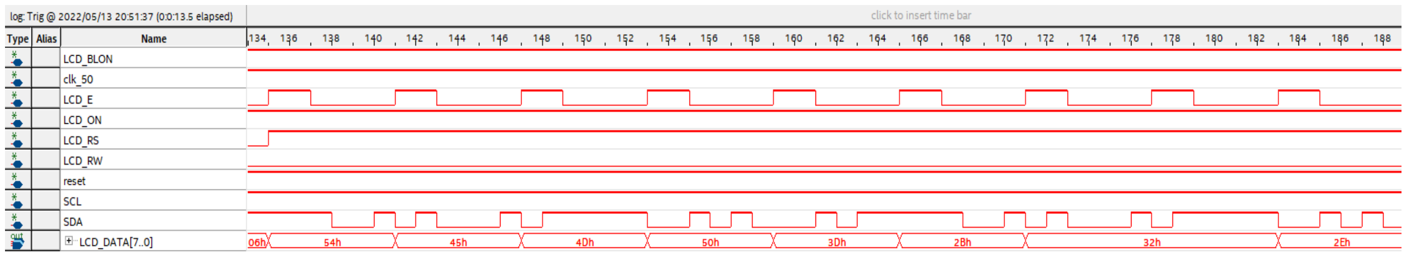
I2C master gives out a data which is 16 bits longs and the important bits for us are 9 so we have a temperature register which divides the 16 bits data into 9 bits and that 9-bit data is sent out as MSB which is first 8 bits f the 15 bits data and LSB which is 9th bits of that 16-bit data.

This MSB is a binary data, and we need binary coded decimal data for the LCD data. Therefore, we must convert this binary data to BCD. We use a binary to BCD converter which uses a special algorithm called double dabble. This double dabble is a twostep process. First, we take an 8-bit binary number, and we shift to the left and check whether the number is greater than or equal to 5 if it's not equal to 5 we shift one more bit to the left and check if the numbers are or equal to or greater than 5 if they are, then we add 0011 to the bits. we continue this process until no data bits are left. We get a BCD number as an output.

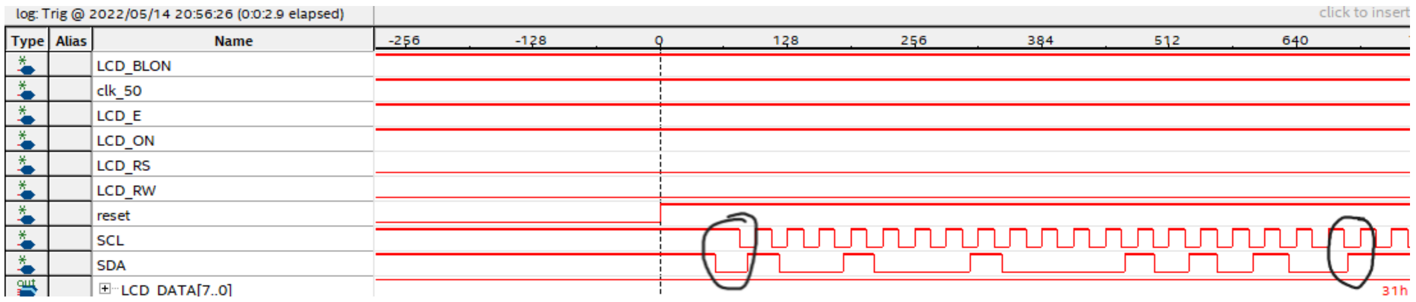
Finally, this BCD number is shown on the LCD display on the board. This BCD number is again gets converted into an ASCII code for displaying the temperature. The objective of determining the temperature as the desired outcome of the project is dependent on the first four bits, where if the 11th bit is 1 then the temperature detected is negative and if its 0 then the temperature is positive and if LSB is 0 then the granularity of the temperature is 0 and when 1 its 0.5.the whole project is implemented on Celsius scale of temperature.

6.2 RESULTS

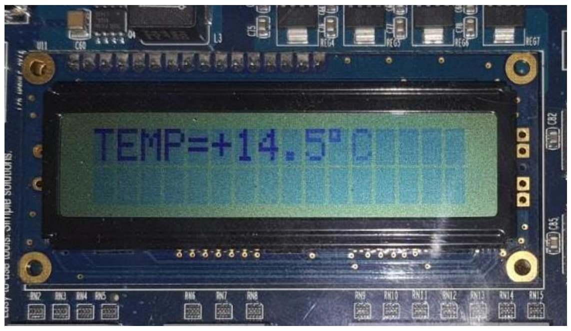
We have achieved the desired output by implementing the above-mentioned working using VHDL code and the results are shown in the below signal tap and picture of the board showing different temperatures.

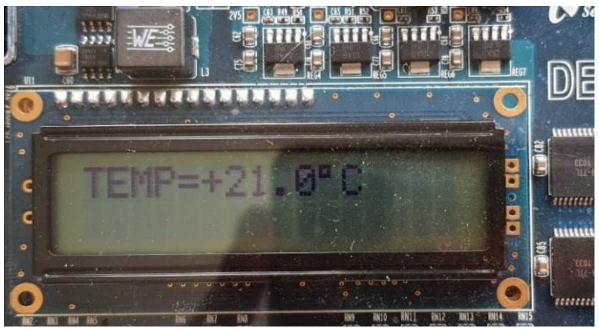


CLK_50 and SCL signals are very fast when compared to the clock of lcd display so I have made the sample clock slower for getting a reasonable wave form of the temperature logger.



Here I have introduced a much faster sample clock of frequency 1 MHz, so we got a clear SCL wave form.



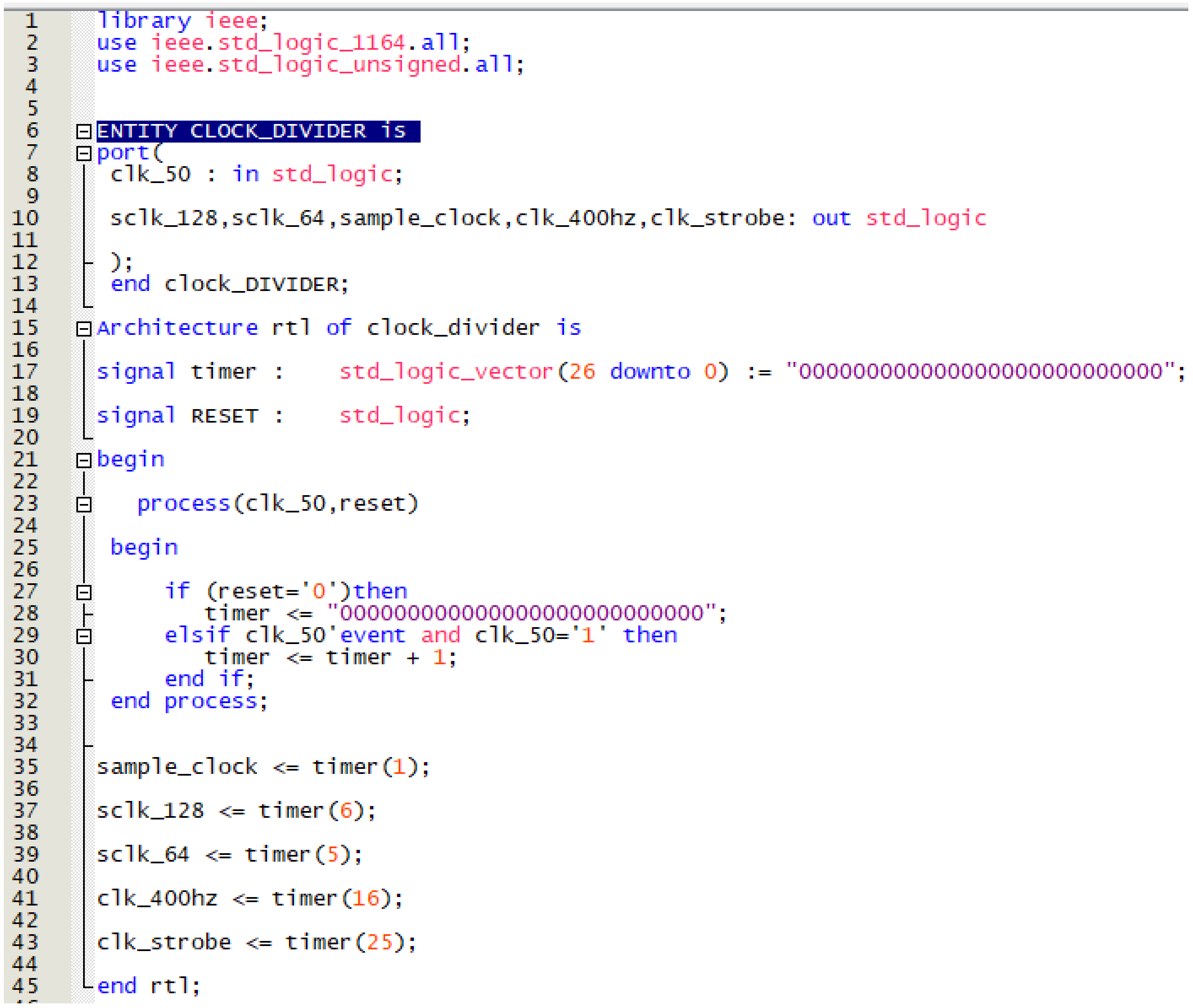


7. CONCLUSION

The objective of building a temperature logger system using temperature sensor LM75 gave results as expected which are in line with the requirements specifications with a few minor discrepancies. Understanding the concepts of I2C communication were the pivotal factors and served as an important base in building the temperature logger system. Key aspects like the communication between the DE2-115 and the temperature sensor were the fundamental building blocks of the project.

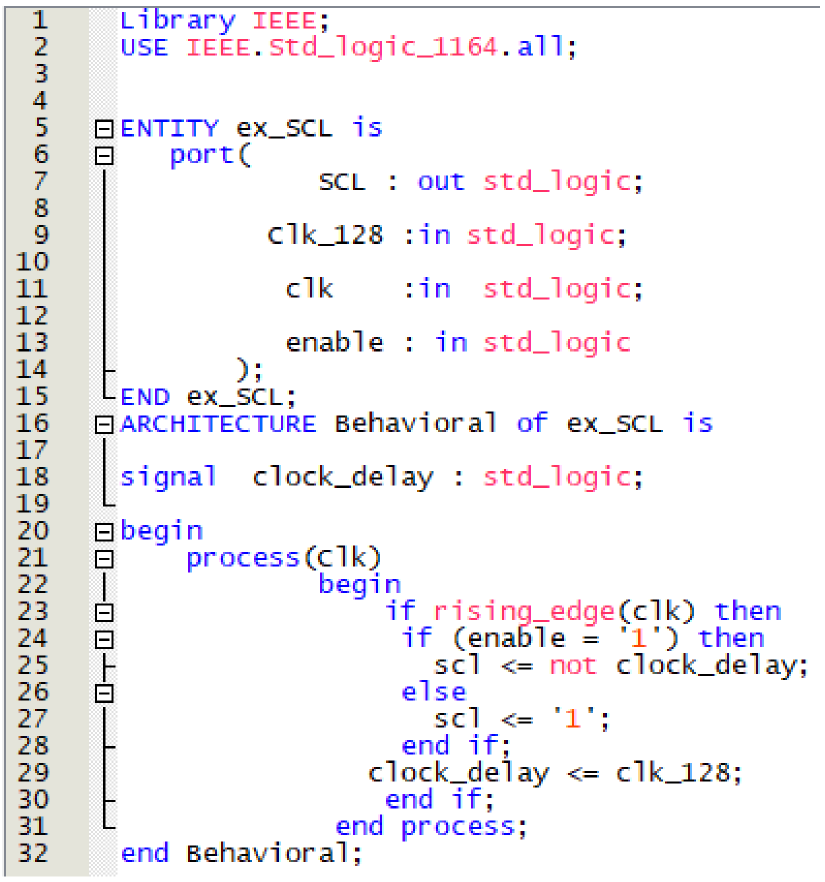
8. Appendix A: Source code

8.1 Clock divider:



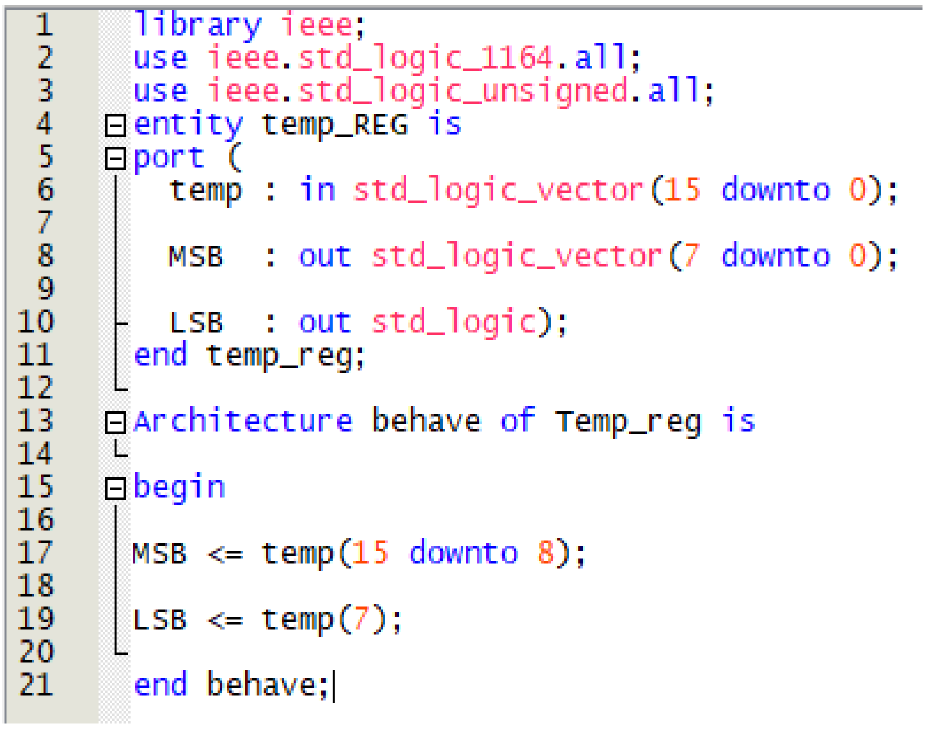
```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;


ENTITY CLOCK_DIVIDER is
port(
 clk_50 : in std_logic;

 sclk_128,sclk_64,sample_clock,clk_400hz,clk_strobe: out std_logic

 );
 end clock_DIVIDER;

Architecture rtl of clock_divider is

signal timer :    std_logic_vector(26 downto 0) := "000000000000000000000000000";

signal RESET :    std_logic;

begin

   process(clk_50,reset)

 begin

     if (reset='0')then
        timer <= "000000000000000000000000000";
     elsif clk_50'event and clk_50='1' then
        timer <= timer + 1;
     end if;
 end process;


sample_clock <= timer(1);

sclk_128 <= timer(6);

sclk_64 <= timer(5);

clk_400hz <= timer(16);

clk_strobe <= timer(25);

end rtl;
```

8.2 External Serial Clock:



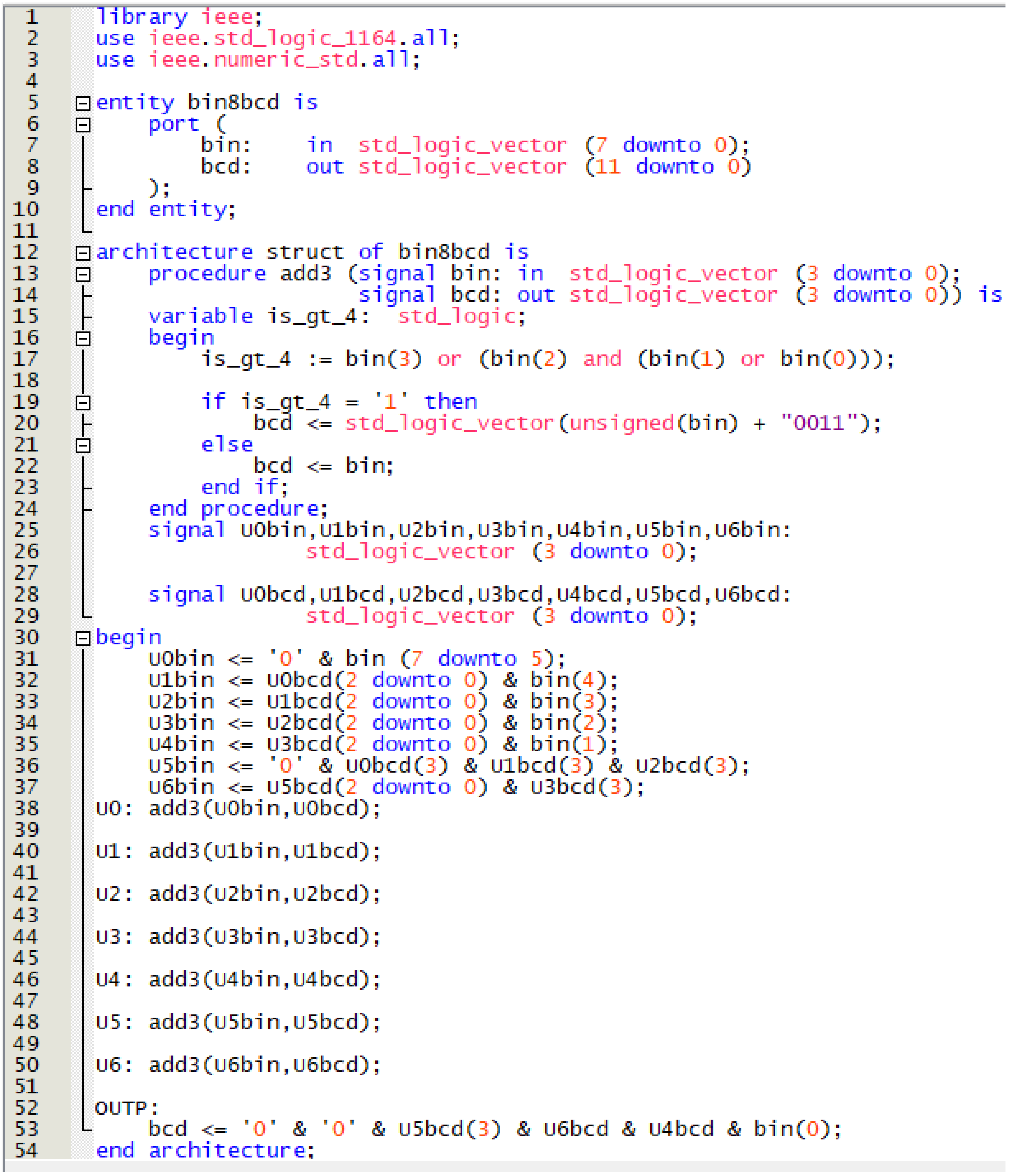
```vhdl
Library IEEE;
USE IEEE.Std_logic_1164.all;


ENTITY ex_SCL is
   port(
            SCL : out std_logic;

         clk_128 :in std_logic;

          clk     :in  std_logic;

          enable : in std_logic
       );
END ex_SCL;
ARCHITECTURE Behavioral of ex_SCL is

signal  clock_delay : std_logic;

begin
    process(clk)
            begin
                if rising_edge(clk) then
                 if (enable = '1') then
                   scl <= not clock_delay;
                 else
                   scl <= '1';
                 end if;
               clock_delay <= clk_128;
                end if;
             end process;
end Behavioral;
```

8.3 Temperature Register:



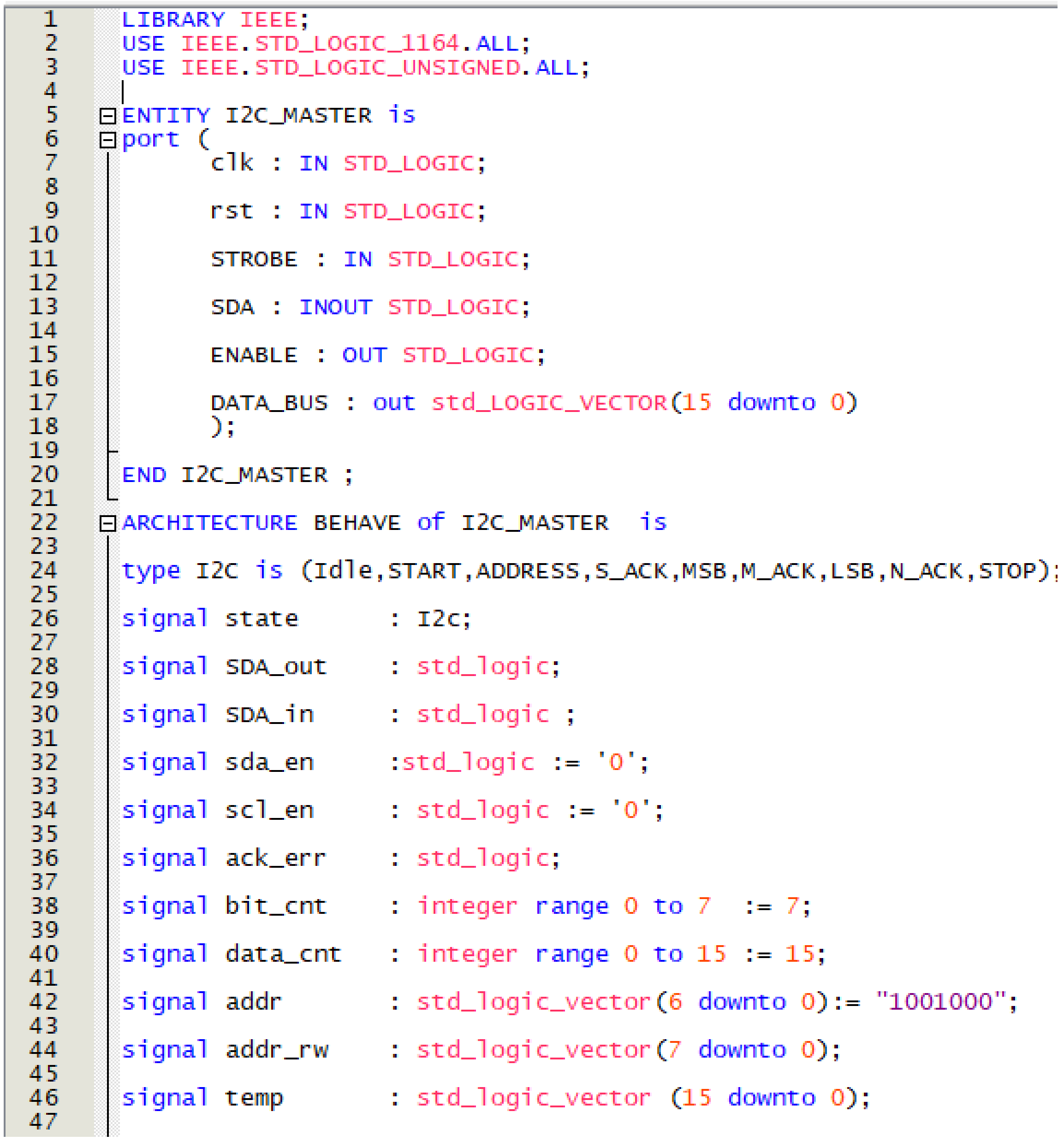
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity temp_REG is
port (
   temp : in std_logic_vector(15 downto 0);

   MSB  : out std_logic_vector(7 downto 0);

   LSB  : out std_logic);
end temp_reg;

Architecture behave of Temp_reg is

begin

MSB <= temp(15 downto 8);

LSB <= temp(7);

end behave;
```

8.4 Binary to BCD converter:

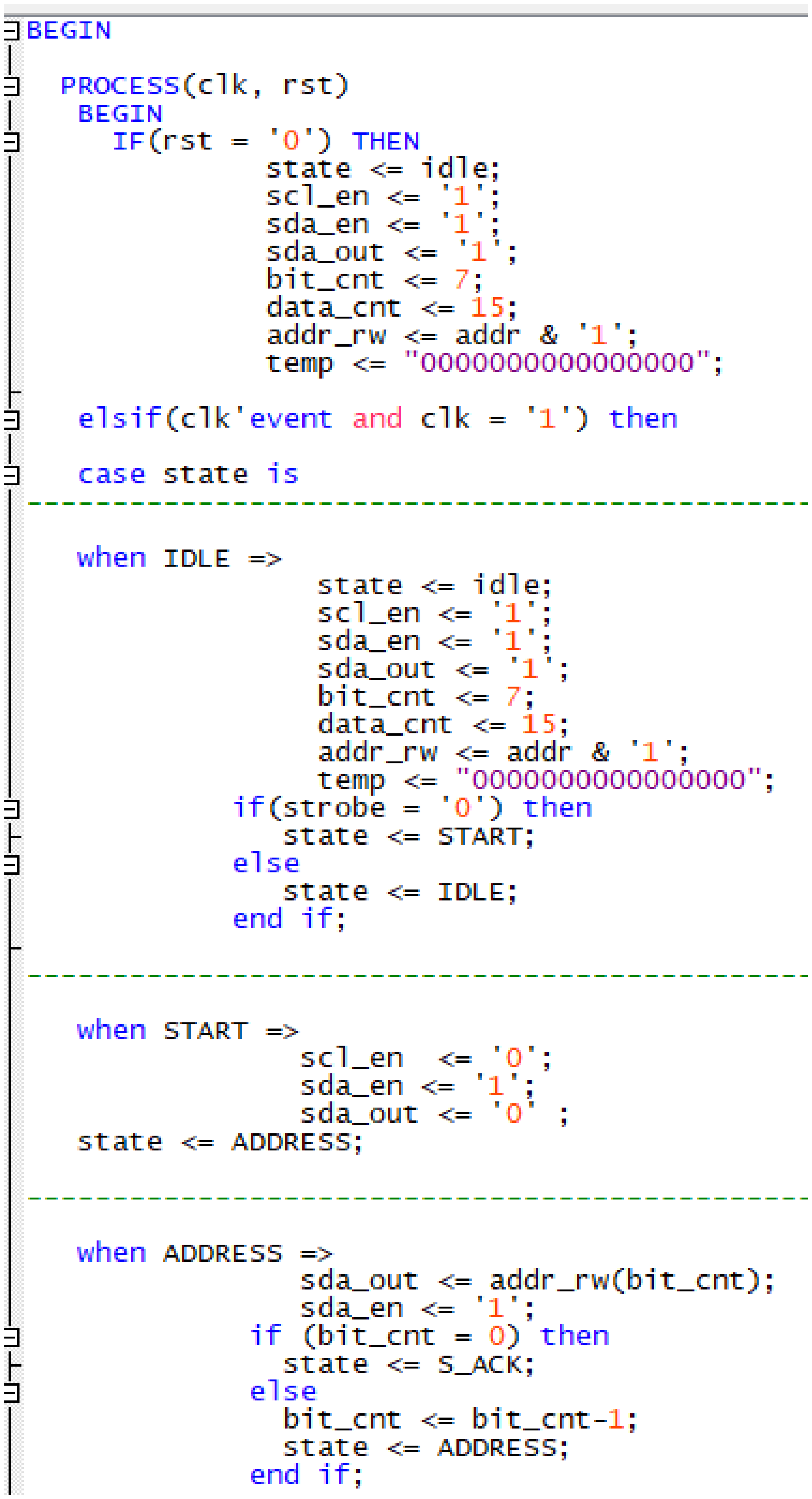


```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity bin8bcd is
    port (
        bin:    in  std_logic_vector (7 downto 0);
        bcd:    out std_logic_vector (11 downto 0)
    );
end entity;

architecture struct of bin8bcd is
    procedure add3 (signal bin: in  std_logic_vector (3 downto 0);
                    signal bcd: out std_logic_vector (3 downto 0)) is
    variable is_gt_4:  std_logic;
    begin
        is_gt_4 := bin(3) or (bin(2) and (bin(1) or bin(0)));

        if is_gt_4 = '1' then
            bcd <= std_logic_vector(unsigned(bin) + "0011");
        else
            bcd <= bin;
        end if;
    end procedure;
    signal U0bin,U1bin,U2bin,U3bin,U4bin,U5bin,U6bin:
                std_logic_vector (3 downto 0);

    signal U0bcd,U1bcd,U2bcd,U3bcd,U4bcd,U5bcd,U6bcd:
                std_logic_vector (3 downto 0);
begin
    U0bin <= '0' & bin (7 downto 5);
    U1bin <= U0bcd(2 downto 0) & bin(4);
    U2bin <= U1bcd(2 downto 0) & bin(3);
    U3bin <= U2bcd(2 downto 0) & bin(2);
    U4bin <= U3bcd(2 downto 0) & bin(1);
    U5bin <= '0' & U0bcd(3) & U1bcd(3) & U2bcd(3);
    U6bin <= U5bcd(2 downto 0) & U3bcd(3);
U0: add3(U0bin,U0bcd);

U1: add3(U1bin,U1bcd);

U2: add3(U2bin,U2bcd);

U3: add3(U3bin,U3bcd);

U4: add3(U4bin,U4bcd);

U5: add3(U5bin,U5bcd);

U6: add3(U6bin,U6bcd);

OUTP:
    bcd <= '0' & '0' & U5bcd(3) & U6bcd & U4bcd & bin(0);
end architecture;
```

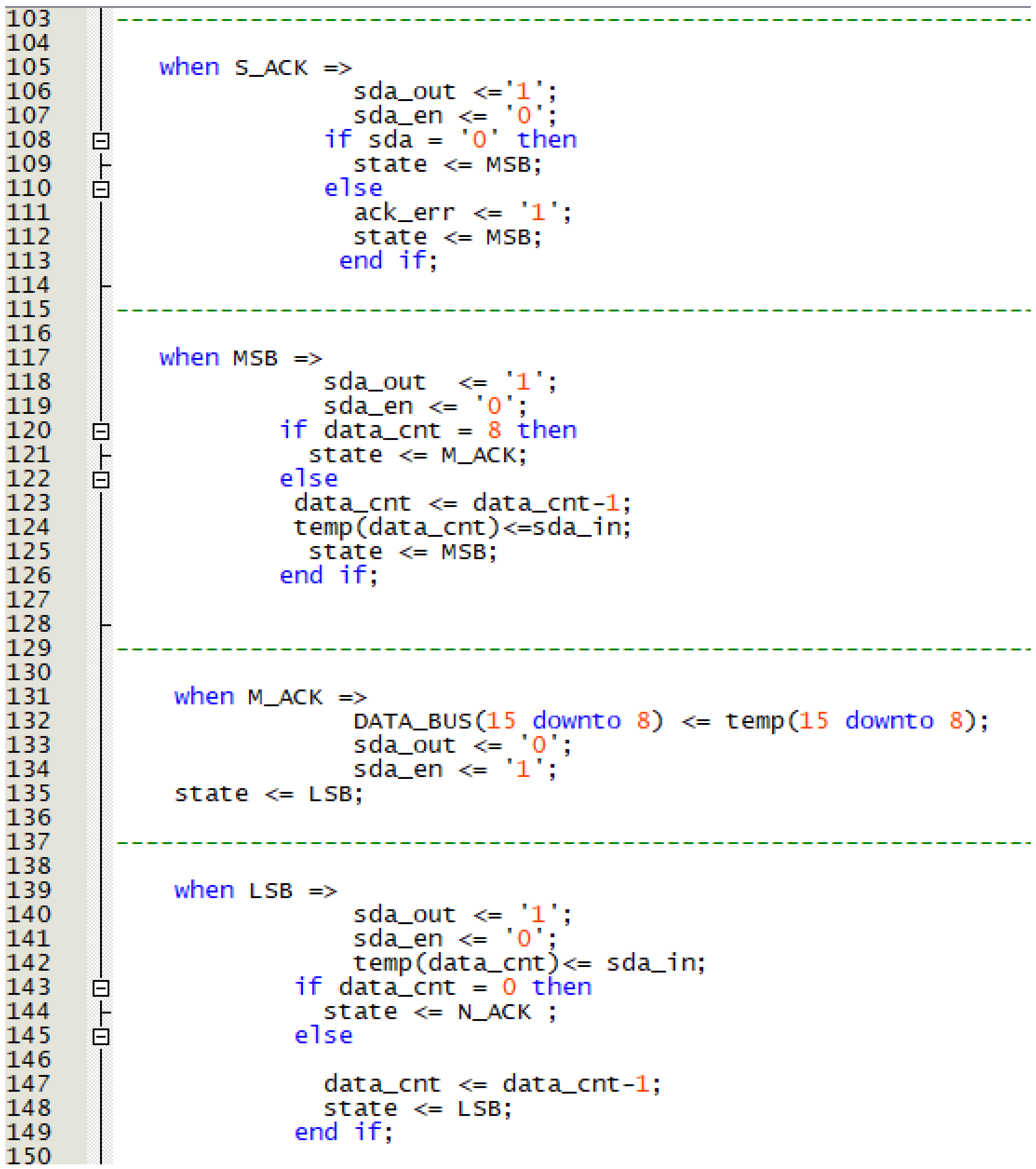
8.5 I2C Master



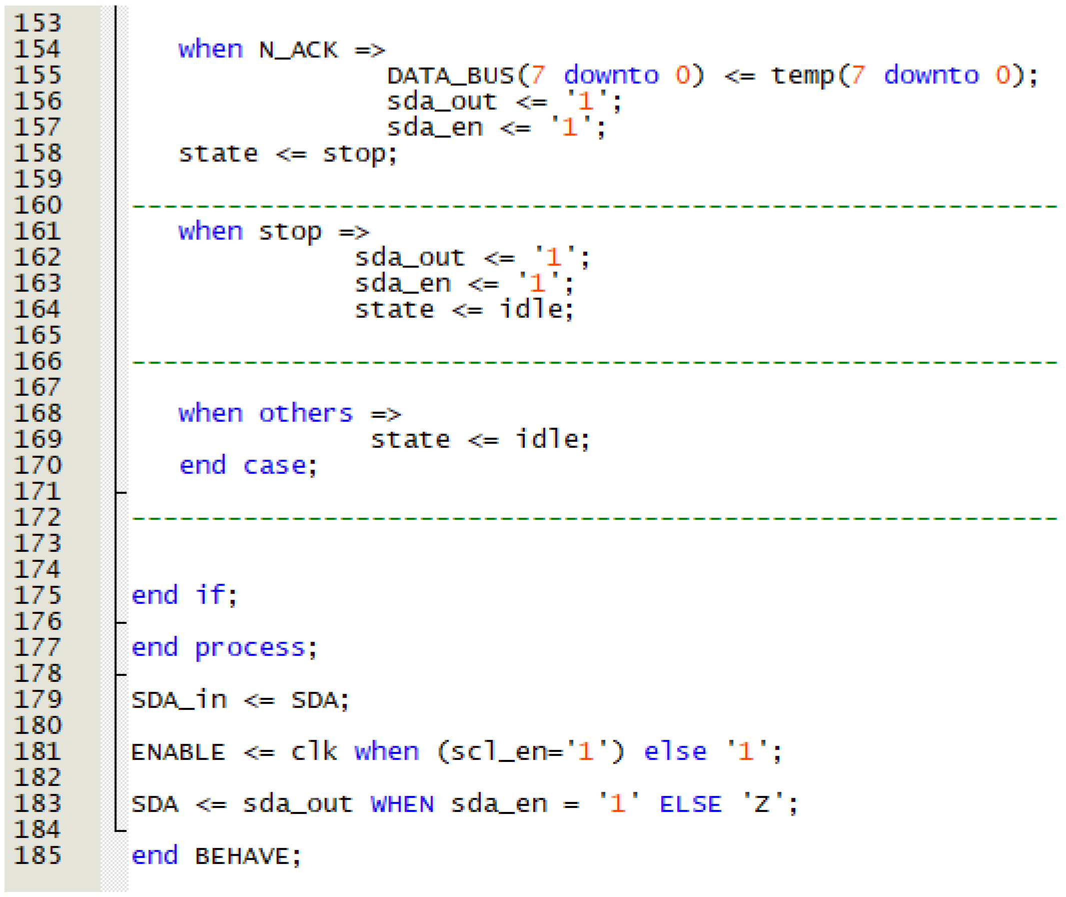
```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY I2C_MASTER is
port (
        clk : IN STD_LOGIC;

        rst : IN STD_LOGIC;

        STROBE : IN STD_LOGIC;

        SDA : INOUT STD_LOGIC;

        ENABLE : OUT STD_LOGIC;

        DATA_BUS : out std_LOGIC_VECTOR(15 downto 0)
        );

END I2C_MASTER ;

ARCHITECTURE BEHAVE of I2C_MASTER  is

type I2C is (Idle,START,ADDRESS,S_ACK,MSB,M_ACK,LSB,N_ACK,STOP);

signal state      : I2c;

signal SDA_out    : std_logic;

signal SDA_in     : std_logic ;

signal sda_en     :std_logic := '0';

signal scl_en     : std_logic := '0';

signal ack_err    : std_logic;

signal bit_cnt    : integer range 0 to 7  := 7;

signal data_cnt   : integer range 0 to 15 := 15;

signal addr       : std_logic_vector(6 downto 0):= "1001000";

signal addr_rw    : std_logic_vector(7 downto 0);

signal temp       : std_logic_vector (15 downto 0);

```



```vhdl
BEGIN

  PROCESS(clk, rst)
   BEGIN
     IF(rst = '0') THEN
              state <= idle;
              scl_en <= '1';
              sda_en <= '1';
              sda_out <= '1';
              bit_cnt <= 7;
              data_cnt <= 15;
              addr_rw <= addr & '1';
              temp <= "0000000000000000";

   elsif(clk'event and clk = '1') then

   case state is
----------------------------------------------

   when IDLE =>
                 state <= idle;
                 scl_en <= '1';
                 sda_en <= '1';
                 sda_out <= '1';
                 bit_cnt <= 7;
                 data_cnt <= 15;
                 addr_rw <= addr & '1';
                 temp <= "0000000000000000";
            if(strobe = '0') then
               state <= START;
            else
               state <= IDLE;
            end if;

----------------------------------------------

   when START =>
                scl_en  <= '0';
                sda_en <= '1';
                sda_out <= '0' ;
   state <= ADDRESS;

----------------------------------------------

   when ADDRESS =>
                sda_out <= addr_rw(bit_cnt);
                sda_en <= '1';
             if (bit_cnt = 0) then
               state <= S_ACK;
             else
               bit_cnt <= bit_cnt-1;
               state <= ADDRESS;
             end if;
```

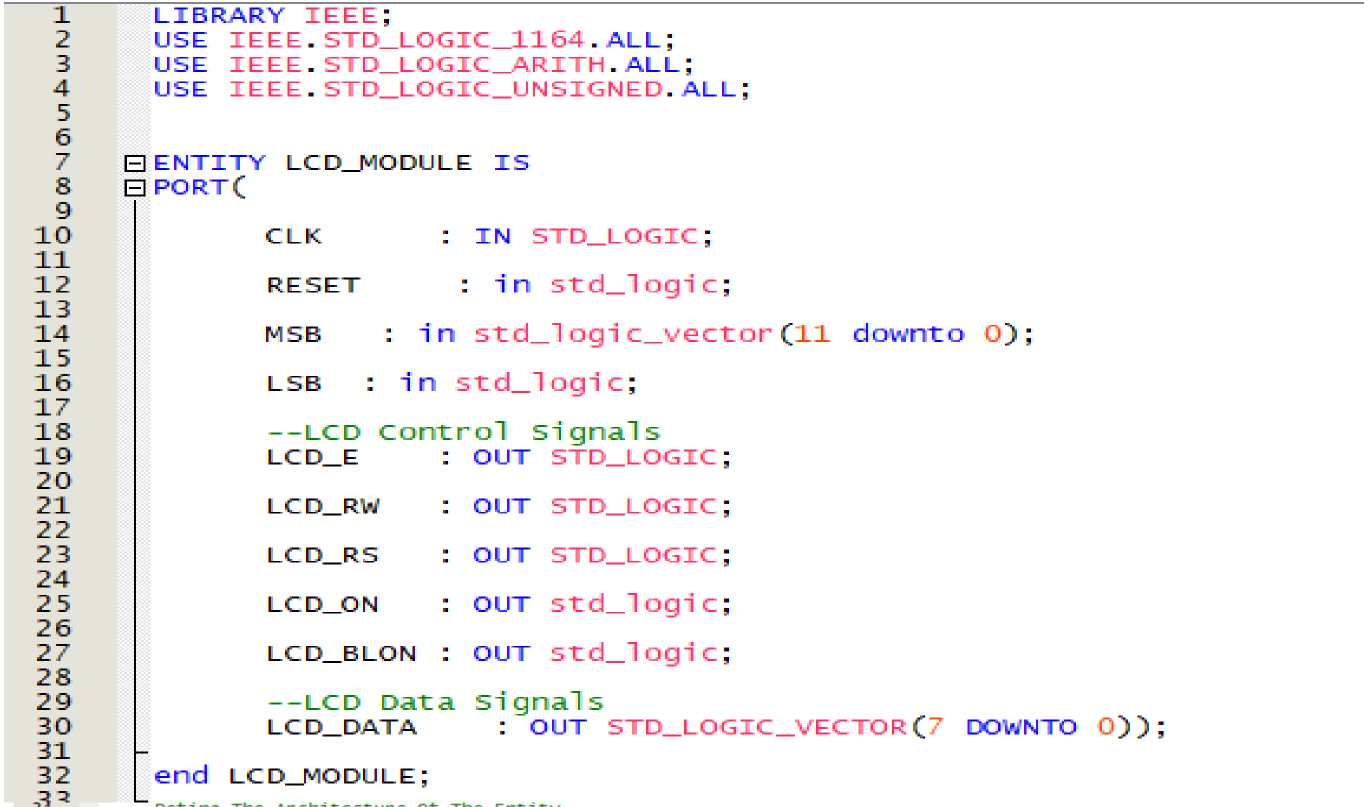


```vhdl
--------------------------------------------------------------

   when S_ACK =>
                sda_out <='1';
                sda_en <= '0';
              if sda = '0' then
                state <= MSB;
              else
                ack_err <= '1';
                state <= MSB;
               end if;

--------------------------------------------------------------

   when MSB =>
              sda_out   <= '1';
              sda_en <= '0';
           if data_cnt = 8 then
             state <= M_ACK;
           else
            data_cnt <= data_cnt-1;
            temp(data_cnt)<=sda_in;
             state <= MSB;
           end if;


--------------------------------------------------------------

    when M_ACK =>
                DATA_BUS(15 downto 8) <= temp(15 downto 8);
                sda_out <= '0';
                sda_en <= '1';
    state <= LSB;

--------------------------------------------------------------

    when LSB =>
                sda_out <= '1';
                sda_en <= '0';
                temp(data_cnt)<= sda_in;
            if data_cnt = 0 then
              state <= N_ACK ;
            else

              data_cnt <= data_cnt-1;
              state <= LSB;
            end if;

```

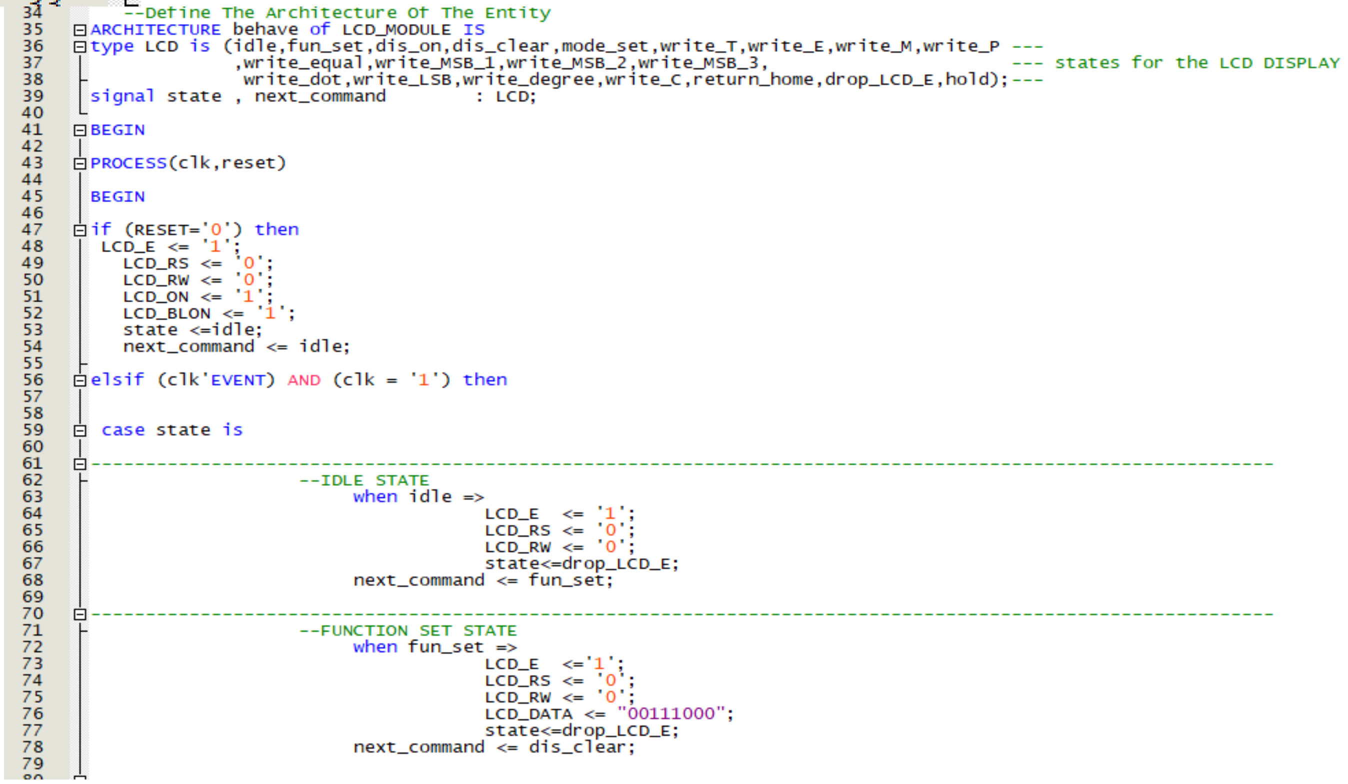


```vhdl

   when N_ACK =>
                DATA_BUS(7 downto 0) <= temp(7 downto 0);
                sda_out <= '1';
                sda_en <= '1';
   state <= stop;

----------------------------------------------------------
   when stop =>
              sda_out <= '1';
              sda_en <= '1';
              state <= idle;

----------------------------------------------------------

   when others =>
               state <= idle;
   end case;

----------------------------------------------------------


end if;

end process;

SDA_in <= SDA;

ENABLE <= clk when (scl_en='1') else '1';

SDA <= sda_out WHEN sda_en = '1' ELSE 'Z';

end BEHAVE;
```

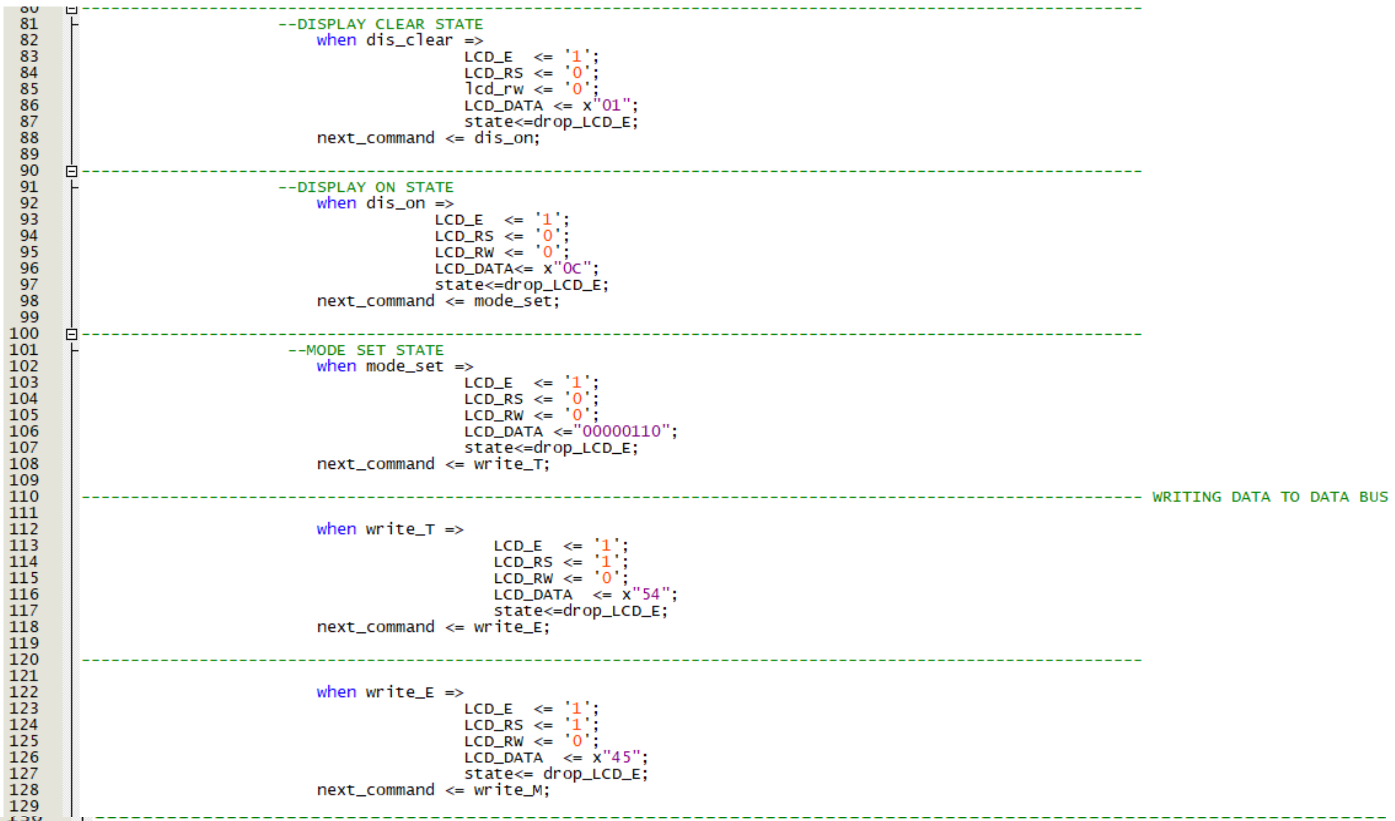
8.6 LCD Module:



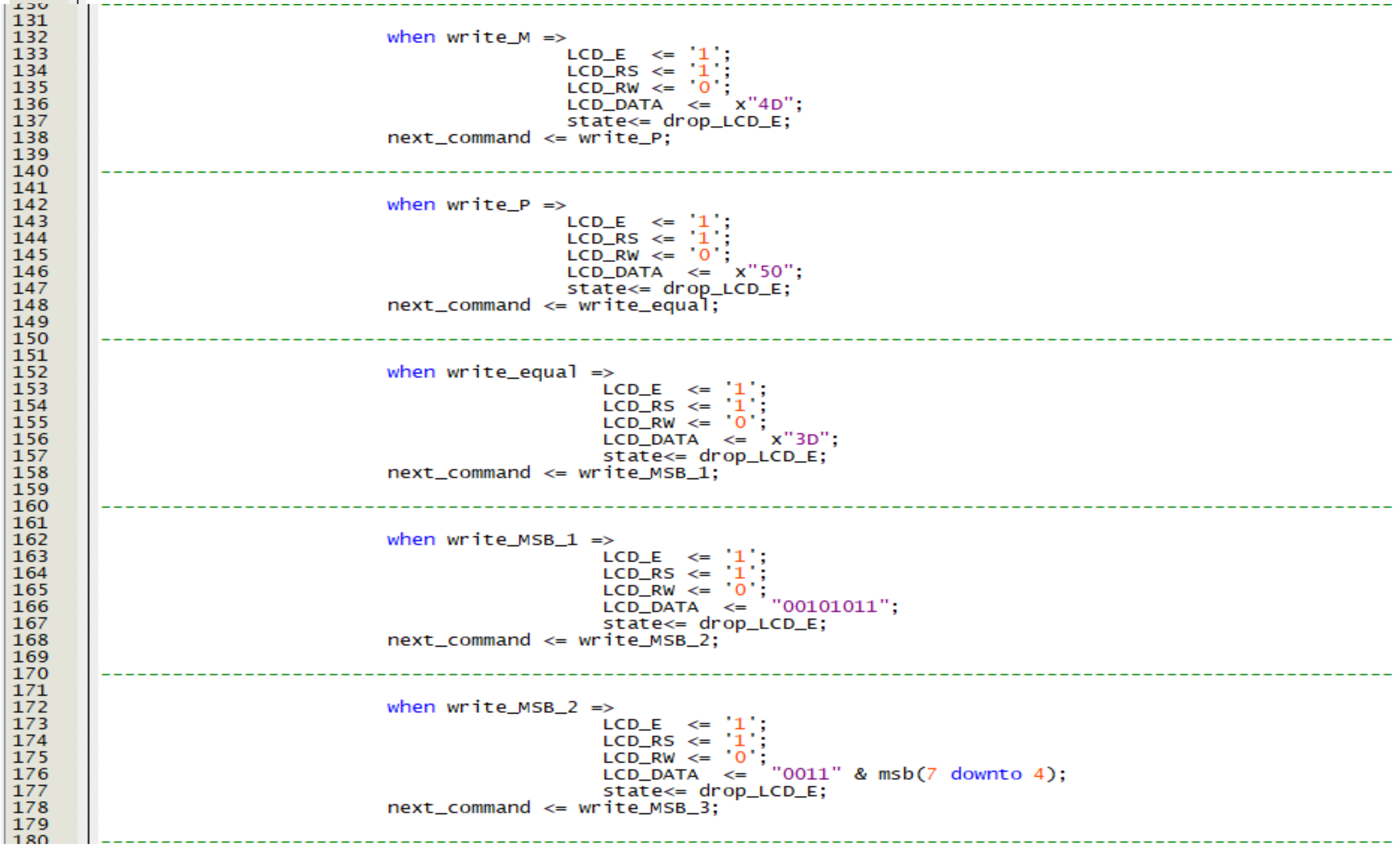
```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;


ENTITY LCD_MODULE IS
PORT(

        CLK        : IN STD_LOGIC;

        RESET        : in std_logic;

        MSB    : in std_logic_vector(11 downto 0);

        LSB   : in std_logic;

        --LCD Control Signals
        LCD_E      : OUT STD_LOGIC;

        LCD_RW     : OUT STD_LOGIC;

        LCD_RS     : OUT STD_LOGIC;

        LCD_ON     : OUT std_logic;

        LCD_BLON : OUT std_logic;

        --LCD Data Signals
        LCD_DATA      : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));

end LCD_MODULE;
```



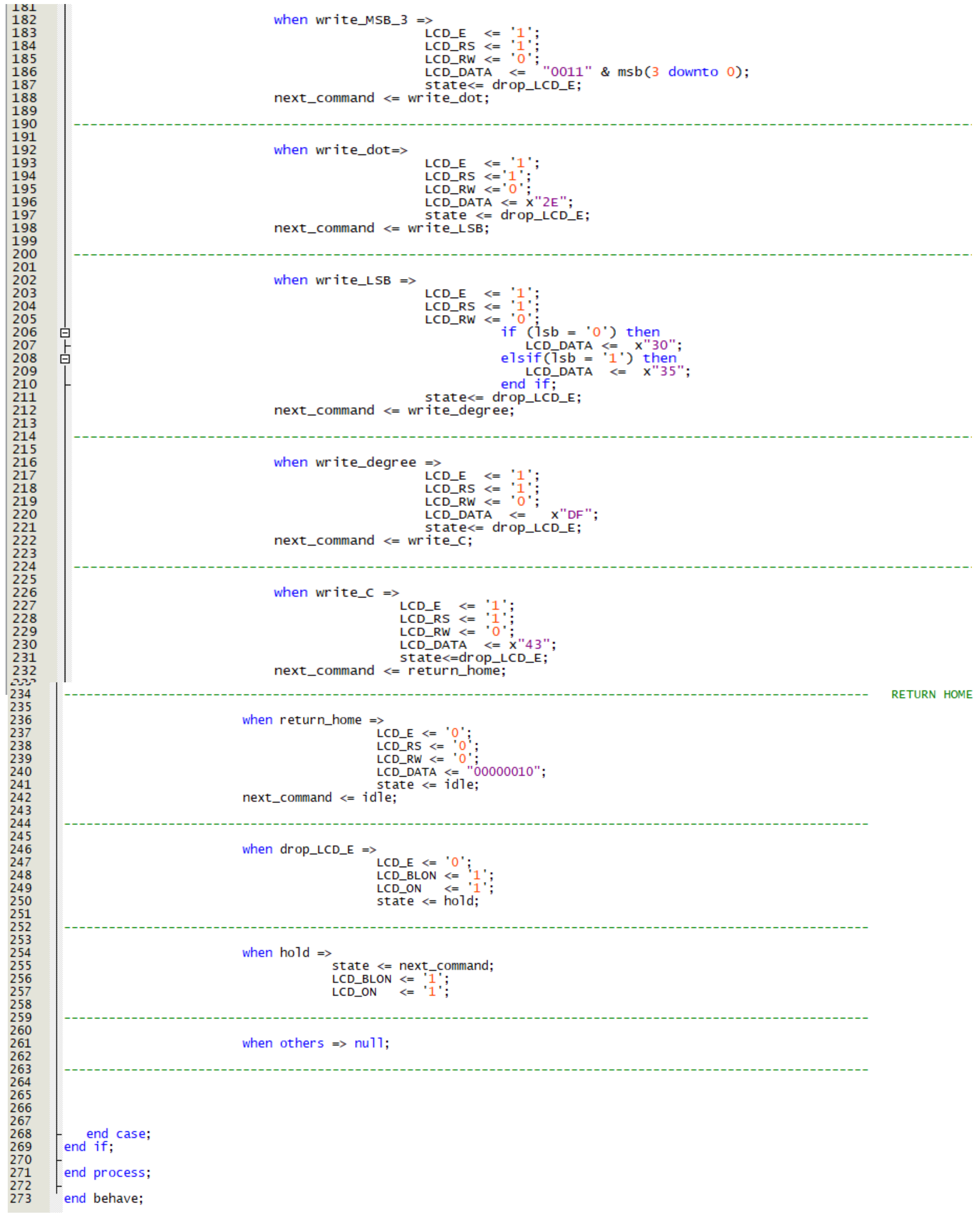
```vhdl
    --Define The Architecture Of The Entity
ARCHITECTURE behave of LCD_MODULE IS
type LCD is (idle,fun_set,dis_on,dis_clear,mode_set,write_T,write_E,write_M,write_P ---
             ,write_equal,write_MSB_1,write_MSB_2,write_MSB_3,                    --- states for the LCD DISPLAY
              write_dot,write_LSB,write_degree,write_C,return_home,drop_LCD_E,hold);---
signal state , next_command         : LCD;

BEGIN

PROCESS(clk,reset)

BEGIN

if (RESET='0') then
 LCD_E <= '1';
   LCD_RS <= '0';
   LCD_RW <= '0';
   LCD_ON <= '1';
   LCD_BLON <= '1';
   state <=idle;
   next_command <= idle;

elsif (clk'EVENT) AND (clk = '1') then


 case state is

-------------------------------------------------------------------------------------------------------------
                   --IDLE STATE
                        when idle =>
                                    LCD_E   <= '1';
                                    LCD_RS <= '0';
                                    LCD_RW <= '0';
                                    state<=drop_LCD_E;
                        next_command <= fun_set;

-------------------------------------------------------------------------------------------------------------
                   --FUNCTION SET STATE
                        when fun_set =>
                                    LCD_E   <='1';
                                    LCD_RS <= '0';
                                    LCD_RW <= '0';
                                    LCD_DATA <= "00111000";
                                    state<=drop_LCD_E;
                        next_command <= dis_clear;
```



```
---------------------------------------------------------------------------------------------------------
                    --DISPLAY CLEAR STATE
                        when dis_clear =>
                                        LCD_E  <= '1';
                                        LCD_RS <= '0';
                                        lcd_rw <= '0';
                                        LCD_DATA <= x"01";
                                        state<=drop_LCD_E;
                        next_command <= dis_on;

---------------------------------------------------------------------------------------------------------
                    --DISPLAY ON STATE
                        when dis_on =>
                                    LCD_E  <= '1';
                                    LCD_RS <= '0';
                                    LCD_RW <= '0';
                                    LCD_DATA<= x"0C";
                                    state<=drop_LCD_E;
                        next_command <= mode_set;

---------------------------------------------------------------------------------------------------------
                     --MODE SET STATE
                        when mode_set =>
                                        LCD_E  <= '1';
                                        LCD_RS <= '0';
                                        LCD_RW <= '0';
                                        LCD_DATA <="00000110";
                                        state<=drop_LCD_E;
                        next_command <= write_T;

--------------------------------------------------------------------------------------------------------- WRITING DATA TO DATA BUS

                        when write_T =>
                                           LCD_E  <= '1';
                                           LCD_RS <= '1';
                                           LCD_RW <= '0';
                                           LCD_DATA  <= x"54";
                                           state<=drop_LCD_E;
                        next_command <= write_E;

---------------------------------------------------------------------------------------------------------

                        when write_E =>
                                        LCD_E  <= '1';
                                        LCD_RS <= '1';
                                        LCD_RW <= '0';
                                        LCD_DATA  <= x"45";
                                        state<= drop_LCD_E;
                        next_command <= write_M;
```



```
---------------------------------------------------------------------------------------------------------

                        when write_M =>
                                        LCD_E  <= '1';
                                        LCD_RS <= '1';
                                        LCD_RW <= '0';
                                        LCD_DATA   <=  x"4D";
                                        state<= drop_LCD_E;
                        next_command <= write_P;

---------------------------------------------------------------------------------------------------------

                        when write_P =>
                                        LCD_E  <= '1';
                                        LCD_RS <= '1';
                                        LCD_RW <= '0';
                                        LCD_DATA   <=  x"50";
                                        state<= drop_LCD_E;
                        next_command <= write_equal;

---------------------------------------------------------------------------------------------------------

                        when write_equal =>
                                          LCD_E  <= '1';
                                          LCD_RS <= '1';
                                          LCD_RW <= '0';
                                          LCD_DATA   <=  x"3D";
                                          state<= drop_LCD_E;
                        next_command <= write_MSB_1;

---------------------------------------------------------------------------------------------------------

                        when write_MSB_1 =>
                                          LCD_E  <= '1';
                                          LCD_RS <= '1';
                                          LCD_RW <= '0';
                                          LCD_DATA   <=  "00101011";
                                          state<= drop_LCD_E;
                        next_command <= write_MSB_2;

---------------------------------------------------------------------------------------------------------

                        when write_MSB_2 =>
                                          LCD_E  <= '1';
                                          LCD_RS <= '1';
                                          LCD_RW <= '0';
                                          LCD_DATA   <=  "0011" & msb(7 downto 4);
                                          state<= drop_LCD_E;
                        next_command <= write_MSB_3;

---------------------------------------------------------------------------------------------------------
```



```vhdl
                    when write_MSB_3 =>
                                    LCD_E   <= '1';
                                    LCD_RS  <= '1';
                                    LCD_RW  <= '0';
                                    LCD_DATA   <=  "0011" & msb(3 downto 0);
                                    state<= drop_LCD_E;
                    next_command <= write_dot;

------------------------------------------------------------------------------------------------------------

                    when write_dot=>
                                    LCD_E   <= '1';
                                    LCD_RS <='1';
                                    LCD_RW <='0';
                                    LCD_DATA <= x"2E";
                                    state <= drop_LCD_E;
                    next_command <= write_LSB;

------------------------------------------------------------------------------------------------------------

                    when write_LSB =>
                                    LCD_E   <= '1';
                                    LCD_RS  <= '1';
                                    LCD_RW  <= '0';
                                             if (lsb = '0') then
                                                LCD_DATA <=  x"30";
                                             elsif(lsb = '1') then
                                                LCD_DATA   <=  x"35";
                                             end if;
                                    state<= drop_LCD_E;
                    next_command <= write_degree;

------------------------------------------------------------------------------------------------------------

                    when write_degree =>
                                    LCD_E   <= '1';
                                    LCD_RS  <= '1';
                                    LCD_RW  <= '0';
                                    LCD_DATA   <=   x"DF";
                                    state<= drop_LCD_E;
                    next_command <= write_C;

------------------------------------------------------------------------------------------------------------

                    when write_C =>
                                 LCD_E   <= '1';
                                 LCD_RS  <= '1';
                                 LCD_RW  <= '0';
                                 LCD_DATA   <= x"43";
                                 state<=drop_LCD_E;
                    next_command <= return_home;
------------------------------------------------------------------------------------------------------------   RETURN HOME

                    when return_home =>
                                    LCD_E <= '0';
                                    LCD_RS <= '0';
                                    LCD_RW <= '0';
                                    LCD_DATA <= "00000010";
                                    state <= idle;
                    next_command <= idle;

------------------------------------------------------------------------------------------------------------

                    when drop_LCD_E =>
                                    LCD_E <= '0';
                                    LCD_BLON <= '1';
                                    LCD_ON    <= '1';
                                    state <= hold;

------------------------------------------------------------------------------------------------------------

                    when hold =>
                               state <= next_command;
                               LCD_BLON <= '1';
                               LCD_ON   <= '1';

------------------------------------------------------------------------------------------------------------

                    when others => null;

------------------------------------------------------------------------------------------------------------



    end case;
end if;

end process;

end behave;
```

9. REFERENCES

1. VHDL for designers, book written by Stefan Sjoholm and Lennart Lindh
2. For clock divider: https://www.youtube.com/watch?v=GYj_G6KVP1Y&t=124s
3. For I2C Master: Material provided by the faculty during beginning of the project
4. For Binary to BCD converter: https://www.youtube.com/watch?v=yki0PkBcJbg&t=182s
5. For LCD display: https://www.youtube.com/watch?v=z9PQb7XoceM&t=1093s
6. For rest of registers and debugging: Labs performed during the course .