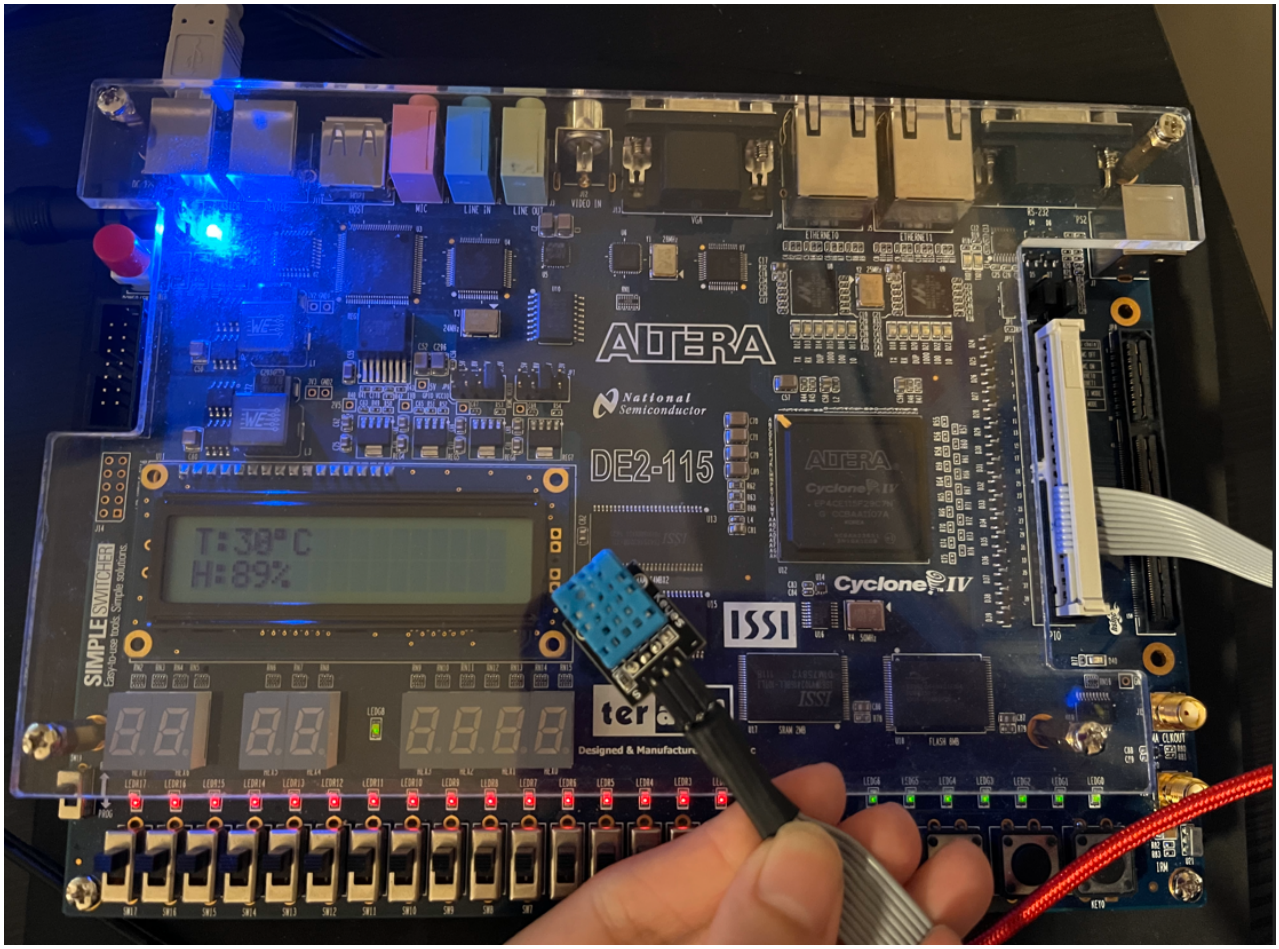

2024-05-04

Author: Jiaying Wu

Project: 13

Course: Digital Electronics Design with VHDL

Abstract

The project task is to design a 1-wire interface which shows the humidity and temperature on an LCD-display embedded in Altera DE2 FPGA Board. Data is read and updated every minute from the DHT11 sensor. The frequency is suggested as 400 Hz for LCD display and 400k Hz for the sensors.

Table of contents

Abstract 2

1. Introduction 4

2. Project Description 5

2.1 Tools & Software 5

2.2 design 6

3. Theory 7

3.1 LCD 7

3.2 Clock divider 8

3.3 DHT11 sensor 8

4. Implementation 10

4.1 LCD 10

4.2 Clock Divider 400 HZ & 400KHz 11

4.3 DHT11 sensor 12

4.4 Binary to BCD converter 13

4.5 Complete Design 13

5. Tests & Result 14

6. Conclusion 15

7. Appendix 16

References 24

1. Introduction

This project offers the possibility to acquire the temperature and humidity data from a DHT11 sensor and display them on the LCD screen which is embedded on the DE2-115 board.

The project gives a valuable knowledge and better understanding about the single-wire communication between FPGA board and external DHT11 sensor. A principle drawing for the system can be seen in **Figure 1.1**.

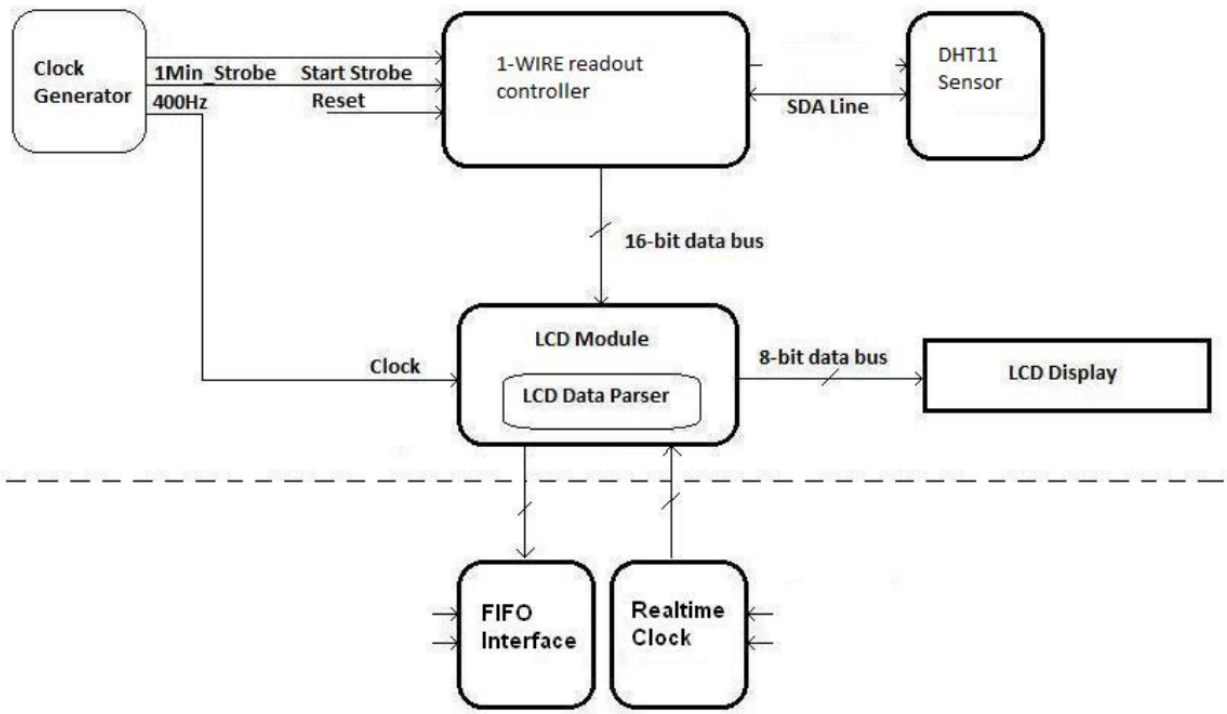

Figure 1.1 The overview drawing of the system.

2. Project Description

2.1 Tools & Software

The project task is to design a 1-wire interface to humidity- and temperature sensor with LCD-display in Quartus by using VHDL. This software enables users to individually design and produce components while also verifying the RTL figure. After the compilation of the connected design, one can open ModelSim simulator to examine the result. At the same time, it is also possible to implement the design onto hardware via a DE2_SOC card. The layout of DE2-115 is shown in **Figure 2.1** below:

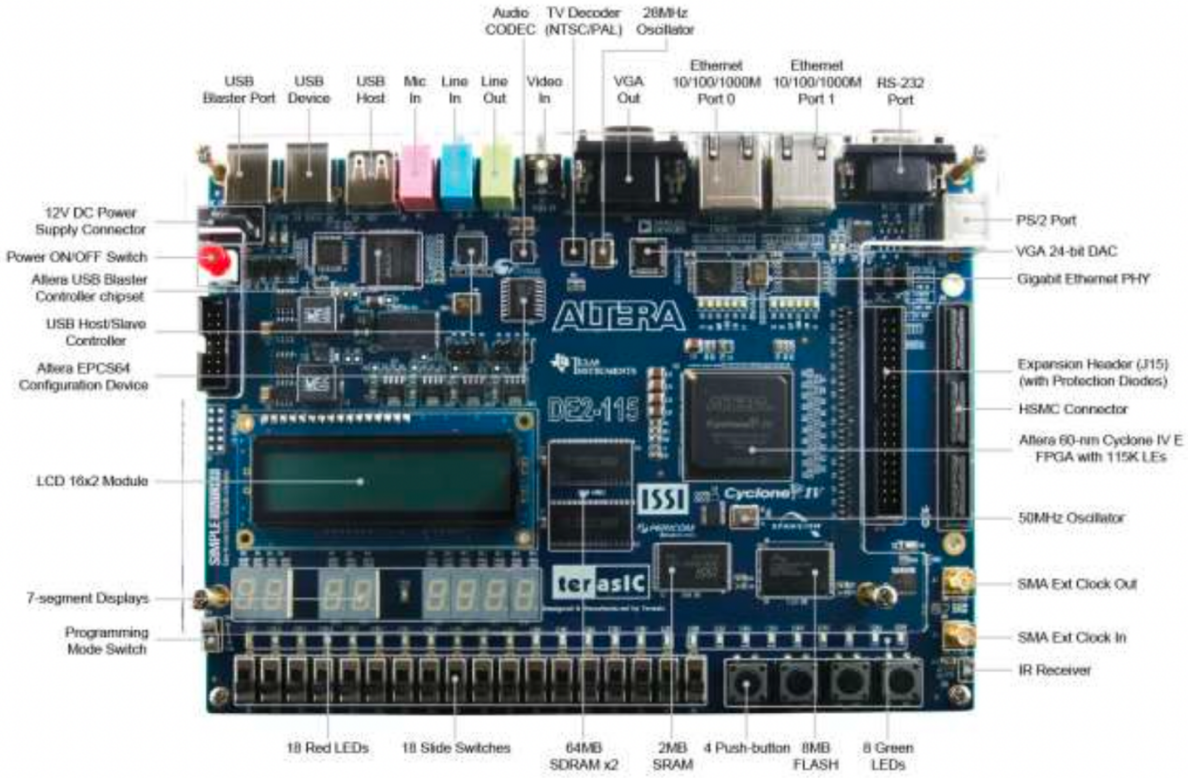

Figure 2.1 The layout of DE2-115 board.

This board has three important features as to this project:

- 50MHz oscillator clock inputs.
- 16x2 LCD display module
- 40-pin Expansion Port

In this task, FPGA is configured in JTAG Mode. To ensure this, one should set the RUN/PROG switch SW19 to the RUN position as **Figure 2.2** and **Figure 2.3** demonstrated.

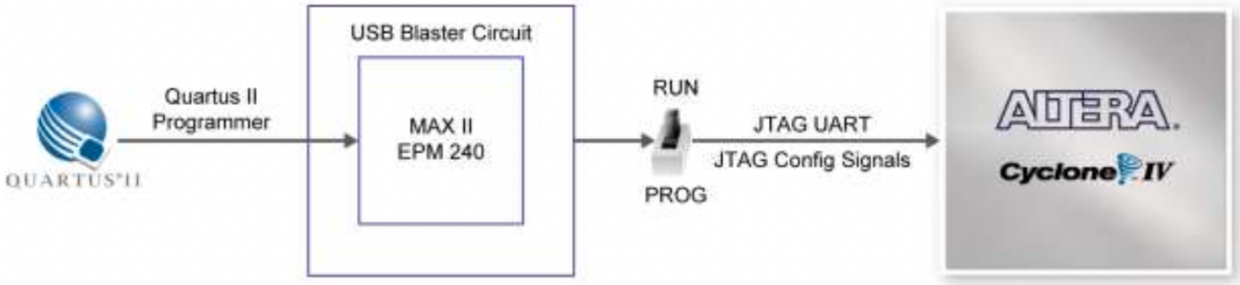

Figure 2.2 The JTAG configuration scheme.

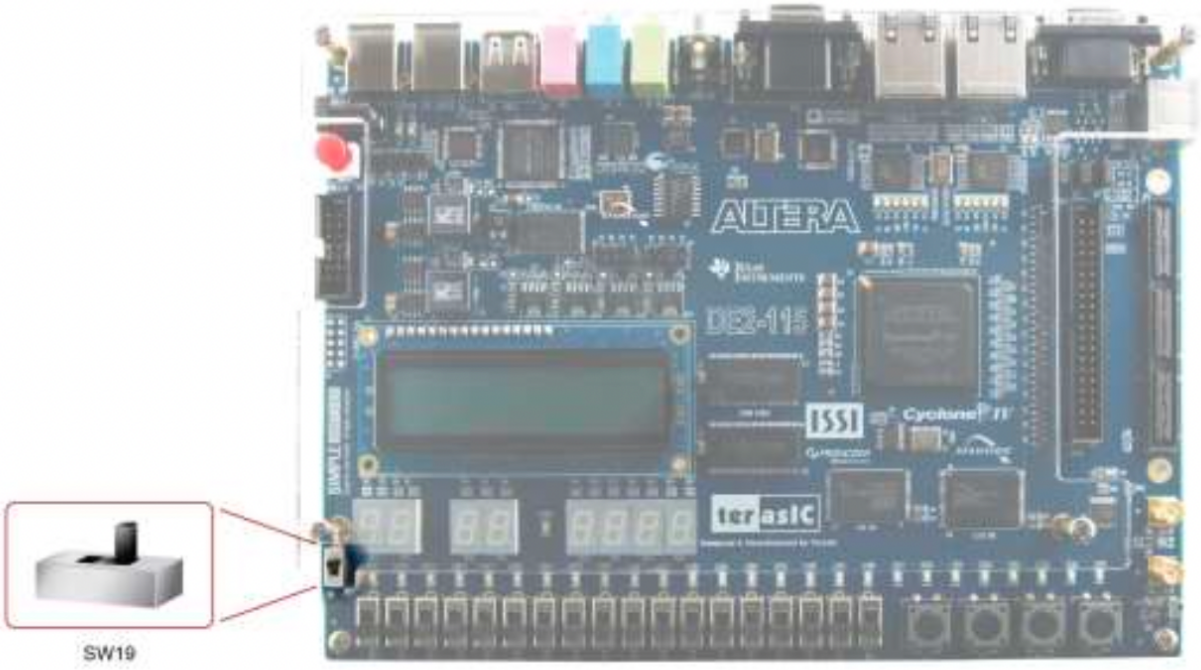

Figure 2.3 SW19 is set in JTAG mode.

2.2 design

The overall design is composed of 4 modules, as described below:

1. **lcd_only**: activate and control the display screen HD44780.
2. **clk_div_2**: take 50M Hz as input and output clock frequencies of 400 Hz and 400K Hz.
3. **dht11**: read out 16-bit temperature and humidity data every minute and transmit through SDA line while sensor is connected to the board.
4. **bi2bcd:** convert the binary numbers to BCD form before it is sent to the LCD data bus.

3. Theory

3.1 LCD

The lcd controller itself needs sequential execution of initialization every time when get reset or power. In this project it is implemented through state machine. It basically goes through:

- Function set
- Display on/off control
- Entry mode set
- Write data to CGRAM/DDRAM
- Return home

This product has two 8-bit registers, that is instruction register and data register. The operation towards these two registers could be decided by different situations of the values of RS and RW, which is basically demonstrated by **Figure 3.1:**

RS	R/$\overline{W}$	Operation
0	0	IR write as an internal operation (display clear, etc.)
0	1	Read busy flag (DB7) and address counter (DB0 to DB6)
1	0	DR write as an internal operation (DR to DDRAM or CGRAM)
1	1	DR read as an internal operation (DDRAM or CGRAM to DR)

Figure 3.1 the register selections

In this project, the 8-bit data needs to be sent, therefore the eight bus lines (DB0 to DB7) are used. If the data is about to be read, for example, RS should be set to 0 while RW to 1. The figure **Figure 3.2** shows an example of the 4-bit transfer procedure.

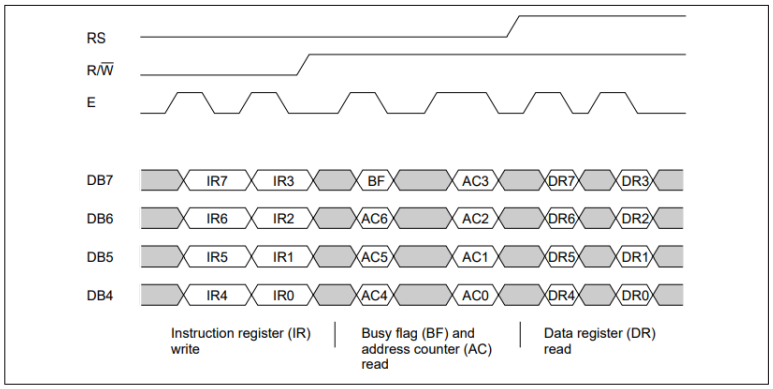

Figure 3.2 4-Bit Transfer Example

3.2 Clock divider

This is actually a frequency divider, using a counting signal to generate delay, when the value reaches the certain amount, another signal would toggle. In this way, a lower output frequency could be generated. Technically, this divider could be varied by modifying the counting value.

3.3 DHT11 sensor

As **Figure 3.3** shows, DHT11 sensor has one port that can be used for data transmission. This is served as a bidirectional port since the single bus is used for both communication and synchronization between MCU and DHT11 sensor. Therefore, the master device must strictly follow the sequence and only when the master calls the slave, the slave device can answer. To avoid contention, the tri-state buffer is achieved by the circuit design in **Figure 3.4**, a pull-up resistor is used.

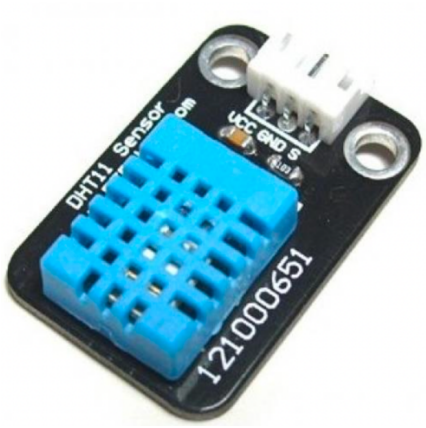

Figure 3.3 DHT11 sensor.

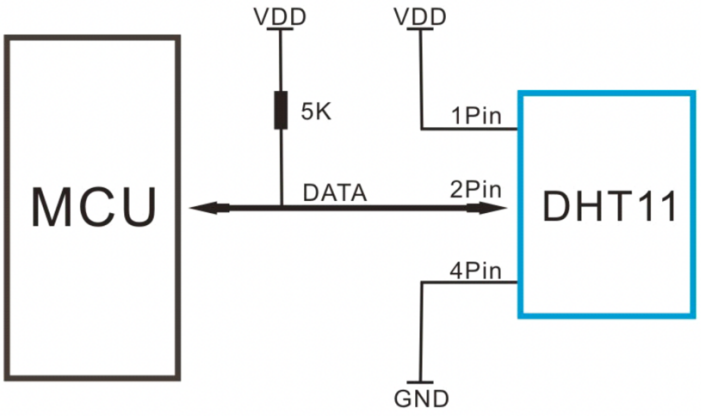

Figure 3.4 Master-slave communication interface

The buffer has three possible outputs: high, low, and high impedance ('Z'). When the slave device wants to transmit data, it activates the tri-state buffer to drive the data bus, otherwise the buffer will be set to the high impedance state, then the sensor and MCU is disconnected.

The sensor sends higher data bit first. A complete data transmission is 40 bits which is consist of 8-bit humidity integer data + 8-bit humidity fractional data + 8-bit temperature integer data + 8-bit temperature fractional data + 8-bit parity bit.

The entire data transmission process can be described as the following state transition:

1. Initial: the data line is floating for 1 second, the signal which is used for counting received sensor data bit is set to be 40.
2. Master_call: MCU sends a start signal and pulls the data line down for 18 ms.
3. Master_wait: MCU pulls up data line and wait for 20-40 us for sensor's response. State exit happens when the falling edge is detected.
4. Slave_response: The sensor detected the start signal and pulls down data line 80 us as response. State exit happens when the rising edge is detected.

5. Slave_prepare: The sensor pulls up data line for another 80 us as preparation for sending data. When the falling edge is detected, switch to next state.
6. Data_begin: When DHT11 is sending data to MCU, every bit begins with 50 us ‘0’. Once the rising edge is detected, and the 40-bit data is not completely transferred, go to next state. Otherwise go to the last state.
7. Date_read: A temporary buffer will take in data bit by bit. If the data line has been pulled up more than 50 us, this bit will be considered as ‘1’, otherwise ‘0’. At the same time, the data bit counter will be updated and go back to the former state.
8. Data_end: After the last bit data is transmitted, the integer part of humidity and temperature will be read. Back to the initial state.

The procedure is also described in **Figure 3.5**:

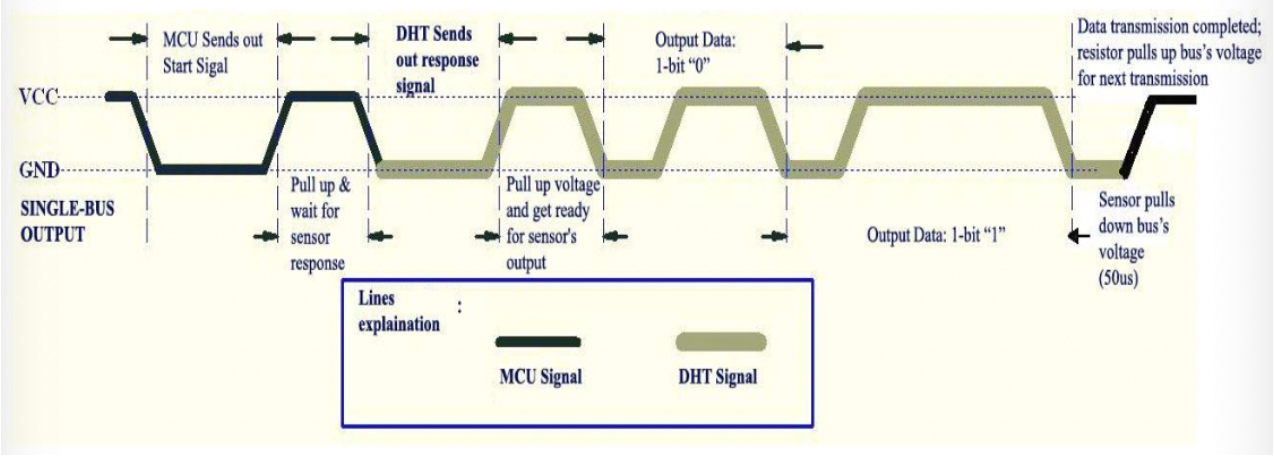

Figure 3.5 Communication Process

4. Implementation

4.1 LCD

This component includes four input ports to the left and six output ports to the right. 8-bit humidity data and 8-bit temperature data will be displayed. The thought is based on the datasheet, as **Figure 4.1** shows.

Figure 4.1 The LCD module on the board

There are nine mandatory setting states in all, including the “disable” one: idle, idle_n ,function_set, function_set_n, display_control, display_control_n, entry_mode_set, entry_mode_set_n,return_home. Except from these, there’re states for showing specific character pattern according to character code in the HD44780 datasheet.

The symbol and RTL view could be seen in **Figure 4.2** and **Figure 4.3.**

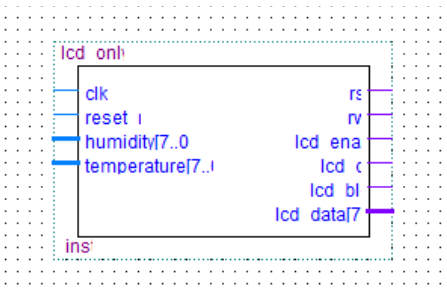

Figure 4.2 The symbol of LCD module.

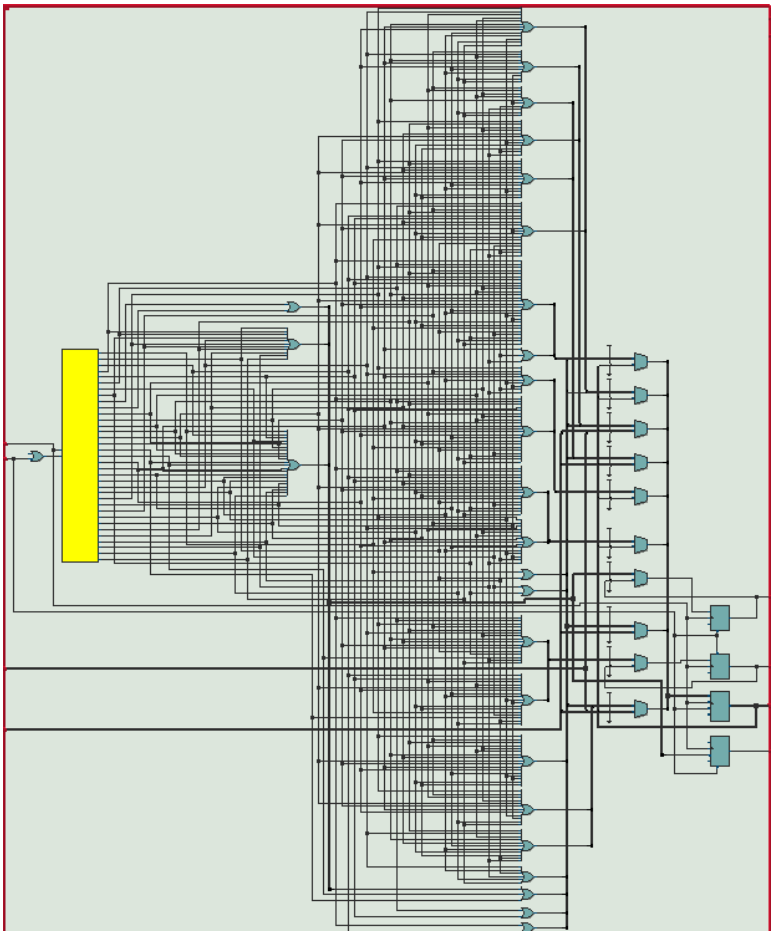

Figure 4.3 The LCD RTL view

4.2 Clock Divider 400 Hz & 400KHz

This component takes 50M Hz clock frequency on board as input and two output ports to the right. The counting value is set to be 62500 to generate 400 HZ output signal for lcd component and 63 to generate 400k Hz output signal for DHT11 sensor component. The symbol and RTL view could be seen in **Figure 4.4** and **Figure 4.5.**

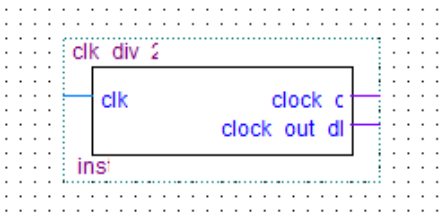

Figure 4.4 The symbol of the clock divider module.

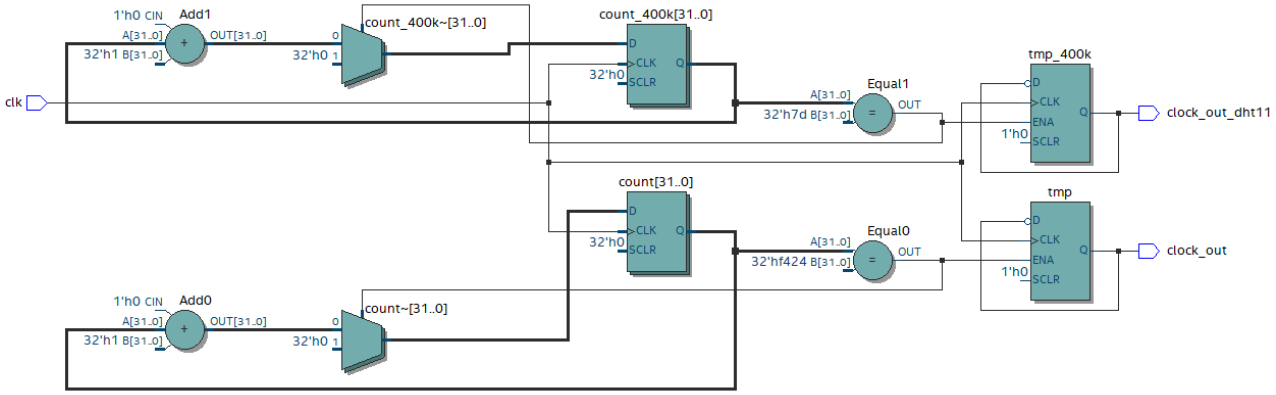

Figure 4.5 The clock divider RTL viewer

4.3 DHT11 sensor

This component includes two input ports, two output ports with 16 bits data together and one bidirectional port. The sensor data reading function are implemented in this component. The symbol and RTL view could be seen in **Figure 4.6** and **Figure 4.7**

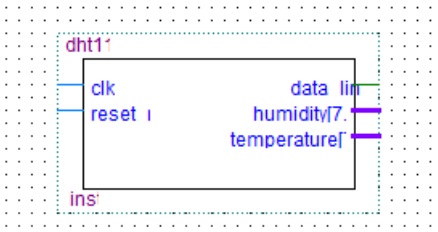

Figure 4.6 The symbol of the sensor module.

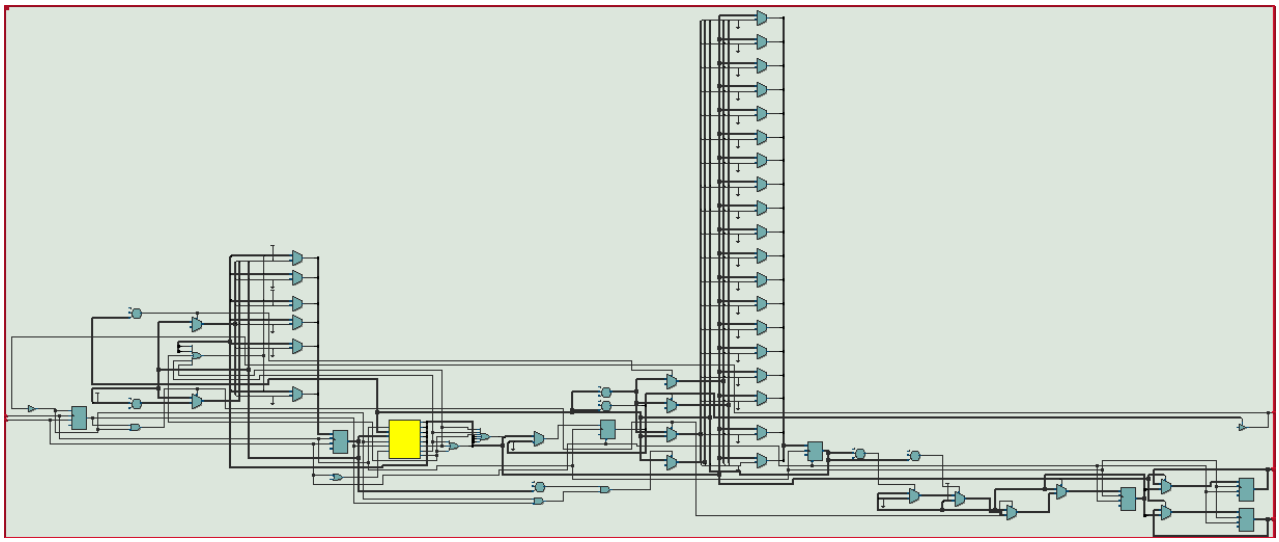

Figure 4.7 DHT11 RTL view.

4.4 Binary to BCD converter

This component receives 8-bit binary data as input and outputs 8-bit data in the form of binary-coded decimal. The symbol and RTL view could be seen in **Figure 4.8** and **Figure 4.9.**

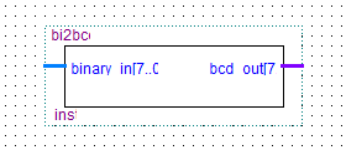

Figure 4.8 The symbol of the Binary to BCD module.

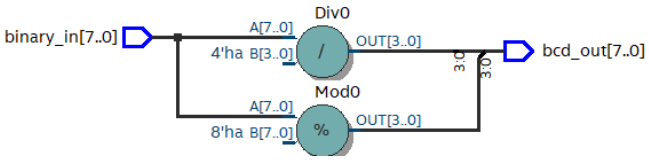

Figure 4.9 The symbol of the Binary to BCD module.

4.5 Complete Design

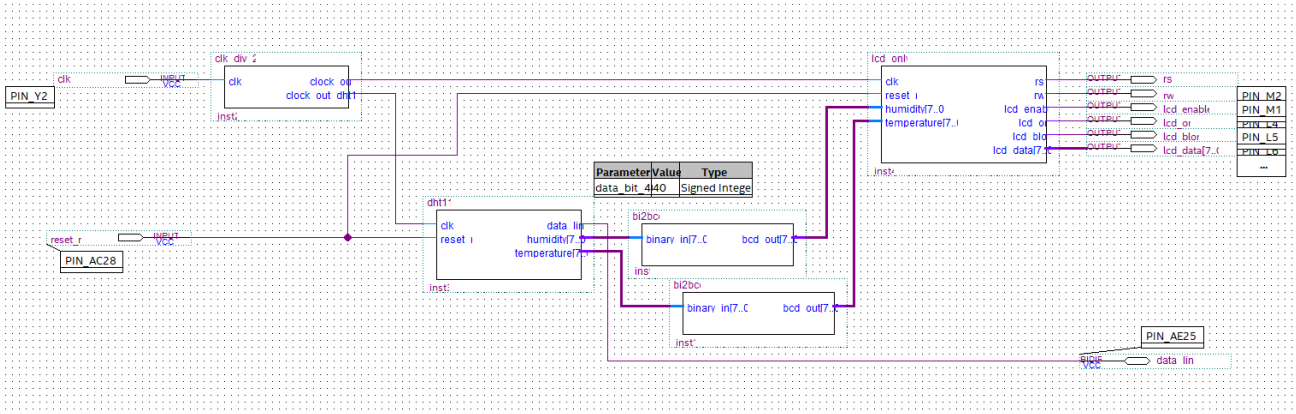

Figure 4.10 Complete block diagram of the project.

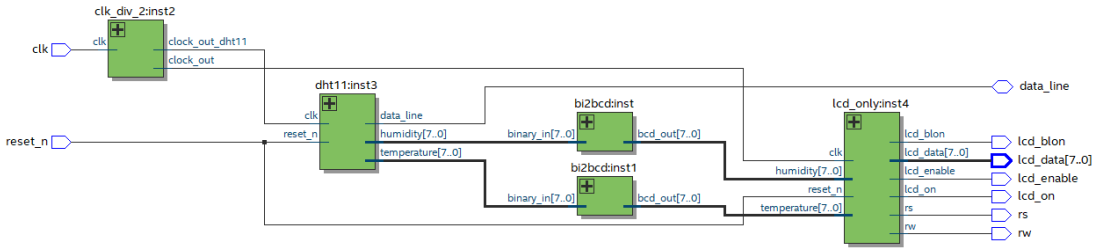

Figure 4.11 Complete RTL view of the project.

5. Tests & Result

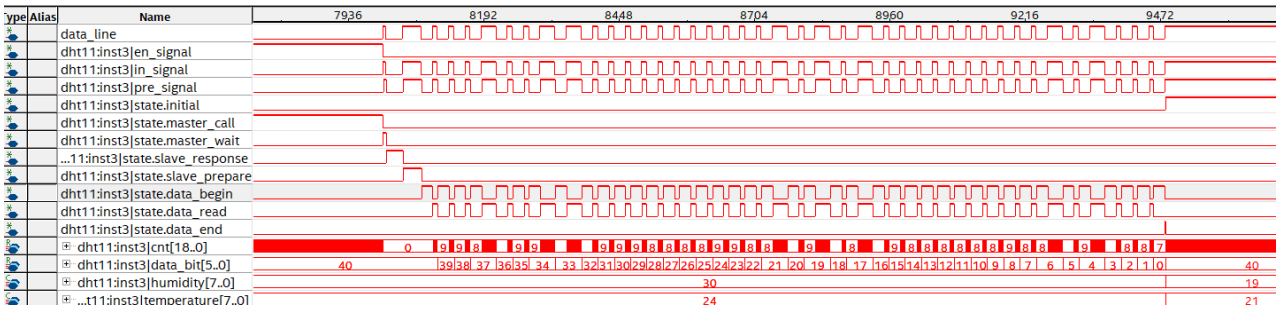

Figure 5.1 Signal Tap of the DHT11 sensor.

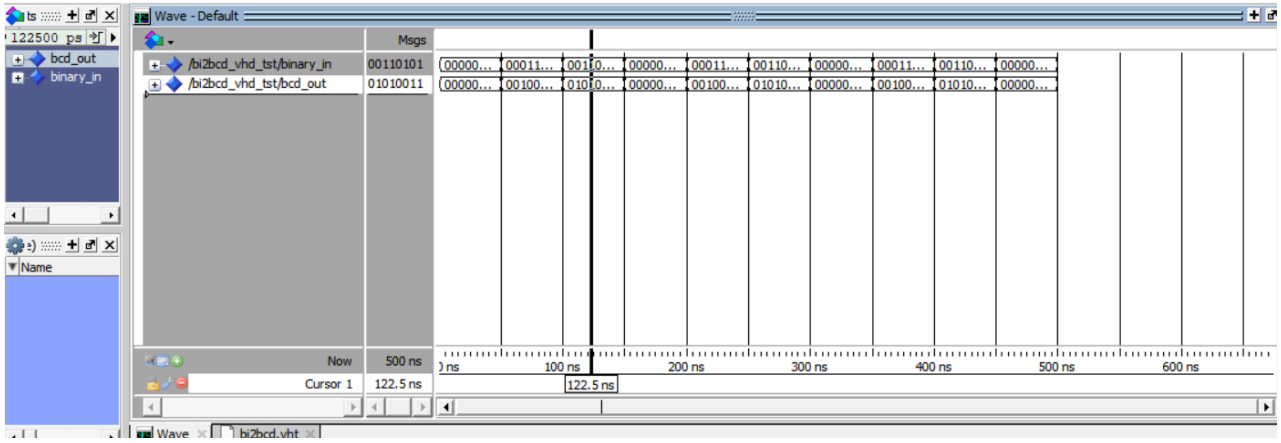

Figure 5.2 Simulation of Binary to BCD converter.

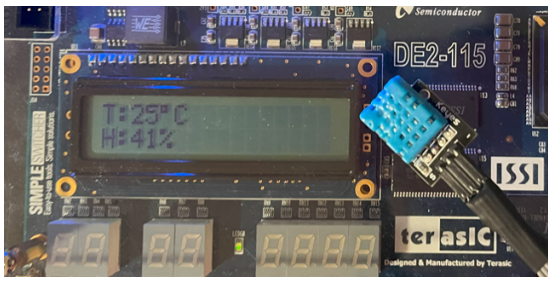

Figure 5.3 Result on board 1.

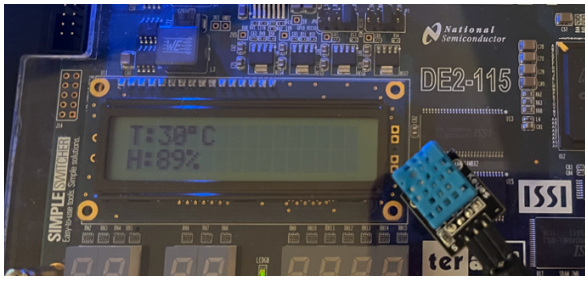

Figure 5.4 Result on board 2.

6. Conclusion

The main task is to implement the 1-wire communication between DE2-115 board and DHT11 sensor according to the strict time sequence. Eventually the decimal part of humidity and temperature is displayed on the LCD screen and is possible to be updated.

7. Appendix

1. lcd_only.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY lcd_only IS
   PORT(
         clk         : in std_logic; -- after clock divider, it's 400 HZ now
         reset_n     : in std_logic; -- set'0' will restart, back to idle state
         humidity    : in std_logic_vector(7 downto 0);
         temperature : in std_logic_vector(7 downto 0);

         rs          : out std_logic; --set'1' will send data
                                      --'0' will send instruction
         rw          : out std_logic; --set'1' will read
                                      --'0' will write
         lcd_enable  : out std_logic; --set'1' will read data from the bus

         lcd_on      : out std_logic;
         lcd_blon    : out std_logic;--back light
         lcd_data    : out std_logic_vector(7 downto 0)


         );
END lcd_only;

ARCHITECTURE controller OF lcd_only IS

   -- state declaration
   type lcd_control is (idle, idle_n ,function_set, function_set_n, display_clear, display_clear_n, display_control, display_control_n,
                        entry_mode_set, entry_mode_set_n,

                        write_T, write_T_n, write_colon, write_colon_n, data_T_tens, data_T_tens_n, data_T_ones, data_T_ones_n,

                        write_degree, write_degree_n, write_C, write_C_n, next_line, next_line_n,

                        write_H, write_H_n, write_colon2, write_colon2_n,  data_H_tens, data_H_tens_n, data_H_ones, data_H_ones_n,

                        write_percentage, write_percentage_n,

                        return_home);

   signal state : lcd_control;
```

```
47  BEGIN
48
49      lcd_on   <= '1';
50      lcd_blon <= '0'; -- back light always off
51
52      PROCESS(clk)
53      BEGIN
54
55          if(reset_n ='0') then
56                  lcd_enable <= '1';
57                  rs         <= '0';
58                  rw         <= '0';
59
60                  state <= idle;
61
62          elsif(clk'EVENT and clk = '1') then
63
64             case state is
65  ------------------------------------------------------------------------
66                      when idle   =>
67                                                  rs       <= '0';        -- send instructions
68                                                  rw       <= '0';        -- write
69                                                  lcd_enable <= '1';
70                                                  state    <= idle_n;
71
72                      when idle_n   =>
73                                                  rs       <= '0';
74                                                  rw       <= '0';
75                                                  lcd_enable <= '0';
76                                                  state    <= function_set;
77  ------------------------------------------------------------------------
78                      when function_set   =>
79                                                  rs       <= '0';        -- send instructions
80                                                  rw       <= '0';        -- write
81                                                  lcd_data <= "00111000"; -- 2-line, 5*8 - 0x38
82                                                  lcd_enable <= '1';
83                                                  state    <= function_set_n;
84
85                      when function_set_n =>
86                                                  rs       <= '0';
87                                                  rw       <= '0';
88                                                  lcd_data <= "00111000";
89                                                  lcd_enable <= '0';
90                                                  state    <= display_control;
91  ------------------------------------------------------------------------

92                      when display_control   =>
93                                                  rs       <= '0';
94                                                  rw       <= '0';
95                                                  lcd_data <= "00001100"; -- display on, cursor off, blinking off
96                                                  lcd_enable <= '1';
97                                                  state    <= display_control_n;
98
99                      when display_control_n =>
100                                                 rs       <= '0';
101                                                 rw       <= '0';
102                                                 lcd_data <= "00001100";
103                                                 lcd_enable <= '0';
104                                                 state    <= entry_mode_set;
105 ------------------------------------------------------------------------
106                     when entry_mode_set  =>
107                                                 rs       <= '0';
108                                                 rw       <= '0';
109                                                 lcd_data <= "00000110"; -- shift cursor to the right, display is not shifted
110                                                 lcd_enable <= '1';
111                                                 state    <= entry_mode_set_n;
112
113                     when entry_mode_set_n =>
114                                                 rs       <= '0';
115                                                 rw       <= '0';
116                                                 lcd_data <= "00000110";
117                                                 lcd_enable <= '0';
118                                                 state    <= write_T;
119 ------------------------------------------------------------------------
120                     when write_T        =>
121                                                 rs       <= '1';
122                                                 rw       <= '0';
123                                                 lcd_data <= "01010100";  -- 0x54 -- 'T'
124                                                 lcd_enable <= '1';
125                                                 state    <= write_T_n;
126                     when write_T_n      =>
127                                                 rs       <= '0';
128                                                 rw       <= '0';
129                                                 lcd_data <= "01010100";
130                                                 lcd_enable <= '0';
131                                                 state    <= write_colon;
132 ------------------------------------------------------------------------
```

```
132 ----------------------------------------------------------------------
133                     when write_colon        =>
134                                             rs       <= '1';
135                                             rw       <= '0';
136                                             lcd_data <= "00111010";  -- 0x3A -- ':'
137                                             lcd_enable <= '1';
138                                             state    <= write_colon_n;
139                     when write_colon_n      =>
140                                             rs       <= '0';
141                                             rw       <= '0';
142                                             lcd_data <= "00111010";
143                                             lcd_enable <= '0';
144                                             state    <= data_T_tens;
145 ----------------------------------------------------------------------
146                     when data_T_tens        =>
147                                             rs       <= '1';
148                                             rw       <= '0';
149                                             lcd_data <= "0011" & temperature(7 downto 4);
150                                             lcd_enable <= '1';
151                                             state    <= data_T_tens_n;
152                     when data_T_tens_n      =>
153                                             rs       <= '0';
154                                             rw       <= '0';
155                                             lcd_data <= "0011" & temperature(7 downto 4);
156                                             lcd_enable <= '0';
157                                             state    <= data_T_ones;
158 ----------------------------------------------------------------------
159                     when data_T_ones        =>
160                                             rs       <= '1';
161                                             rw       <= '0';
162                                             lcd_data <= "0011" & temperature(3 downto 0);
163                                             lcd_enable <= '1';
164                                             state    <= data_T_ones_n;
165                     when data_T_ones_n      =>
166                                             rs       <= '0';
167                                             rw       <= '0';
168                                             lcd_data <= "0011" & temperature(3 downto 0);
169                                             lcd_enable <= '0';
170                                             state    <= write_degree;
171 ----------------------------------------------------------------------

172                       when write_degree        =>
173                                                rs       <= '1';
174                                                rw       <= '0';
175                                                lcd_data <= "11011111";  -- degree
176                                                lcd_enable <= '1';
177                                                state    <= write_degree_n;
178                       when write_degree_n      =>
179                                                rs       <= '0';
180                                                rw       <= '0';
181                                                lcd_data <= "11011111";
182                                                lcd_enable <= '0';
183                                                state    <= write_C;
184 ----------------------------------------------------------------------
185                       when write_C        =>
186                                                rs       <= '1';
187                                                rw       <= '0';
188                                                lcd_data <= "01000011";  -- C
189                                                lcd_enable <= '1';
190                                                state    <= write_C_n;
191                       when write_C_n      =>
192                                                rs       <= '0';
193                                                rw       <= '0';
194                                                lcd_data <= "01000011";
195                                                lcd_enable <= '0';
196                                                state    <= next_line;
197 ----------------------------------------------------------------------
198                       when next_line         =>
199                                                rs       <= '0';
200                                                rw       <= '0';
201                                                lcd_data <= "11000000";
202                                                lcd_enable <= '1';
203                                                state    <= next_line_n;
204                       when next_line_n       =>
205                                                rs       <= '0';
206                                                rw       <= '0';
207                                                lcd_data <= "00000000";
208                                                lcd_enable <= '0';
209                                                state    <= write_H;
210 ----------------------------------------------------------------------
```

```
210 ------------------------------------------------------------------------
211                     when write_H        =>
212                                             rs       <= '1';
213                                             rw       <= '0';
214                                             lcd_data <= "01001000";  -- 0x48 -- 'H'
215                                             lcd_enable <= '1';
216                                             state    <= write_H_n;
217                     when write_H_n      =>
218                                             rs       <= '0';
219                                             rw       <= '0';
220                                             lcd_data <= "01001000";
221                                             lcd_enable <= '0';
222                                             state    <= write_colon2;
223 ------------------------------------------------------------------------
224                     when write_colon2       =>
225                                             rs       <= '1';
226                                             rw       <= '0';
227                                             lcd_data <= "00111010";  -- 0x3A -- ':'
228                                             lcd_enable <= '1';
229                                             state    <= write_colon2_n;
230                     when write_colon2_n     =>
231                                             rs       <= '0';
232                                             rw       <= '0';
233                                             lcd_data <= "00111010";
234                                             lcd_enable <= '0';
235                                             state    <= data_H_tens;
236 ------------------------------------------------------------------------
237                     when data_H_tens        =>
238                                             rs       <= '1';
239                                             rw       <= '0';
240                                             lcd_data <= "0011" & humidity(7 downto 4);
241                                             lcd_enable <= '1';
242                                             state    <= data_H_tens_n;
243                     when data_H_tens_n      =>
244                                             rs       <= '0';
245                                             rw       <= '0';
246                                             lcd_data <= "0011" & humidity(7 downto 4);
247                                             lcd_enable <= '0';
248                                             state    <= data_H_ones;
249 ------------------------------------------------------------------------
250                     when data_H_ones        =>
251                                                 rs       <= '1';
252                                                 rw       <= '0';
253                                                 lcd_data <= "0011" & humidity(3 downto 0);
254                                                 lcd_enable <= '1';
255                                                 state    <= data_H_ones_n;
256                     when data_H_ones_n      =>
257                                                 rs       <= '0';
258                                                 rw       <= '0';
259                                                 lcd_data <= "0011" & humidity(3 downto 0);
260                                                 lcd_enable <= '0';
261                                                 state    <= write_percentage;
262 ------------------------------------------------------------------------
263                     when write_percentage        =>
264                                                 rs       <= '1';
265                                                 rw       <= '0';
266                                                 lcd_data <= "00100101";  -- %
267                                                 lcd_enable <= '1';
268                                                 state    <= write_percentage_n;
269                     when write_percentage_n      =>
270                                                 rs       <= '0';
271                                                 rw       <= '0';
272                                                 lcd_data <= "00100101";
273                                                 lcd_enable <= '0';
274                                                 state    <= return_home;
275 ------------------------------------------------------------------------
276                     when return_home    =>
277                                                 rs       <= '0';
278                                                 rw       <= '0';
279                                                 lcd_data <= "00000010";
280                                                 lcd_enable <= '0';
281                                                 state    <= idle;
282
283                     when others => NULL; --state <= idle;
284                 end case;
285 ------------------------------------------------------------------------
286
287             end if;
288
289       END PROCESS;
290
291 END controller;
```

2. clk_div_2.vhd

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

entity clk_div_2 is
port (
      clk      : in std_logic;       -- 50M HZ
      clock_out: out std_logic;      -- 400 HZ for LCD
      clock_out_dht11: out std_logic -- 400K HZ for DHT11 sensor
   );
end clk_div_2;

architecture bhv of clk_div_2 is

signal count: integer:= 0;
signal tmp : std_logic := '0';

signal count_400k: integer:= 0;
signal tmp_400k : std_logic := '0';

begin
   process(clk)
   begin

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

         if (count = 62500) then   --50M/2/400
            count <= 0;
            tmp <= NOT tmp;
         else
            count <=count + 1;
         end if;
      end if;
   end process;

   process(clk)
   begin

      if(clk'event and clk='1') then
         if (count_400k = 63) then  --50M/2/400000
            count_400k <= 0;
            tmp_400k <= NOT tmp_400k;
         else
            count_400k <= count_400k + 1;
         end if;
      end if;
   end process;

   clock_out <= tmp;
   clock_out_dht11 <= tmp_400k;

end bhv;
```

3. dht11.vhd

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


entity dht11 is
    generic(
          data_bit_40 : integer := 40);
    port(
          clk         : in std_logic;  -- after clock divider, it's 400kHZ now
          reset_n     : in std_logic; -- set'0' will restart
          data_line   : inout std_logic; -- SDA line, will be toggled to 1 or 0
          humidity    : out std_logic_vector(7 downto 0);
          temperature: out std_logic_vector(7 downto 0));

end dht11;

architecture bhv of dht11 is

    type dht_control is ( initial, master_call,master_wait, slave_response, slave_prepare,
                          data_begin, data_read, data_end);
    signal state : dht_control := initial;

    --signal cnt : integer range 0 to 399999 := 0;
    signal cnt : integer range 0 to 400000 := 0;

    signal data_bit : integer range 40 downto 0;
    signal data_get : std_logic_vector(39 downto 0) := (others => '0'); -- Initializing all bits to '0'
    signal tmp_get : std_logic_vector(39 downto 0) := (others => '0');

    signal humidity_tmp : std_logic_vector(7 downto 0) := (others => '0');
    signal temperature_tmp : std_logic_vector(7 downto 0) := (others => '0');

    signal pre_signal : std_logic;  -- store the voltage of the dataline before
    signal en_signal : std_logic := '0';    -- for the inout buffer data_line, set it input or output
    signal in_signal : std_logic;   -- I cannot assign value to data_line("multiple constant drivers"),
                                    -- instead, I need this take enable signal from data_line

    constant clk_period : time := 2.5 us; -- input clk 400kHZ
    constant delay_18_ms : integer := 7200; -- 18*10e-3 / 2.5*10e-6
    constant delay_20_ms : integer := 8000;
    constant delay_40_us : integer := 16;  -- 40*10e-6 / 2.5*10e-6
    constant delay_50_us : integer := 20;
    constant delay_80_us : integer := 32;
    constant delay_1_s : integer := 400000;
```

```
49  begin
50     process(clk)
51     begin
52
53        if(reset_n = '0') then
54           cnt <= 0;
55           data_get <= (others => '0');
56           en_signal <= '0';
57           state <= initial;
58
59        elsif(clk'EVENT and clk = '1') then
60
61           case state is
62  ------------------------------------------------------------------------
63                       when initial   => if (cnt >= delay_1_s) then
64                                            cnt <= 0;
65                                            data_bit <= data_bit_40; -- 40 bit
66                                            state <= master_call;
67                                         else
68                                            cnt <= cnt + 1;
69                                         end if;
70
71   --------- MCU sets data line from 1 to 0 at least 18ms, make sure DHT11 can detect it.
72
73                       when master_call   => if (cnt >= delay_20_ms) then
74                                                cnt <= 0;
75                                                en_signal <= '0'; --------------- data_line <= 'Z'
76                                                state <= master_wait;
77                                             else
78                                                cnt <= cnt + 1;
79                                                --data_line <= '0';
80                                                en_signal <= '1'; -------------- data_line <= '0'
81                                             end if;
82
83  --------- MCU pulls up data line and wait 20~40us for DHT's response. When DHT responds, the data line gets pulled down.
84   --------- if '1' to '0', then go to next state.
85
86                       when master_wait => if pre_signal = '1' and in_signal = '0' then
87                                              state <= slave_response;
88                                           end if;
89
90  --------- DHT detects the signal, and pulls down data line 80us as response.
91   --------- if '0' to '1', the response signal is finished, go to next state.
92
93                       when slave_response => if pre_signal = '0' and in_signal = '1' then
94                                                state <= slave_prepare;
95                                             end if;

97  --------- DHT pulls up data line 80us for DHT preparation for sending data.
98   --------- if '1' to '0', the preparation is finished, go to next state.
99
100                      when slave_prepare   => if pre_signal = '1' and in_signal = '0' then
101                                                state <= data_begin;
102                                             end if;
103
104 --------- Every bit begin with 50us low-voltage-level.
105  --------- go to next state when '0' to '1'
106  --------- 1.data is still sending
107  --------- 2.all 40 bits has been received
108
109                      when data_begin   => if pre_signal = '0' and in_signal = '1' and data_bit > 0 then
110                                              state <= data_read;
111                                              cnt <= 0;
112                                           elsif pre_signal = '0' and in_signal = '1' and data_bit = 0 then
113                                             state <= data_end;
114                                           end if;
115
116 --------- if '1' to '0', one bit is transmitted.
117  --------- 1. pulls up for 50us : '0' (26-28us)
118  --------- 2. pulls up for 70us : '1' (70us)
119
120                      when data_read    =>  --if(data_line = '0') then  -- data line is low, start to transmit next bit data
121                                           if pre_signal = '1' and in_signal = '0' then
122                                               data_bit <= data_bit - 1; -- one bit is alrealy transmitted
123
124                                               if (cnt >= delay_50_us) then -- read '1'
125                                                   tmp_get <= tmp_get(data_bit_40 - 2 downto 0) & std_logic_vector(to_unsigned(1,1));
126                                               elsif (cnt < delay_50_us) then -- read '0'
127                                                  tmp_get <= tmp_get(data_bit_40 - 2 downto 0) & std_logic_vector(to_unsigned(0,1));
128                                               end if;
129
130                                           state <= data_begin; -- read next bit
131                                        else
132                                          cnt <= cnt + 1;    -- before data line turn to '0', count the delay to read data
133                                        end if;
```

```
--------- 2. pulls up for 70us : '1' (70us)

                    when data_end    =>  --data_line <= '1';   -- pull up data line
                                         en_signal <= '0'; ------------- data_line <= 'Z'
                                         data_bit <= data_bit_40; -- reset bit counter
                                         humidity_tmp <= tmp_get(39 downto 32);
                                         temperature_tmp <= tmp_get(23 downto 16);
                                         cnt <= 0;
                                         state <= initial;

      end case;
      pre_signal <= in_signal; -- cannot put before'when'
   end if;
   end process;

humidity <= humidity_tmp;
temperature <= temperature_tmp;

in_signal <= data_line; -- read the current voltage of the line
data_line <= '0' when en_signal ='1' else 'Z';  --tristate iobuffer, prevent contention.

end bhv;
```

4. bi2bcd.vhd

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

entity bi2bcd is

   Port (
      binary_in : in std_logic_vector (7 downto 0);
      bcd_out   : out std_logic_vector (7 downto 0)
   );

end bi2bcd;

architecture Behavioral of bi2bcd is

begin

   process(binary_in)

         variable temp_binary : integer range 0 to 255;
         variable temp_bcd    : std_logic_vector (7 downto 0);
   begin
         temp_binary := to_integer(unsigned(binary_in)); -- turn binary number to interger
                                                         -- to_integer takes unsigned as argument

         temp_bcd(7 downto 4) := std_logic_vector(to_unsigned(temp_binary/10, 4));       -- tens
         temp_bcd(3 downto 0) := std_logic_vector(to_unsigned(temp_binary mod 10, 4));   -- ones

         bcd_out <= temp_bcd;

   end process;

end Behavioral;
```

References

1. https://allaboutfpga.com/vhdl-code-for-clock-divider/
2. https://www.alldatasheet.com/datasheet-pdf/pdf/63673/HITACHI/HD44780.html
3. https://stackoverflow.com/questions/33264066/what-to-unsigned-does
4. Tri-State Buffers in VHDL (startingelectronics.org)https://stackoverflow.com/questions/50427271/interface-dht22-to-fpga-elbert-v2
5. vhdl - Interface DHT22 to FPGA - elbert v2 - Stack Overflow
6. fpga - VHDL Code explanation needed (std_logic_vector) - Stack Overflow
7. https://community.intel.com/t5/Intel-Quartus-Prime-Software/Dicrepancy-between-compiled-projects-in-Quartus-II-Prime/m-p/1481614/highlight/true
8. https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf
9. https://pdf1.alldatasheet.com/datasheet-pdf/view/63673/HITACHI/HD44780.html