CHALMERS
UNIVERSITY OF TECHNOLOGY

EDA234 Digital Project Laboratory

Thermometer Report

GitHub Repository

Group D

Autumn 2019

Angione Francesco

Ernesto Lozano Calvo

Candi Wu

Hong Zhou

Abstract

The aim of the project is to develop a system on FPGA which is capable of reading the indoor and outdoor temperature, the latter will be obtained through a wireless transmission from another board.
Depending on the user preferences the indoor or outdoor temperature will be shown on an LCD, together with its maximum and minimum value. In case of no action from the user, the temperature value will be refreshed every ten seconds. Regarding the humidity value, it will be used for changing the brightness of an LED and sampled every 100 milliseconds.

Contents

List of Figures

1 System specifications

Due to the circumstances in which this project is set (low volume, easy and quick implementation needed), the system is carried out by an FPGA.
It is important to mention that with the purpose of rising its complexity and versatility, RF communication as well as a humidity sensor with a PWM LED toggle control are added to the initial system.

Based on that, the needed components are collected in the following table:

COMPONENTS LIST		
Description	Type number	Data Sheet link
FPGA	XC7A100T1CSG324C	XilinxArtic7
Temperature Sensors	DS18S20	TempSensor
LCD	Digilent PmodCLP	Digilent LCD
Humidity Sensor	Hygrochip HYT-221	HumSensor
AM Hybrid Transmitter	AM-RT4-433	AM Transmitter
AM Super-Regen Receiver	AM-HRR30-433	AM Receiver

2 From Top to Down...

2.1 Starting point and evolution. Block diagrams

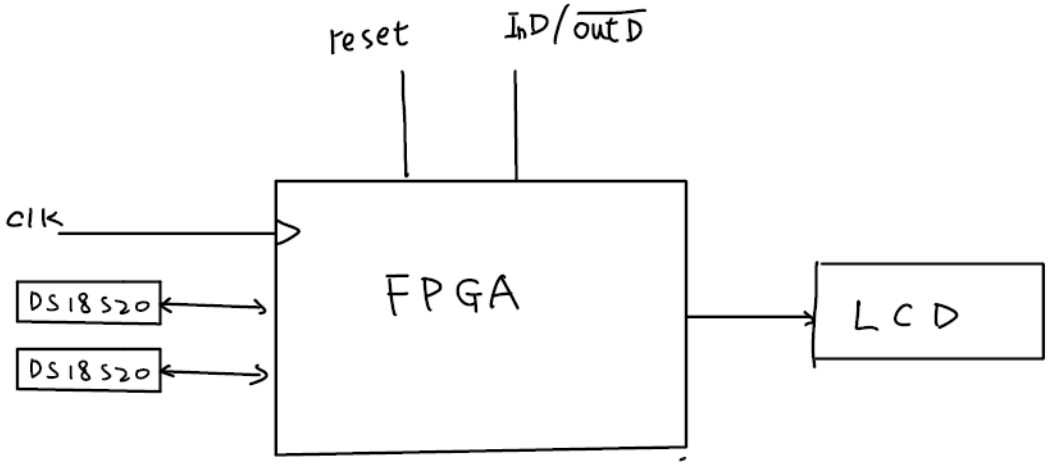


Figure 2.1: first version of block diagram

The previous Figure 2.1 shows the first approach (white paper) to the system design, a general sight of it, that represents the top level idea, following the previously mentioned requirements.

General view known, a deeper analysis of the system is needed and therefore presented in Figure 2.2.

The temperature read by the sensors is channeled by MUX road to an specific interface prepared for the reception and treating of the temperature values. Through a comparison block, the temperature values will be compared with the ones held in registers (maximum, minimum, etc) and the LCD interface will get and send to be printed those values if they

fulfilled the comparisons.Last but not least, as it can be seen, every single vital decision is carried out by the control unit, manifested as the brain of the system, enabling, resetting and keeping track of all the processes that take place.

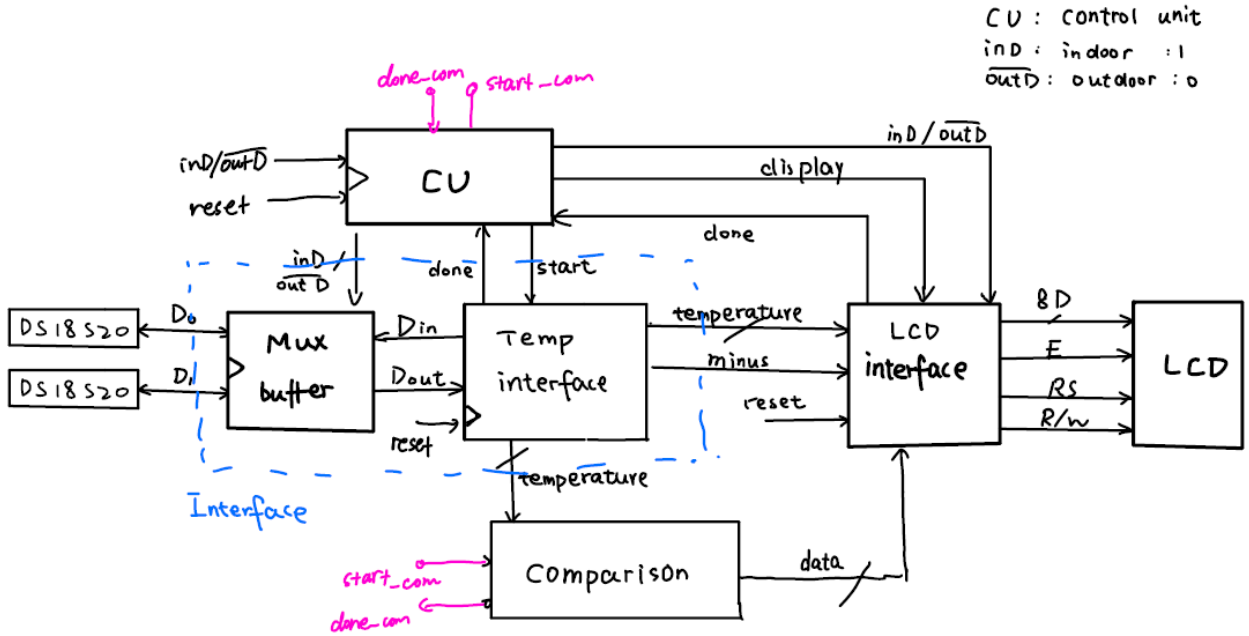
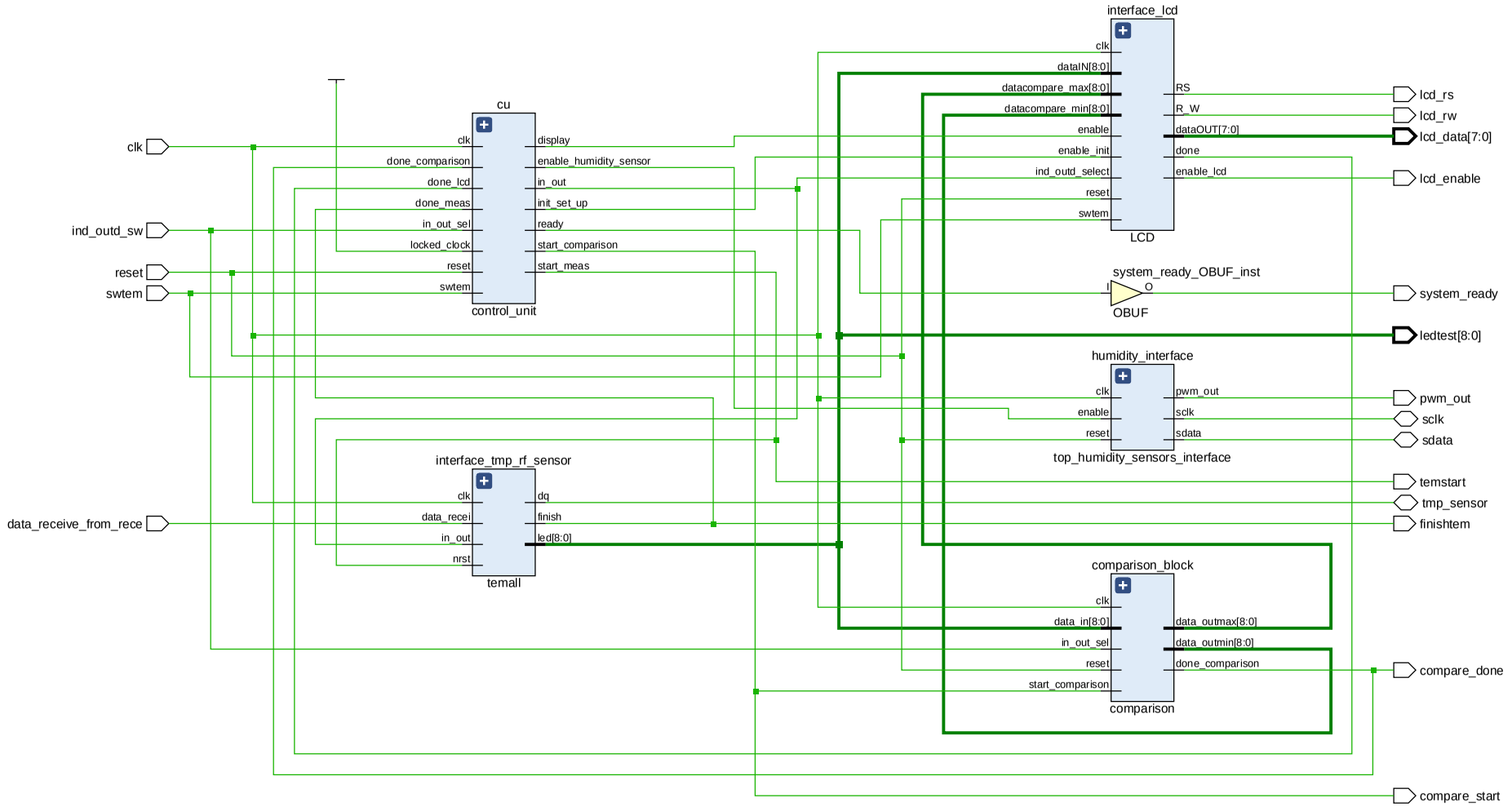


Figure 2.2: Deeper Detailed Concept Block Diagram

Wireless transmission module was added to the previous design, see Figure 2.3. Uses indoor FPGA's switch0 to control the LCD to display indoor/outdoor temperature, and switch1 to control the LCD to display the maximum, minimum, or real-time temperature. When switch0 selects outdoor (sw0 = '0'), after the outdoor temperature conversion is completed, the 9-bit temperature value will be transmitted to the receiver component by the wireless transmitter module, and then the receiver component will receive the 9 bit temperature value which is passed to the LCD or comparison module. When indoor (sw0 = '1') is selected for switch0, the LCD will display the temperature value of the indoor temperature sensor.

At this point, the diagram already allows the understanding of the different modules' roles and the starting of programming, acting as a base, a place to look when designing.

Getting the Register Transfer Level schematic from Vivado, the final version of the block diagram is the following:


clk
ind_outd_sw
reset
swtem
data_receive_from_rece
cu
clk
done_comparison
done_lcd
done_meas
in_out_sel
locked_clock
reset
swtem
display
enable_humidity_sensor
in_out
init_set_up
ready
start_comparison
start_meas
control_unit
interface_tmp_rf_sensor
clk
data_recei
in_out
nrst
dq
finish
led[8:0]
temall
interface_lcd
clk
dataIN[8:0]
datacompare_max[8:0]
datacompare_min[8:0]
enable
enable_init
ind_outd_select
reset
swtem
RS
R_W
dataOUT[7:0]
done
enable_lcd
LCD
system_ready_OBUF_inst
I
O
OBUF
humidity_interface
clk
enable
reset
pwm_out
sclk
sdata
top_humidity_sensors_interface
comparison_block
clk
data_in[8:0]
in_out_sel
reset
start_comparison
data_outmax[8:0]
data_outmin[8:0]
done_comparison
comparison
lcd_rs
lcd_rw
lcd_data[7:0]
lcd_enable
system_ready
ledtest[8:0]
pwm_out
sclk
sdata
temstart
tmp_sensor
finishtem
compare_done
compare_start

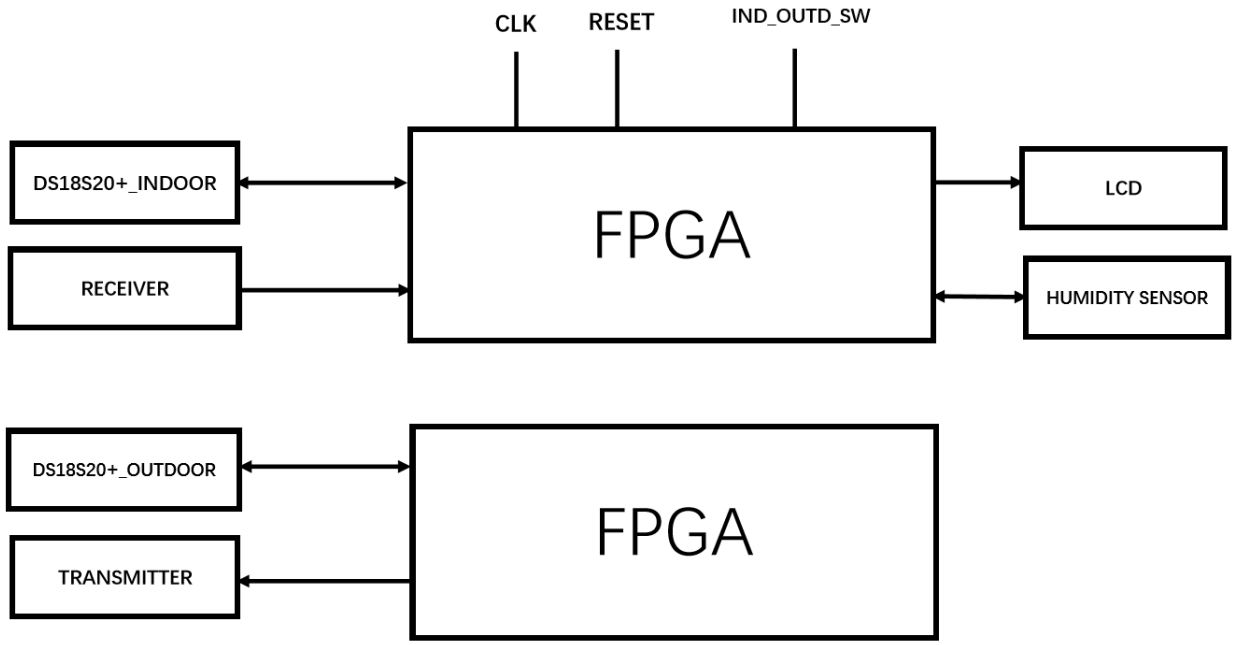


Figure 2.3: second version of block diagram

3 Temperature sensors and their interface

3.1 Conceptions of DS18S20+

Ds18s20+ is a 1-wire temperature sensor produced by Dallas. It has good economy, strong anti-interference ability, and can be used for temperature measurement in harsh conditions. The temperature range of Ds18s20+ is from -55°C to + 125°C , and the accuracy is ± 0.5 °C in the temperature range of -10 to + 85 °C . This device also has the advantages of minimal pins, simple interface, no external components required, flexible power supply mode, and the ability to connect multiple devices at the same time for multi-point temperature measurement. Each ds18s20 + has a globally unique 64-bit serial number. By identifying the serial number of each temperature sensor, multiple temperature sensors can be connected on one bus.

3.1.1 Ds18s20 + internal structure

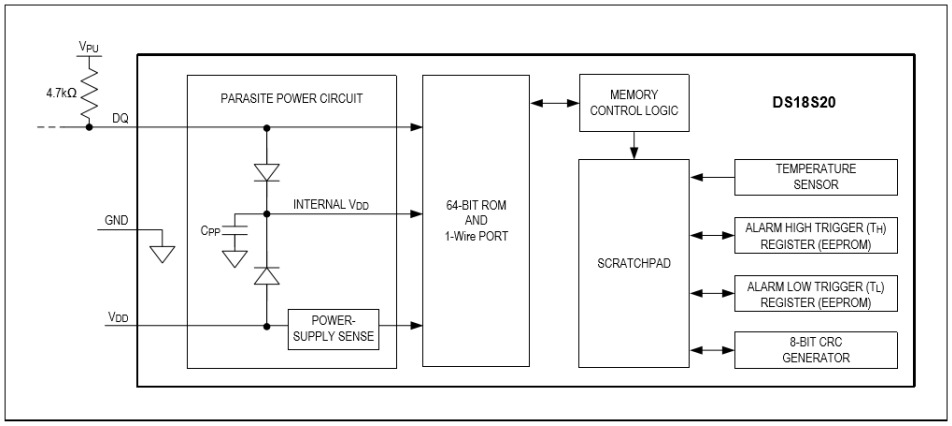
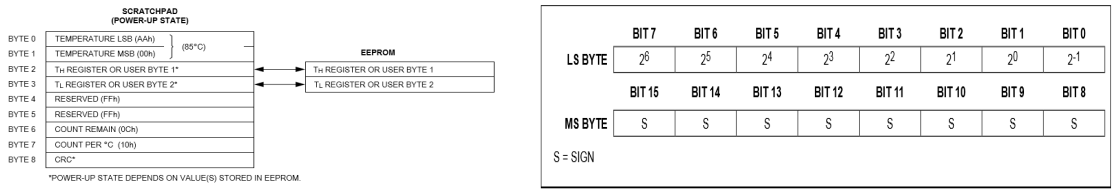


Figure 3.1: ds18s20+ block diagram

As shown in Fig 3.1, the temperature sensor contains one 64-bit ROM and a scratchpad memory which is shown in Fig 3.2a. The LSB and MSB of temperature are stored in the byte 0 and byte 1 of scratchpad separately. In the Fig 3.2b shows the format of the temperature. The bit 8 to bit15 are all sign bits, therefore, we only need the bit 8 to indicate the positive/negative of the value of temperature.



(a) (b)

Figure 3.2: details of ds18s20+.

3.2 The implementation of DS18S20+

3.2.1 State machine diagram

In Fig 3.3 shows the flow of the temperature sensor. Because only one temperature sensor is implemented, there is no need to read the ROM in ds18s20 +. After the initialization command which is shown in fig 3.4a, use the command CMD_CC to skip the ROM command and then write the CONV_44 conversion temperature command. After waiting for the temperature conversion delay, initialize ds18s20 + and skip the ROM command again and write CMD_BE to read the 9-bit temperature value in the temporary register. Finally back to initial operation again to complete a full temperature operation.

If we do not use the wireless components to send the outdoor temperature, there are two temperature sensors in the circuits, therefore, first, we need to read out the unique silicon serial number(64 bits for a unique ROM code and the first 8 bits for Ds18s20+: 10h). And the CMD_CC will be replaced by the Match_ROM, which is send a command to match the temperature sensors. For example, when we want to get the indoor temperature, we will match the indoor temperature sensor's serial number, and then, the indoor temperature sensor will work rather than the outdoor temperature sensor.

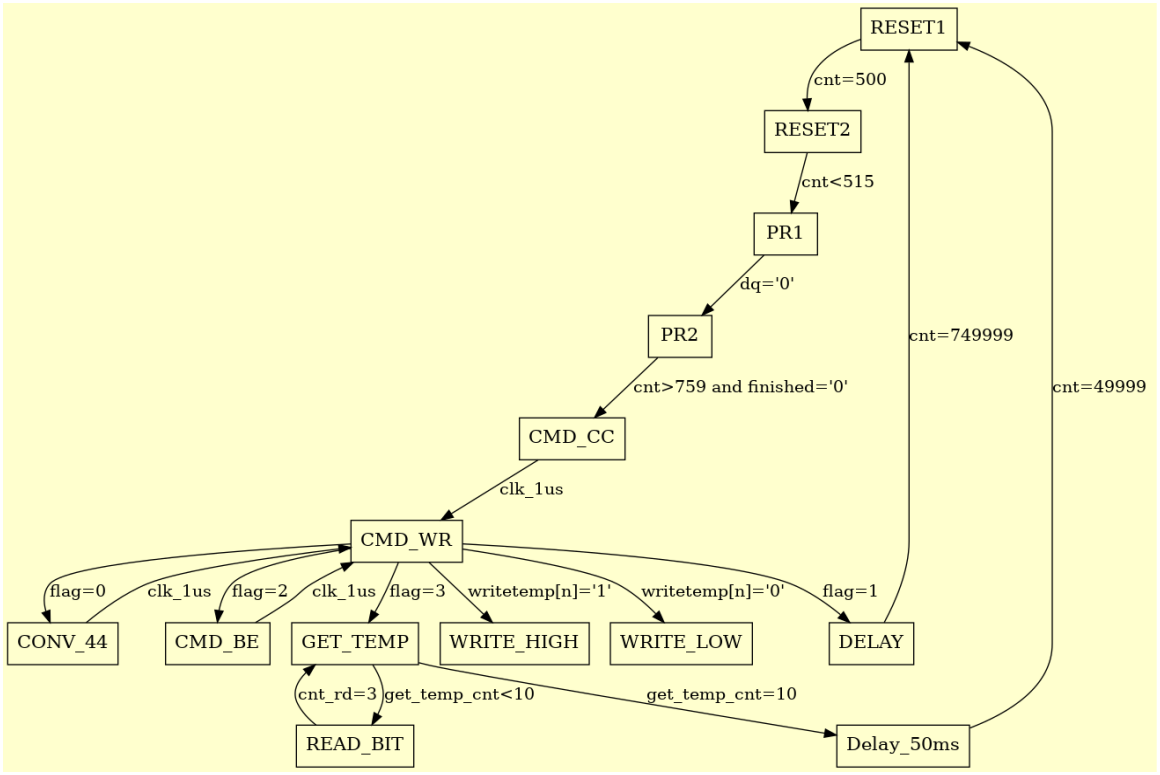
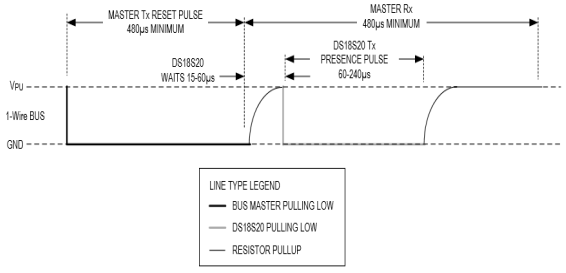
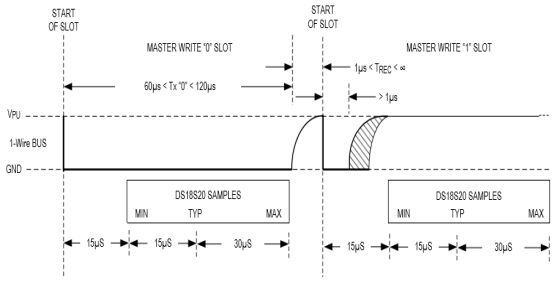


Figure 3.3: state machine of temperature

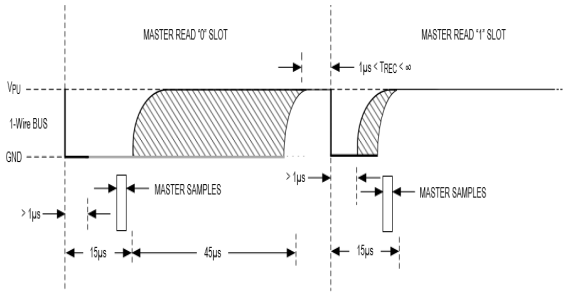
Those command are sent to the bus line by CMD_WR, WRITE_LOW and WRITE_HIGH which are shown in Fig 3.4a,Fig 3.4b. And the temperature conversion command is implemented by GET_TEMP and READ_BIT which is shown in Fig 3.4c.



(a) initialization



(b) write bits



(c) read bits

Figure 3.4: Time slot of ds18s20+.

3.3 Results of DS18S20+

Because the simulation is not readable for one picture, it do not show in this report, but the simulation is as we expect. After the master give initial signal to temperature sensor, it will give a present signal to pull down the bus line. And next, all states in the state machines will be visited to implement a full operation.

When we download our VHDL design to the hardware, the led will show the temperature, it shows as we expect. When I put my finger to the temperature sensor, the value of temperature will increase and when I move my finger, its value will decrease and stop at 22°C which is an indoor temperature.

3.4 Wireless transmission

This experiment is designed to test indoor and outdoor temperatures. If a wired transmission system is used, the complex wiring and buffer module need to be designed. Therefore, in this case, wireless systems that are not constrained by the environment and space have great advantages. This section will specifically describe how to apply the wireless system to this project.

3.4.1 Implementation

The wireless module is mainly composed of two parts: transmitting and receiving. Each part is composed of one FPGA board and one wireless component. The control module first encodes the data by the encoder and then sends the wireless signal by the RF(Radio Frequency) transmitter. After receiving the signal through the antenna, the receiver transmits the received binary data to the decoder, and then the decoder sends the decoded data to the top-level control module. The system block diagram is shown in Fig 3.5.

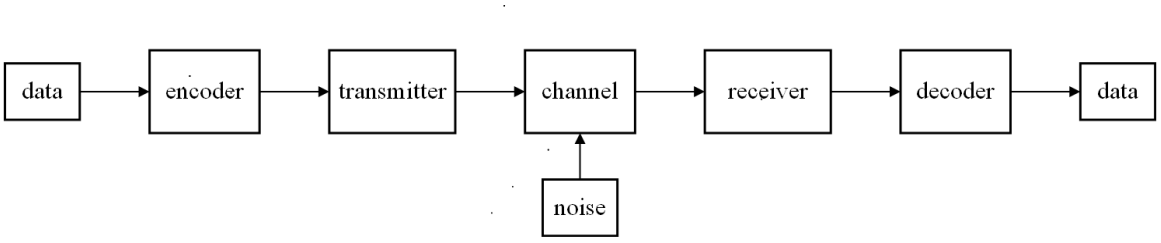


Figure 3.5: The block diagram of RF transmission

3.4.2 RF transmitter and receiver

After receiving the low-frequency signal, the transmitter has two tasks: modulation and amplification. Modulation refers to the use of the original electrical signal (modulation signal) to control a certain parameter (such as the change of the envelope) of a high-frequency oscillating signal (carrier signal) so that it changes with the variation of the modulation signal. In this way, the modulated modulated signal can be sent to the channel which is air in this project for transmission after being amplified and filtered.

The receiver can select the modulated signal from many signals and noise in the channel and processes it to restore the original electrical signal consistent with the transmission to the transmitter. After long distance transmission, the signal will become weak due to the loss of energy, so it needs to be amplified by the amplifier. And also, this signal needs to be processed by the filter to remove noise signals outside the operating frequency. The original signal is recovered by the demodulator, and then amplified by the amplifier and output.

The length of the antenna in wireless transmission is generally one-half or one-quarter the wavelength. The typical operating frequency of the transmitter and receiver we choose in this experiment is 315MHz or 433MHz. According to the formula $\lambda = \frac{c}{f}$(c is the speed of light in a vacuum), When the working frequency f is 433MHz, the wavelength λ is about 69.3cm, so we choose an antenna with a length around 17.3 cm.

3.4.3 Encoding and decoding

The data transmitted this project is a 9-bit binary data. The encoding of the data uses Manchester code, where the data '1' is represented by a high to low transition, and the data bit '0' is represented by a low to high transition. The duration of the positive and negative phases in '0' and '1' are the same. There is a start sequence in front of each data. As long as the decoder finds this start sequence, it considers that the nine-bit data after this sequence is the data that needs to be transmitted. In order to avoid confusion between the start sequence and the data to be transmitted, nine-bit data 000111110 was selected as the start sequence in this project.

4 The Control Unit, the brain of every Integrated Circuit

It is the Main Control Unit of the overall design, and it is in charge of scheduling all the operations in a precise order, starting from a temperature measurement to display the value on an LCD. An important point to mention is that the Control Unit has been design in order to implement concept of handshake signal for synchronizing the operations between itself and the interfaces which use an extreme slow clocks compared with the one used from the Control Unit (100 Mhz).
As every integrated circuit, there will be an initial phase (set up) in which the FSM will wait until the desired clock has been reached (in the case of using a Phase Loop Lock) and then it will start the real initialization phase. This is important especially for the LCD in order to show correctly the output, therefore in the set up phase it will wait until the done signal from the LCD arrives.
For making the design tolerant to possible error into the initialization phase of the LCD, a watchdog counter has been added in order to restart the initialization phase after 200 ms in the case the done signal from the LCD interface is not received.
The Control unit will wait in the idle state until there will be a rising/falling edge in the in the indoor/outdoor selection signal (a physical switch) or it will refresh the temperature if a 10 second counter will expire.
Moving from the idle state, a measurement request to the temperature interface will be requested, as soon as the done signal from the temperature interface will arrive it will move to the computation of the new maximum and minimum value of the temperature (activating the signal for that sub-block). Completed the evaluation of the maximum and minimum temperature, they will be shown on the LCD. In order to do that, the data (maximum, minimum or current value) will be selected through a multiplexer in the comparison entity and the printing phase for the LCD will be activated. A detection of a rising edge in the done signal of the LCD is followed by a detection of a falling edge on the same signal before printing any others values. This is principally done because the of the different clock frequency between the CU and LCD interface. Actually, the done signal will be activated for 2 ms, this means that if the CU would not wait for the falling edge it will sample the done signal 200000 This operation will be done for each and every value before return to the idle phase.

[1] edge detect="10" or edge_detect_com="10" or edge detect="01" or edge_detect_com="01"

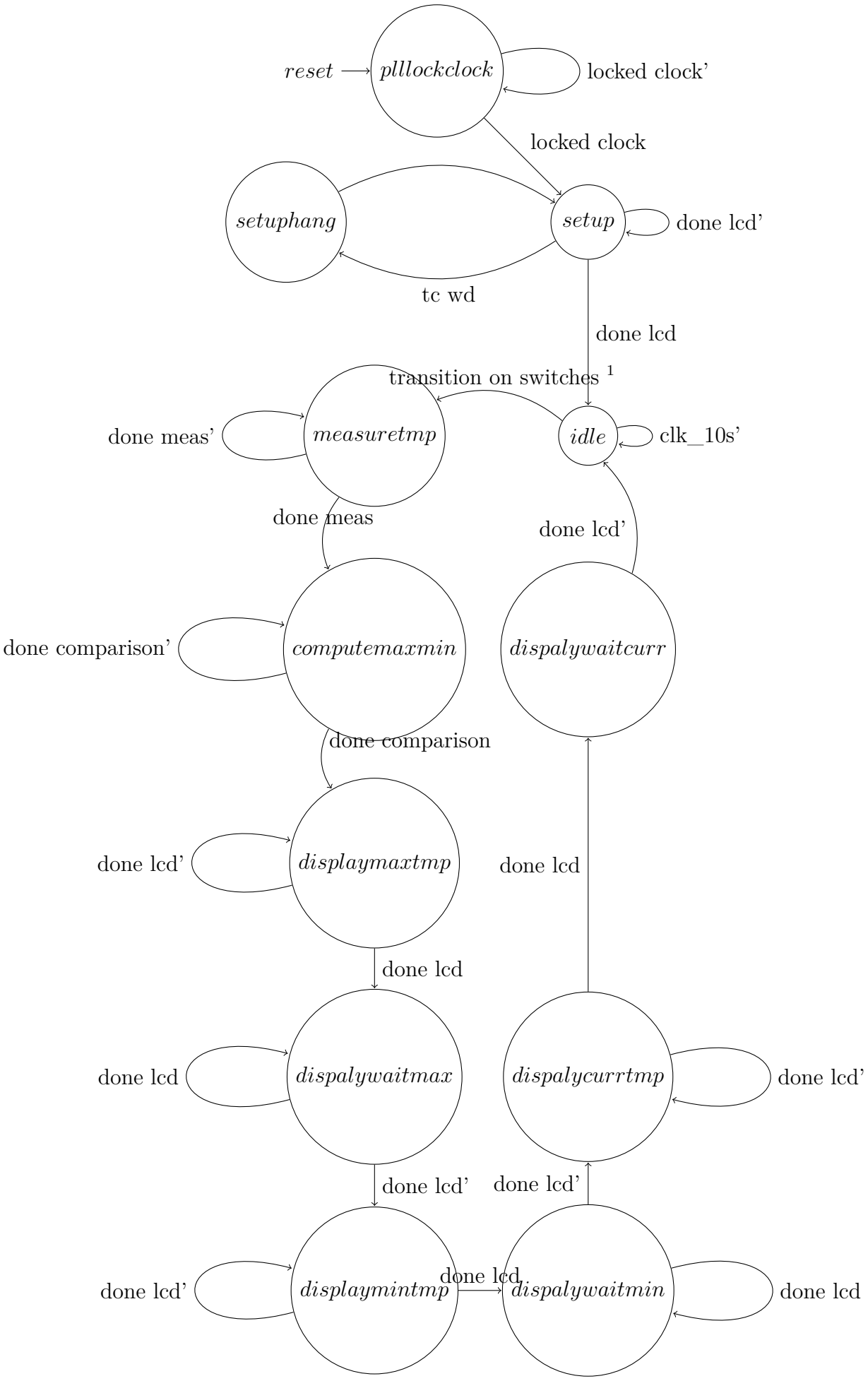


Figure 4.1: State Diagram, Active Signals into states are in Appendix

5 Comparison block

The function of the comparison block is to output the historical maximum and minimum temperature values. There are two comparators to compare indoor and outdoor temperatures separately. The stored maximum and minimum values set to default value when reset .

The work flow of the comparator is shown in fig 5.1. When new data is transmitted, it will be compared with the historical maximum and minimum respectively. If it is higher than the maximum or lower than the minimum, the maximum and minimum data will be updated. Otherwise, the internal register are not updated.

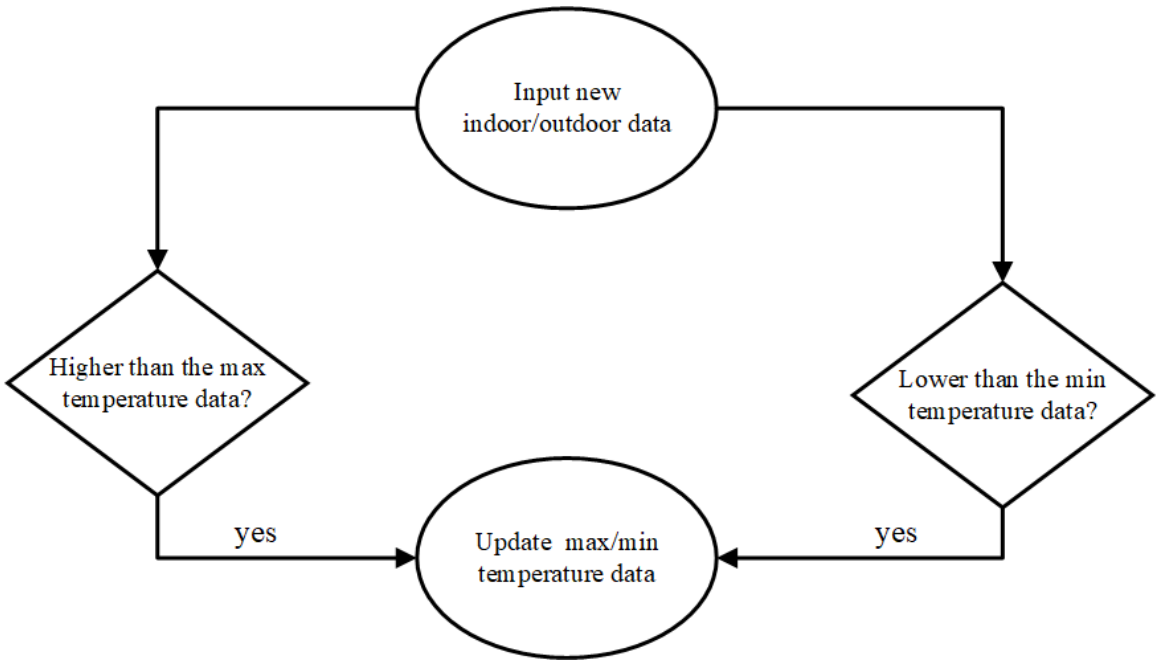


Figure 5.1: The work flow of the comparator

6 LCD interface

No single electronic device makes sense without a module that gives the user feedback and allows the control of it, showing the data the system works with, and that is precisely the role the LCD plays in this project.

Taking the coming data from the comparison block, what means temperature values according to the customer's choice (current indoor and outdoor compared with the registered ones) and with the help of the control unit (CU), the temperature will be printed out for the user to read it.

6.1 Initialization process

General behaviour known, a deeper explanation of the LCD and its interface is needed. Like many electronic components of this kind, the LCD needs a long and accurate initialization process to be ready, see the following Figure 6.1

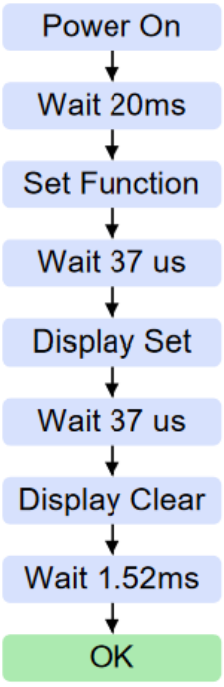


Figure 6.1: LCD initialization

This process is carried out in the first states, as it has to be done right after powering up. All the timing will be regulated by a clock enable of 2ms, and through counters, with values some needed times the 2ms one (example: 20ms would be 10 enable times 2ms).

With respect to the data sent in those states, in the component sheet the different possible configurations are stated. Some of the implemented characteristics are: 16x2, two lines, cursor showed or no display shifting.

Initialization finished, the LCD is ready to print.

6.2 Printing

One peculiarity of the device is that it works with ascii code as an input, what means the data should be in that format for it to be shown properly.

For this precise mission, fixed and dynamic data can be distinguished for speeding up the process.
The words written in the screen just for making the reading task easier, or for preparing the information that it is going to be presented (example: "Maximum: ") are fixed and can be written beforehand in ascii code directly.

Nevertheless, the read actual temperature values change every time, so a translator is needed to handle this variation.

Again, when it comes to showing the temperature, a dividing strategy is a convenient way. Then, three different parts of the number to print can be seen: the sign (+ or -), the whole part and the decimal part.

For the first one, the most significant bit is checked, if it is a 1 the number is negative and '-' is applied, otherwise it is positive '+'.
Regarding to the whole part, three digits (enough information) will be presented, obtained through divisions by ten (one position shifting) for the quotient and using the module command for the remainder.
For ending, the decimal digit will be either .5 or .0 if the least significant bit is either 1 or 0, respectively.

7 Humidity Sensor Interface

The humidity sensor interface is in charge of reading the humidity value from the sensor, adapting it to a percentage value and dynamically change the duty cycle of the PWM generator in order to vary the brightness of the Green LED.

7.1 I2C interface and protocol

In order to communicate with the humidity sensor, the I2C protocol must be used.
According to the timing specifications of the sensor a clock of 100 KHz (Appendix - Critical Signals Fig. 10.1,10.2). will be created and used for the Finite State Machine and the *SCLK* signal.
At the power up of the entire system, before every transaction on the bus can start it will wait for 1 second. Then, the idle state will be reached waiting for a terminal count signal from a counter(counting up to 100 ms) which will start a measure request to the sensor. This will start the first transaction on the bus:

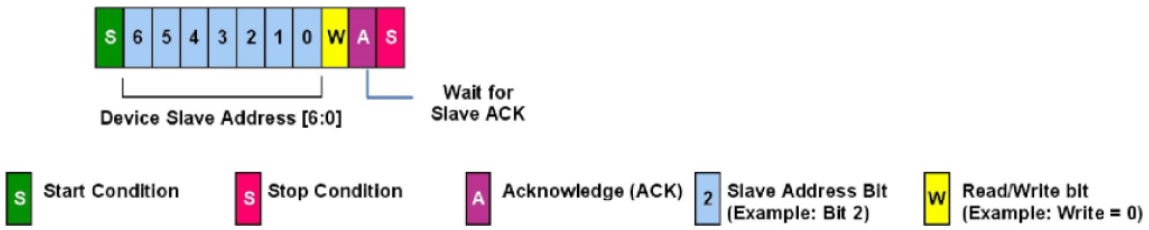


Figure 7.1: Measure Request Command

Using as address the default value 0x28, the measurements will start when the slave sends the acknowledge. With just one master and one slave there is no need of arbitration on *SDATA* neither on *SCLK* because the lowest value of the frequency has been used.
As soon as the measuring request will be done, the interface will wait for 90 seconds for the humidity measurement. Then, the data fetch phase will be done according to Figure 7.2.

The slave is also capable of measuring the temperature and since it is not needed, the master will stop the communication after it will receive the second byte of Humidity (otherwise the slave will send also the temperature value on the bus). The data on the

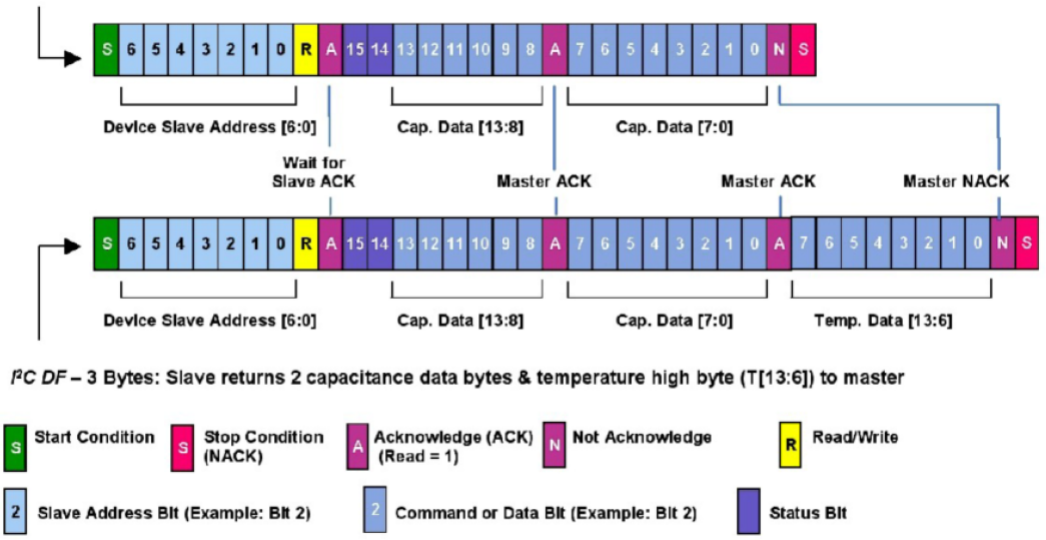


Figure 7.2: Data Fetch Command

bus will be sampled during the rising edge of *SCLK* and in order to achieve that, the output clock will be the negated version of the internal clock (values are kept for one clock cycle).

Regarding the external connection(they will be Open Drain outputs from the point of view of the FPGA), the I2C protocol demands of two pull up resistances for keeping the line at the power supply (high logic state) when none is writing on the bus.

$$R_{PU} = \frac{V_{dd}}{(i_{Omax})} = 275\Omega \; V_{dd} = 3.3V, i_{Omax} = 12mA$$

A standard resistance value of 330 Ω has been chosen while the maximum output current is selected from FPGA I/O configuration.

7.2 PWM generator

The PWM generator has been designed accordingly to the following figure:

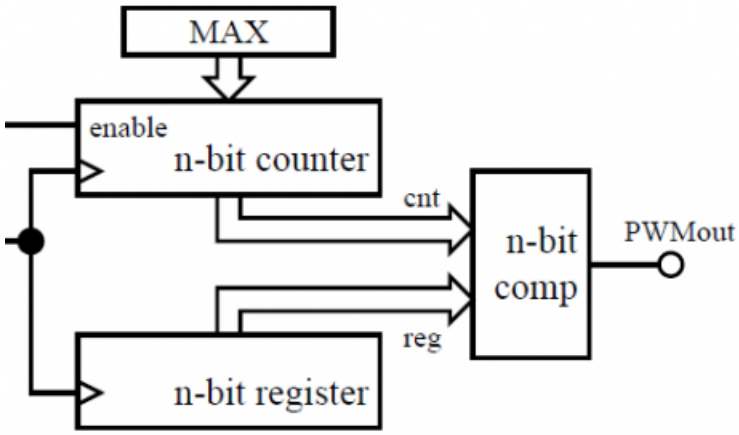


Figure 7.3: PWM structural view

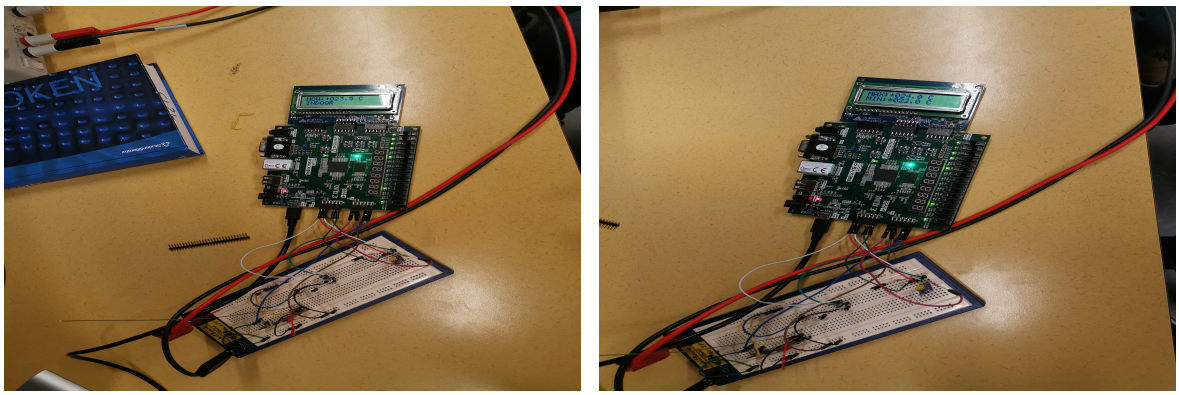
It is a simple entity for which at the output a signal with varying duty cycle and frequency can be created. The human eyes are not able to see oscillations greather than 50 Hz, therefore in order to mimic different brightness a PWM frequency of 1 KHz has been chosen (actually, the LED will be switched off and switched). Thus, according to the formulas:

$$f_{pwm} = \frac{f_{clk}}{(MAX + 1)} \quad \sigma = \frac{reg}{Max + 1}$$

the maximum value will be calculated and also the correct value of the duty cycle register at every update (the default duty cycle is 50 %).

8 Summary

This report describes a digital thermometer that can measure indoor and outdoor temperatures. This digital thermometer can start measuring temperature each time it is turned on (reset = '0'), transmit or receive temperature through wireless transmission, control the LCD screen to display indoor or outdoor temperature through switch, and display real-time or maximum and minimum temperature. The real-time temperature is refreshed every ten seconds, and the accuracy error is within 0.5 °C. It only takes 2 seconds to complete a temperature measurement. Humidity sensors and voice modules were also added to this project, but for some reasons, we were unable to complete these successfully.



(a) (b)

Figure 8.1: temperature value from DS18S20+

Regarding the humidity sensor, in the following figure it can be seen that the protocol has been implemented correctly. However, the master does not receive the acknowledge signal from the slave.

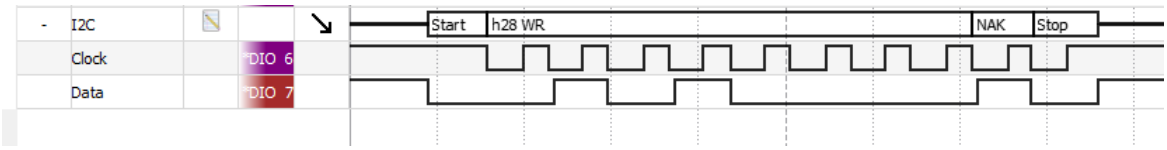


Figure 8.2: Measuring request for Humidity Sensor

9 Appendix - Active Signals in CU states

- pll lock clock: None
- set up: init_set_up , enable_wd, ready, display
- set up hang: reset_i
- idle: ready, enable_humidity_sensor
- measure tmp: ready, start_meas, enable_humidity_sensor
- compute_max_min: ready, start_comparison, enable_humidity_sensor
- display_max_tmp: select_data="01", ready, display, enable_humidity_sensor
- display_wait_max:enable_humidity_sensor, ready, display, select_data="01"
- display_min_tmp: select_data="10", ready, display, enable_humidity_sensor
- display_wait_min:select_data="10", ready, display, enable_humidity_sensor
- display_curr_tmp: ready, display, enable_humidity_sensor
- display_wait_now:ready, display, enable_humidity_sensor

10 Appendix - Critical Signals

10.0.1 Timing specification for I2C interface for Humidity Sensor

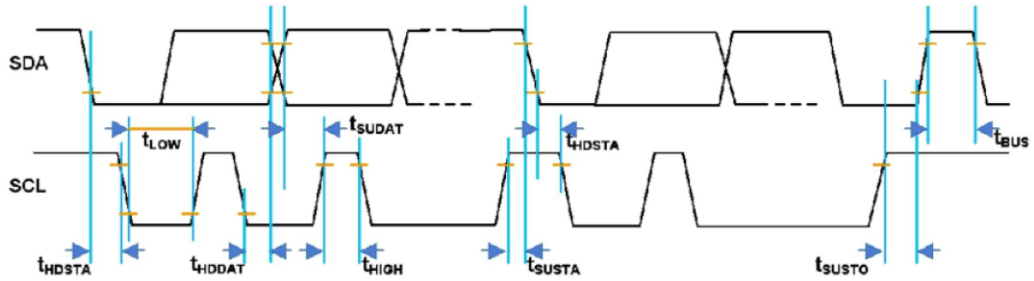
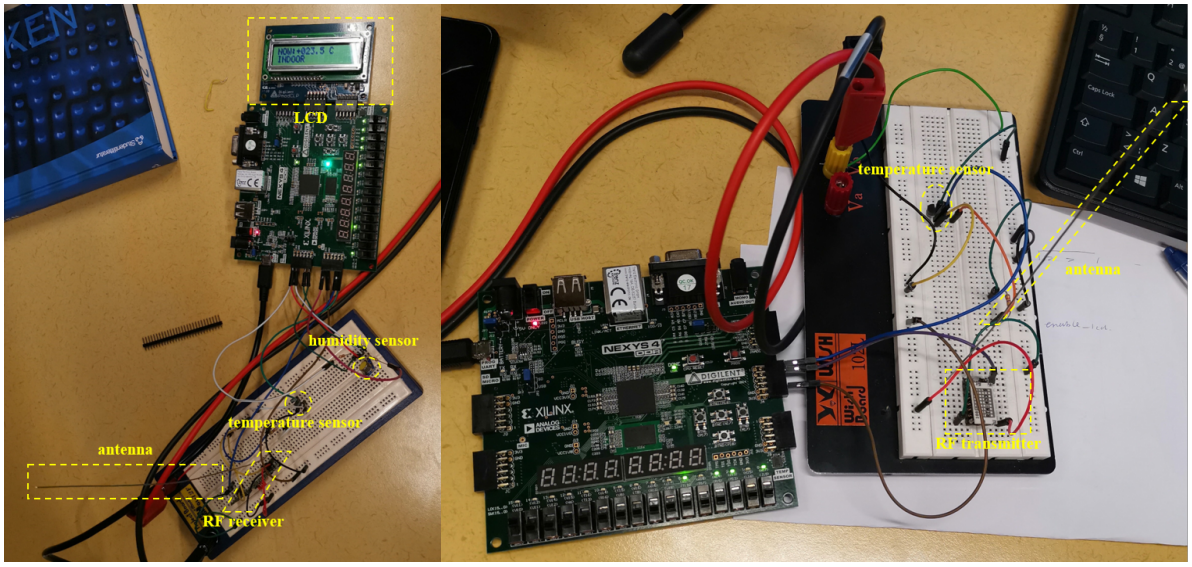


Figure 10.1: I2C Timing Diagram

PARAMETER	SYMBOL	MIN	MAX	UNIT
SCL clock frequency	fSCL	100	400	kHz
Start condition hold time relative to SCL edge	tHDSTA	0.1		µs
Minimum SCL clock low width 1	tLOW	0.6		µs
Minimum SCL clock high width 1	tHIGH	0.6		µs
Start condition setup time relative to SCL edge	tSUSTA	0.1		µs
Data hold time on SDA relative to SCL edge	tHDDAT	0		µs
Data setup time on SDA relative to SCL edge	tSUDAT	0.1		µs
Stop condition setup time on SCL	tSUSTO	0.1		µs
Bus free time between stop condition and start condition	tBUS	1		µs

Figure 10.2: I2C Requirements

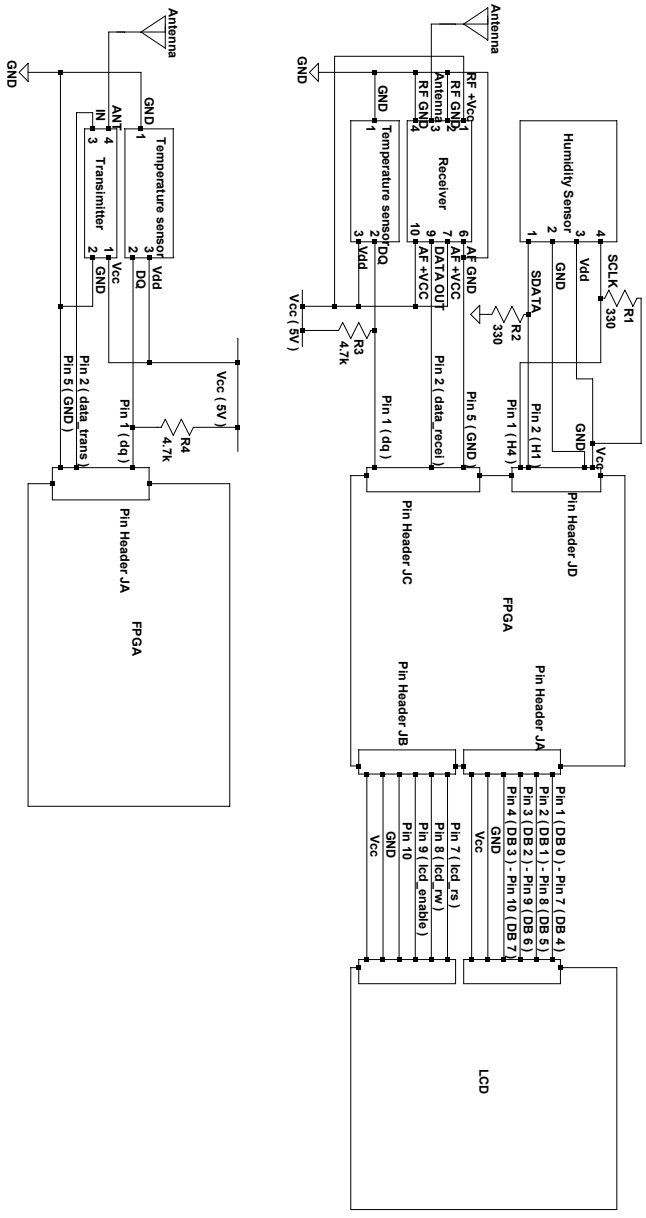
11 Appendix - System



(a) indoor system (b) outdoor system

Figure 11.1: whole system implementation

12 Appendix - Circuit Schematic



13 Appendix - VHDL code

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


entity top_thermometer is
    Port ( reset : in  STD_LOGIC;        --SW[15]
           clk : in  STD_LOGIC;
                  ind_outd_sw: in STD_LOGIC;        --SW[0]
----------TEMPERATURE-------------------------
           tmp_sensor : inout  STD_LOGIC;--dq indoor JC[1]
                  data_receive_from_rece: in std_logic;--JC[2]
                  ledtest: out std_logic_vector(8 downto 0);
                  finishtem: out std_logic;
--------------------LCD---------------------------------
           lcd_rs : out  STD_LOGIC;        --JB[7]
           lcd_rw : out  STD_LOGIC;        --JB[8]
           lcd_enable : out  STD_LOGIC;        --JB[9]
           lcd_data : out  STD_LOGIC_VECTOR (7 downto 0);--JA[0] --JA[10]
                  temstart: out std_logic;
                  swtem:          in std_logic;
--                   led: out std_logic_vector( 8 downto 0);
                     compare_start: out std_logic;
                     compare_done: out std_logic;
        --                 generate_enable: out std_logic;
                  system_ready: out std_logic; -- signal to an led on the board
-------humidity sensor---------------------------------
           pwm_out: OUT std_logic;
           sclk,sdata: INOUT std_logic
                              );
end top_thermometer;

architecture structural of top_thermometer is
```

```
 -- main control unit
 COMPONENT control_unit is
GENERIC ( WATCH_DOG_COUNT: integer:= 100);
PORT ( reset,clk: IN std_logic;
-- selecting the indoor or outdoor temp sensor
in_out_sel: IN std_logic;
-- commond signal to all component for setting up the lcd and sensors
init_set_up: OUT std_logic ;
swtem:         in std_logic;
-------------------------------------------------------------------------
-- done signals for each and every component will be  ----------------
-- also used for understanding that we are out of the set up phase ---
-------------------------------------------------------------------------

-- common signal to all component for indoor outdoor sensor
in_out:OUT std_logic;
-- from/to comparison
start_comparison : OUT std_logic;
done_comparison: IN std_logic;
-- selecting between max,min and curr tmp
select_data: OUT std_logic_vector(1 DOWNTO 0);
-- from/to lcd interface
display: OUT std_logic;
done_lcd: IN std_logic;
--led: out std_logic_vector( 8 downto 0);
-- from/to sensor interface
start_meas: OUT std_logic;
done_meas:IN std_logic;
-- internal reset for all interfaces
reset_i: OUT std_logic;
-- switch on an led for notifyinh that the system is operative
ready: OUT std_logic;
-- notify from pll that clock speed has been reached
locked_clock: IN std_logic ;
-- to humidity sensor interface
enable_humidity_sensor: OUT std_logic
```

```
);
end COMPONENT control_unit;

 --humidity sensor
component  top_humidity_sensors_interface is
    Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           enable : in  STD_LOGIC;
           pwm_out : out  STD_LOGIC;
           sclk : inout  STD_LOGIC;
           sdata : inout  STD_LOGIC);
end component top_humidity_sensors_interface;


 -- datapath ( interfaces have its own CU )
 -- comparison

component comparison is
GENERIC (N:integer:= 8);
    Port ( clk,reset: in STD_LOGIC;
                        in_out_sel: in std_logic;
                data_in : in  STD_LOGIC_VECTOR (N-1 downto 0);
         data_outmax : out  STD_LOGIC_VECTOR (N-1 downto 0);
            data_outmin : out  STD_LOGIC_VECTOR (N-1 downto 0);
        start_comparison : in  STD_LOGIC; -- aka enable
        done_comparison : out  STD_LOGIC);
end component comparison;


-- termometer interfaces

component temall is
port        (clk : in std_logic;
                nrst:in std_logic;
                in_out: in std_logic;
                led: out std_logic_vector(8 downto 0);
                dq:  inout std_logic;             --ja[1]
               ledtest: out std_logic_vector(10 downto 0);
                data_recei :in std_logic;         --ja[2]
                finish:         out std_logic
```

```
                );
end component temall;

-- lcd interfaces

COMPONENT LCD is
        PORT(
                clk:            IN STD_LOGIC;
                reset:          IN STD_LOGIC;
                   dataIN:          IN STD_LOGIC_VECTOR(8 downto 0);
                datacompare_max:        in std_logic_vector(8 downto 0);
                datacompare_min:        in std_logic_vector(8 downto 0);
                enable_init:    IN STD_LOGIC;
                -- notify from pll that clock speed has been reached
                enable:         IN STD_LOGIC;
                ind_outd_select:IN STD_LOGIC;
                swtem:                          in std_logic;
--                generate_enable: out std_logic;
                enable_lcd: OUT std_logic;
                dataOUT:        OUT STD_LOGIC_VECTOR(7 downto 0);
                done:                OUT STD_LOGIC;
                 -- 0 instruction register (write) /  1 (write, read)
                RS:             out STD_LOGIC;
                R_W:            OUT STD_LOGIC  );

end  COMPONENT LCD;



SIGNAL reset_top,start_display,reset_start_tmp,init_set_up,
                        in_out_sel ,start_comparison,done_display,
                        done_comparison,done_meas,
                        start_meas,reset_i: std_logic;
SIGNAL select_data_comparison: std_logic_vector(1 DOWNTO 0);
SIGNAL data_from_comparison_max,data_from_comparison_min,
                data_from_tmp_interface: std_logic_vector(8 DOWNTO 0);
signal datainter : std_logic_vector(8 DOWNTO 0);
signal clk_10mhz,enable_humidity_sensor: std_logic;
begin
datainter<=data_from_tmp_interface;
```

```
reset_top<= reset ;
-- interconnecting components

interface_lcd: LCD  PORT MAP(clk=>clk,reset=>reset_top,
                        dataIN=>data_from_tmp_interface,--data_from_comparison,
                        enable_init=>init_set_up,
                        enable=>start_display,
                        ind_outd_select=>in_out_sel,
                        swtem=>swtem,
                        datacompare_max=>data_from_comparison_max,
                        datacompare_min=>data_from_comparison_min,
                        done=>done_display,
                        dataOUT=>lcd_data,
                        enable_lcd=>lcd_enable,
                        R_W=>lcd_rw,
--                         generate_enable=>generate_enable,
                        RS=>lcd_rs );
interface_tmp_rf_sensor: temall PORT MAP                (clk=>clk,
                         nrst=>start_meas,
                        in_out=>in_out_sel,
                        led=>data_from_tmp_interface,
                        dq=> tmp_sensor,            --ja[1]
--                         ledtest=>ledtest,
                        data_recei=>data_receive_from_rece,--ja[2]
                        finish=>done_meas);
comparison_block:comparison GENERIC MAP (9)
        PORT MAP(clk=>clk,reset=>reset_top,
                        in_out_sel=>ind_outd_sw,
                        data_in=>datainter ,
                        data_outmax=>data_from_comparison_max ,
                    data_outmin=>data_from_comparison_min ,
     --      data_out=>data_from_comparison ,
            start_comparison=>start_comparison ,
            done_comparison=>done_comparison
            --select_data =>select_data_comparison
                  );
cu: control_unit GENERIC MAP (200000000)
                        PORT MAP(locked_clock=>'1',
                                clk=>clk,reset=>reset,
                                in_out_sel=>ind_outd_sw,
```

```
                                    init_set_up=> init_set_up,
                                    swtem=>swtem,
                                    in_out=>in_out_sel,
                                    start_comparison=> start_comparison,
                                    done_comparison=> done_comparison,
                                    select_data=>select_data_comparison,
                                    display=>start_display,
                                    done_lcd=>done_display,
                                    start_meas=>start_meas,
                                    done_meas=>done_meas ,
                                    reset_i=>reset_i,
                                    ready=> system_ready,
                                    enable_humidity_sensor=>enable_humidity_sensor);

reset_start_tmp<=not(start_meas or reset_top);
ledtest<=data_from_tmp_interface;
temstart<=start_meas;
finishtem<=done_meas;
compare_start<=start_comparison;
compare_done<=done_comparison;

humidity_interface: top_humidity_sensors_interface
                                              PORT MAP (clk=>clk,reset=>reset_top,
                                pwm_out=>pwm_out,enable=>enable_humidity_sensor,
                                sdata=>sdata,sclk=>sclk);
end structural;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.math_real."log2";
use IEEE.math_real."ceil";


entity control_unit is
GENERIC ( WATCH_DOG_COUNT: integer:= 100);
PORT ( reset,clk: IN std_logic;
-- selecting the indoor or outdoor temp sensor
in_out_sel: IN std_logic;
--led: out std_logic_vector( 8 downto 0);
-- commond signal to all component for setting up the lcd and sensors
init_set_up: OUT std_logic ;
swtem:         in std_logic;
-------------------------------------------------------------------------------
-- done signals for each and every component will be also    --------------------
--        used for understanding that we are out of the set up phase ---
-------------------------------------------------------------------------------
-- common signal to all component for indoor outdoor sensor
in_out:OUT std_logic;
-- from/to comparison
start_comparison : OUT std_logic;
done_comparison: IN std_logic;
-- selecting between max,min and curr tmp
select_data: OUT std_logic_vector(1 DOWNTO 0);
-- from/to lcd interface
display: OUT std_logic;
done_lcd: IN std_logic;
-- from/to sensor interface
start_meas: OUT std_logic;
done_meas:IN std_logic;
-- internal reset for all interfaces
reset_i: OUT std_logic;
-- switch on an led for notifyinh that the system is operative
ready: OUT std_logic;
-- notify from pll that clock speed has been reached
locked_clock: IN std_logic;
-- to humidity sensor interface
enable_humidity_sensor: OUT std_logic
```

```
);
end entity control_unit;


architecture Behavioral of control_unit is
type  state_t is (pll_lock_clock,set_up,set_up_hang,idle,
        measure_tmp,compute_max_min,display_curr_tmp,display_max_tmp,
        display_min_tmp,display_wait_min,display_wait_max,display_wait_now);

SIGNAL curr_state,next_state: state_t;
SIGNAL curr_in_out_val,next_in_out_val: std_logic:='0';
SIGNAL tc_wd,enable_wd,clk_10s: std_logic;

-- component declaration it works like a watchdog timer
-- for the setup phase and the idle period ( for refreshing the temperature)
component clock_enable_10s is
port( clk: in std_logic;
          clk_10s: out std_logic);
end component;

component counter_wd is
generic ( N : integer := 8;
                        MAX_VAL: integer :=255);
    Port ( clk : in  STD_LOGIC;
                                           enable: in std_logic;
           reset : in  STD_LOGIC;
           tc : out  STD_LOGIC);
end component counter_wd;

signal edge_detect : std_logic_vector( 1 downto 0 );
signal edge_detect_com:std_logic_vector( 1 downto 0 );
begin
watch_dog: counter_wd GENERIC MAP
                        (N=> 1+integer(ceil(log2(real(WATCH_DOG_COUNT)))),
                                              MAX_VAL=>WATCH_DOG_COUNT)
                        PORT MAP(enable=>enable_wd,clk=>clk,reset=>reset,tc=> tc_wd);

clk_com:component clock_enable_10--refreshing temperature
        port map ( clk=>clk, clk_10s=>clk_10s);
```

```
regs:PROCESS(clk,reset)
BEGIN
IF (reset='1') THEN
curr_state<=pll_lock_clock;
curr_in_out_val<='0';
ELSE
        IF(clk='1' AND clk'EVENT) THEN
            edge_detect <= edge_detect(0) &in_out_sel ;
                edge_detect_com <= edge_detect_com(0) &swtem ;
        curr_state<=next_state;
        curr_in_out_val<=next_in_out_val;
        END IF;

END IF;
END PROCESS regs;


comb_logic:PROCESS(locked_clock,in_out_sel,done_comparison
                                ,done_lcd,done_meas,curr_state,
                                tc_wd,edge_detect,curr_in_out_val)
BEGIN
-- default assignments of all signal
next_state<=curr_state;
display<='1';
select_data<="00";
next_in_out_val<=curr_in_out_val;
start_comparison<='0';
init_set_up<='0';
enable_wd<='0';
start_meas<='1';
reset_i<='0';
ready<='1';
in_out<=curr_in_out_val;
enable_humidity_sensor<='1';
CASE curr_state IS
WHEN pll_lock_clock=>         enable_humidity_sensor<='0';
                                            ready<='0';
                    --    led(0)<='1';
                                            enable_wd<='1';
```

```
                                                IF (locked_clock='1') THEN
                                                -- desired frequnecy reached
                                                next_state<=set_up;
                                                ELSE
                                                next_state<=curr_state;
                                                END IF;
WHEN set_up=> init_set_up<='1';
                                    enable_wd<='1';

                                    -- led(1)<='1';
                                    enable_humidity_sensor<='0';
--                            ready<='0';
                                -- interfaces will maintain the done signals up
                                 --( if they have completed the initialization )
                                 -- as soon as the init_set-up remains at 1
                                IF ( done_lcd='1' ) THEN
                                next_state<=idle;
                                ELSIF ( tc_wd='1' ) THEN
                                next_state<=set_up_hang;
                                ELSE
                                next_state<=curr_state;
                                END IF;
WHEN set_up_hang=>
 -- tear downt the initialization signal for one clock cycle
        enable_humidity_sensor<='0';
            --                      led(2)<='1';
--        ready<='0';
        reset_i<='1';
        next_state<=set_up;
WHEN idle=> enable_wd<='1';
    start_meas<='0';
        display<='0';

    --led(3)<='1';
        IF( edge_detect="01" or edge_detect_com="01") THEN
        -- rising edge  INDOOR
        next_state<=measure_tmp;
        -- keeping constant until next idle period
        next_in_out_val<=in_out_sel;
        ELSIF (edge_detect="10" or edge_detect_com="10") THEN
```

```
        -- falling edge outdoor
        next_state<=measure_tmp;
        -- keeping constant until next idle period
        next_in_out_val<=in_out_sel;
        ELSIF ( clk_10s='1') THEN
        next_state<=measure_tmp;
        -- keeping constant until next idle period
        next_in_out_val<=in_out_sel;
        ELSE
        next_state<=curr_state;
        END IF;
WHEN measure_tmp=>--led(4)<='1';
        display<='0';
        IF(done_meas='1')THEN
        next_state<=compute_max_min;
        ELSE
        next_state<=curr_state;
        END IF;
WHEN compute_max_min=>--led(5)<='1';
        start_meas<='0';
        start_comparison<='1';
        display<='0';
        IF(done_comparison='1') THEN
        next_state<=display_max_tmp;
        ELSE
        next_state<=curr_state;
        END IF;
WHEN display_max_tmp=>
        --display<='0';
        start_meas<='0';
        select_data<="01";
        IF ( done_lcd ='1') THEN
        next_state<=display_wait_max;
        ELSE
        next_state<=curr_state;
        END IF;

WHEN display_wait_max=>
        --display<='0';
        select_data<="01";   start_meas<='0';
```

```
        IF ( done_lcd ='0') THEN
        next_state<=display_min_tmp;
        ELSE
        next_state<=curr_state;
        END IF;

WHEN display_min_tmp=>
        select_data<="10";
        start_meas<='0';
--led(7)<='1';
        --        display<='1';
        IF ( done_lcd ='1') THEN
        next_state<=display_wait_min;
        ELSE
        next_state<=curr_state;
        END IF;
WHEN display_wait_min=>
        --display<='0';
        select_data<="10";
        start_meas<='0';
        IF ( done_lcd ='0') THEN
        next_state<=display_curr_tmp;
        ELSE
        next_state<=curr_state;
        END IF;
WHEN display_curr_tmp=>
         --led(8)<='1';
        --select_data<="11";
        start_meas<='0';
        --                display<='1';
        IF ( done_lcd ='1') THEN
        next_state<=display_wait_now;
        ELSE
        next_state<=curr_state;
        END IF;
WHEN display_wait_now=>
        --display<='0';
        --select_data<="11";
        start_meas<='0';
        IF ( done_lcd ='0') THEN
```

```
        next_state<=idle;
        ELSE
        next_state<=curr_state;
        END IF;
-- for a safe fsm
WHEN OTHERS=>
        next_state<=set_up;
        start_meas<='0';display<='0';select_data<="00";
        in_out<='0';start_comparison<='0';init_set_up<='0';enable_wd<='0';
        enable_humidity_sensor<='0';
reset_i<='0';
END CASE;

END PROCESS comb_logic;

end Behavioral;
```

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

ENTITY  LCD is
        PORT(
                clk:            IN STD_LOGIC;
                reset:          IN STD_LOGIC;
                   dataIN:          IN STD_LOGIC_VECTOR(8 downto 0);
                datacompare_max:        in std_logic_vector(8 downto 0);
                datacompare_min:        in std_logic_vector(8 downto 0);
                enable_init:    IN STD_LOGIC;
                enable:         IN STD_LOGIC; -- called display in control unit
                ind_outd_select:IN STD_LOGIC;
                swtem:                          in std_logic;
                enable_lcd: OUT std_logic;
                dataOUT:        OUT STD_LOGIC_VECTOR(7 downto 0);
                done:                OUT STD_LOGIC;
        --        generate_enable: out std_logic;
         -- 0 instruction register (write) /  1 (write, read)
                RS:             out STD_LOGIC;
                R_W:            OUT STD_LOGIC  );
END ENTITY  LCD;



architecture beh of LCD is

----------------------state----------------------
type statetype is (power_up,idle,clear,setmode,
                switchmode,setfunction,        first_line,second_line,
                writing_max_1,writing_max_2,writing_max_3,
                writing_max_4,writing_max_5,writing_max_6,
                writing_max_7, writing_max_8,writing_max_9,
                writing_max_10,writing_max_11,writing_max_12,
                writing_min_1,writing_min_2,writing_min_3,
                writing_min_4,writing_min_5,writing_min_6,
                writing_min_7,writing_min_8,writing_min_9,
                writing_min_10,writing_min_11,writing_min_12,
```

```
                writing_now_1,writing_now_2,writing_now_3,
                writing_now_4,writing_now_5,writing_now_6,
                writing_now_7,writing_now_8,writing_now_9,
                writing_now_10,writing_now_11,writing_now_12,
                print_ind_1,print_ind_2,print_ind_3,
                print_ind_4,print_ind_5,print_ind_6,
                print_outd_1,print_outd_2,print_outd_3,
                print_outd_4,print_outd_5,print_outd_6,print_outd_7,
                print_ind_outd_select,wait_4sec,first_line_second_print,
                done_print ,reset_screen,delay2ms,delay10ms,displayoff);

---------------------
signal state : statetype;
signal clk_2ms,generate_enable_lcd: std_logic:='0';
signal enable_translator: std_logic:='0';
signal cnt_e : std_logic_Vector(15 downto 0);
signal data_item: std_logic_vector(8 downto 0);
--signal dataout: std_logic_Vector( 7 DOWNTO 0);
signal cnt20ms: natural range 0 to 13 :=0;
--signal cnt4sec: natural range 0 to 2000-1;
signal flag: natural range 0 to 5 :=0;
signal dataout1, dataout2, dataout3, dataout4,
         dataout5, dataout6: std_logic_vector (7  downto 0);
signal dataout1_max,dataout2_max, dataout3_max,
            dataout4_max, dataout5_max,
                 dataout6_max: std_logic_vector (7  downto 0);
signal dataout1_min,dataout2_min, dataout3_min,
                dataout4_min, dataout5_min,
            dataout6_min: std_logic_vector (7  downto 0);

COMPONENT translator is
GENERIC ( n: integer:= 9);
    Port ( reset : in  STD_LOGIC;
    clk : in  STD_LOGIC;
    start : in  STD_LOGIC;
    data_in : in  STD_LOGIC_VECTOR (N-1  downto 0);
    -- minus - / plus +
        dataout1: out std_logic_vector (N-2  downto 0);
        -- most significant bit
        dataout2: out std_logic_vector (N-2  downto 0);
```

```
        -- second significant bit
        dataout3: out std_logic_vector (N-2  downto 0);
        -- least significant bit
        dataout4: out std_logic_vector (N-2  downto 0);
                                -- dot
        dataout5: out std_logic_vector (N-2  downto 0);
                                -- decimal
        dataout6: out std_logic_vector (N-2  downto 0));
end COMPONENT translator;



component clock_enable_2ms is
port( clk: in std_logic;
          clk_2ms: out std_logic);
end component;

begin

clk_com:component clock_enable_2ms        --enable delay
        port map ( clk=>clk, clk_2ms=>clk_2ms);



process( clk_2ms,reset)
begin
if(reset='1') then
                state<=power_up;
                cnt20ms<=0;
                flag<=0;
   --      cnt4sec<=0;
else
if(rising_edge(clk_2ms)) then
  r_w<='0';
  rs<='0';
  done<='0';
  dataOUT<=(OTHERS=>'Z');
  enable_translator<='0';
          generate_enable_lcd<='1';
        case(state)is
```

```
        when power_up          => dataOUT<="ZZZZZZZZ";
        if(cnt20ms =10 and enable_init='1') then
                 cnt20ms<=0;
                 state<=setfunction;
                 flag<=0;
            else
                 cnt20ms<=cnt20ms+1;
            end if;
        when setfunction =>rs<='0';
                --mode set 16*2 0011 1000 5*7 dot
                dataOUT(7 downto 5)<="001";
                dataOUT(4)<='1';     --Data length is 8
                dataOUT(3)<='1';        --two line
                dataOUT(2)<='0';         --front5x10
                dataOUT(1 downto 0)<="00";
                if(flag=0) then
                     state<=delay10ms;
                     flag<=1;
                elsif( flag =1) then
                     state<=delay2ms;
                     flag<=2;
                elsif (flag =2) then
                      state<=setfunction;
                      flag<=3;
                elsif (flag = 3) then
                     state<=displayoff;
                     flag<=0;
                else
                     state<=idle;
                end if;
        when delay10ms =>
          if(cnt20ms =5) then
        cnt20ms<=0;
       state<=setfunction;
        else
  cnt20ms<=cnt20ms+1;
dataOUT<="ZZZZZZZZ";
   end if;

        when delay2ms    => state<=setfunction;
```

```
                                        dataOUT<="ZZZZZZZZ";
when displayoff => rs<='0';
                   dataOUT<="00001000";
                   state<=clear;
when clear  =>  rs<='0';
                                dataOUT<="00000001";
                                state<=setmode;
when setmode =>rs<='0';
        -- cursor and shift 0000 0110 no shift
        dataOUT(7 downto 2)<="000001";
        dataOUT(1)<='1';         --cursor I
        dataOUT(0)<='0';         -- no shift
        state<=switchmode;
when switchmode =>rs<='0';
                        --switch and cursor 0000 1100
        dataOUT(7 downto 3)<="00001";
        dataOUT(2)<='1';         --open display
        dataOUT(1)<='0';         --open cursor
        dataOUT(0)<='0';        --blank cursor
        state<=idle;
        done<='1';
  when idle=>
              IF(enable='1') THEN
              state<=first_line;
              END IF;
when first_line=>rs<='0';
        --80H address
        dataOUT(7 downto 0)<="10000000";
        state<=writing_max_1;

when writing_max_1 =>RS<='1';
                R_W<='0';
                   -- Sending M
                dataOUT<=x"4D";
                state<=writing_max_2;

when writing_max_2=>RS<='1';
                R_W<='0';
                  -- Sending A
                dataOUT<=x"41";
```

```
                state<=writing_max_3;
when writing_max_3=>RS<='1';
                R_W<='0';
                 -- Sending x
                dataOUT<=x"58";
                state<=writing_max_4;
when writing_max_4=>RS<='1';
        R_W<='0';
          -- Sending :
        dataOUT<=x"3A";
        state<=writing_max_5;
        enable_translator<='1';
when writing_max_5=>RS<='1';
    -- sign
        R_W<='0';
        enable_translator<='1';
        dataOUT<=dataout1_max;    --dataout
        state<=writing_max_6;
when writing_max_6=>RS<='1';
                 --- x1
                R_W<='0';
                enable_translator<='1';
                dataOUT<=dataout2_max;
                state<=writing_max_7;
when writing_max_7=>RS<='1';
                    --- x2
                R_W<='0';
                enable_translator<='1';
                dataOUT<=dataout3_max;
                state<=writing_max_8;
when writing_max_8=>RS<='1';
                  -- x3
                R_W<='0';
                enable_translator<='1';
                dataOUT<=dataout4_max;
                state<=writing_max_9;
when writing_max_9=>RS<='1';
          --- dot
                R_W<='0';
                enable_translator<='1';
```

```
                    dataOUT<=dataout5_max;
                    state<=writing_max_10;
when writing_max_10=>rs<='1';
                --- decimal
      R_W<='0';
             enable_translator<='1';
             dataOUT<=dataout6_max;
             state<=writing_max_11;
      when writing_max_11=>RS<='1';
             R_W<='0';
              -- Sending a space
             dataOUT<=x"20";
             state<=writing_max_12;
      when writing_max_12=>RS<='1';
             R_W<='0';
             -- Sending C
             dataOUT<=x"43";
             state<=second_line;
      when second_line=>rs<='0';
               done<='1';
            --COH second line address
             dataOUT(7 downto 0)<="11000000";
             state<=writing_min_1;

      when writing_min_1=>RS<='1';
             R_W<='0';
              -- Sending M
             dataOUT<=x"4D";
             state<=writing_min_2;
      when writing_min_2=>RS<='1';
             R_W<='0';
             --Sending I
         dataOUT<=x"49";
         state<=writing_min_3;
      when writing_min_3=>RS<='1';
             R_W<='0';
             -- Sending N
             dataOUT<=x"4E";
              state<=writing_min_4;
      when writing_min_4=>RS<='1';
```

```
        R_W<='0';
                -- Sending :
        dataOUT<=x"3A";
         state<=writing_min_5;
         enable_translator<='1';
when writing_min_5=>RS<='1';
        R_W<='0';
        enable_translator<='1';
        dataOUT<=dataout1_min;
    state<=writing_min_6;
when writing_min_6=>RS<='1';
        R_W<='0';
        enable_translator<='1';
        dataOUT<=dataout2_min;
        state<=writing_min_7;
when writing_min_7=>RS<='1';
        R_W<='0';
        enable_translator<='1';
        dataOUT<=dataout3_min;
         state<=writing_min_8;
when writing_min_8=>RS<='1';
        R_W<='0';
        enable_translator<='1';
        dataOUT<=dataout4_min;
        state<=writing_min_9;
when writing_min_9=>RS<='1';
        R_W<='0';
        enable_translator<='1';
        dataOUT<=dataout5_min;
        state<=writing_min_10;
when writing_min_10=> rs<='1';
    R_W<='0';
        enable_translator<='1';
        dataOUT<=dataout6_min;
        state<=writing_min_11;
when writing_min_11=>RS<='1';
        R_W<='0';
   -- Sending a space
        dataOUT<=x"20";
        state<=writing_min_12;
```

```
        when writing_min_12=>RS<='1';
                R_W<='0';
                -- Sending C
                     dataOUT<=x"43";
                state<=wait_4sec;
          when wait_4sec=>
                generate_enable_lcd<='0';
--      IF(cnt4sec=4000-1)THEN
                        if( swtem ='1') then
                state<=reset_screen;
        --        cnt4sec<=0;
                ELSE
                state<=first_line;
        --        cnt4sec<=cnt4sec+1;
                END IF;


          when reset_screen=>rs<='0';
                dataOUT(7 downto 0)<="00000001";
                state<=  first_line_second_print;

  when first_line_second_print=>
     rs<='0';
         done<='1';
                --80H address
        dataOUT(7 downto 0)<="10000000";
        state<=  writing_now_1;
          when writing_now_1=>RS<='1';
                R_W<='0';
                state<=writing_now_2;
                -- Sending N
              dataOUT<=x"4E";
          when writing_now_2=>
                          RS<='1';
                R_W<='0';
                state<=writing_now_3;
                  -- Sending O
              dataOUT<=x"4F";
          when writing_now_3=>
                          RS<='1';
```

```
                R_W<='0';
                state<=writing_now_4;
            -- Sending W
            dataOUT<=x"57";
  when writing_now_4=>
            RS<='1';
          R_W<='0';
          state<=writing_now_5;
           -- Sending :
        dataOUT<=x"3A";
          enable_translator<='1';
  when writing_now_5=>
            RS<='1';
          R_W<='0';
          enable_translator<='1';
         dataOUT<=dataout1;
          state<=writing_now_6;
  when writing_now_6=>RS<='1';
          R_W<='0';
          enable_translator<='1';
         dataOUT<=dataout2;
          state<=writing_now_7;
  when writing_now_7=>RS<='1';
          R_W<='0';
          enable_translator<='1';
            dataOUT<=dataout3;
          state<=writing_now_8;
  when writing_now_8=>RS<='1';
          R_W<='0';
          enable_translator<='1';
            dataOUT<=dataout4;
          state<=writing_now_9;
  when writing_now_9=>RS<='1';
          R_W<='0';
          enable_translator<='1';
            dataOUT<=dataout5;
          state<=writing_now_10;
when writing_now_10=>RS<='1';
          R_W<='0';
          enable_translator<='1';
```

```
          dataOUT<=dataout6;
        state<=writing_now_11;
  when writing_now_11=>RS<='1';
        R_W<='0';
        -- Sending a space
        dataOUT<=x"20";
        state<=writing_now_12;
  when writing_now_12=> RS<='1';
        R_W<='0';
     -- Sending C
             dataOUT<=x"43";
        state<=print_ind_outd_select;
when print_ind_outd_select=>
     -- select second line
            --COH second line address
     dataOUT(7 downto 0)<="11000000";
     rs<='0';

     IF (ind_outd_select='1') THEN -- indoor
     state<=print_ind_1;
     ELSE -- outdoor
     state<=print_outd_1;
     END IF;
 when print_ind_1=>        RS<='1';
                                R_W<='0';
                                 -- Sending I
                                dataOUT<=x"49";
                                state<=print_ind_2;
 when print_ind_2=>RS<='1';
                         R_W<='0';
                           -- Sending N
                         dataOUT<=x"4E";
                         state<=print_ind_3;
 when print_ind_3=>
                  RS<='1';
        R_W<='0';
          -- Sending D
        dataOUT<=x"44";
        state<=print_ind_4;
 when print_ind_4=>
```

```
                RS<='1';
                 R_W<='0';
         -- sending O
                 dataOUT<=x"4F";
        state<=print_ind_5;
when print_ind_5=>
                RS<='1';
       R_W<='0';
   -- sending O
       dataOUT<=x"4F";
            state<=print_ind_6;
when print_ind_6=>
                RS<='1';
       R_W<='0';
   -- sending R
       dataOUT<=x"52";
            state<=done_print;
when print_outd_1=>
                RS<='1';
       R_W<='0';
       -- Sending O
       dataOUT<=x"4F";
       state<=print_outd_2;
when print_outd_2=>
                RS<='1';
       R_W<='0';
        -- Sending U
       dataOUT<=x"55";
       state<=print_outd_3;
when print_outd_3=>
                RS<='1';
       R_W<='0';
       -- Sending T
       dataOUT<=x"54";
       state<=print_outd_4;
when print_outd_4=>
                RS<='1';
       R_W<='0';
        -- Sending D
       dataOUT<=x"44";
```

```
                state<=print_outd_5;
        when print_outd_5=>
                        RS<='1';
                R_W<='0';
                 -- Sending 0
                dataOUT<=x"4F";
                state<=print_outd_6;
        when print_outd_6=>
                        RS<='1';
                R_W<='0';
                 -- Sending 0
                dataOUT<=x"4F";
                state<=print_outd_7;
        when print_outd_7=>
                        RS<='1';
                R_W<='0';
                -- Sending R
                dataOUT<=x"52";
                state<=done_print;
        when done_print=> done<='1';
                           state<=idle;
        when others          => state<=power_up;

        end case;
end if;
        end if;
end process;


enable_lcd<=not(clk_2ms) when generate_enable_lcd='1' else '0';

-- instantiate translator with same clock of 2 ms
translate_int: translator GENERIC MAP(N=>9)
        PORT MAP (reset=>reset,clk=>clk_2ms,
                start=>enable_translator,data_in=>dataIN,
           dataout1=>dataout1,         -- minus - / plus +
           dataout2=>dataout2,         -- most significant bit
           dataout3=>dataout3,         -- second significant bit
           dataout4=>dataout4,         -- least significant bit
           dataout5=>dataout5,         -- dot
```

```
            dataout6=>dataout6);         -- decimal

maxvalue: translator GENERIC MAP(N=>9)
 PORT MAP (reset=>reset,clk=>clk_2ms,
                start=>enable_translator,
                data_in=>datacompare_max,
            dataout1=>dataout1_max,         -- minus - / plus +
            dataout2=>dataout2_max,         -- most significant bit
            dataout3=>dataout3_max,         -- second significant bit
            dataout4=>dataout4_max,         -- least significant bit
            dataout5=>dataout5_max,         -- dot
            dataout6=>dataout6_max);         -- decimal

minvalue: translator GENERIC MAP(N=>9)
               PORT MAP (reset=>reset,clk=>clk_2ms,
               start=>enable_translator,
               data_in=>datacompare_min,
            dataout1=>dataout1_min,         -- minus - / plus +
            dataout2=>dataout2_min,         -- most significant bit
            dataout3=>dataout3_min,         -- second significant bit
            dataout4=>dataout4_min,         -- least significant bit
            dataout5=>dataout5_min,         -- dot
            dataout6=>dataout6_min);         -- decimal
end beh;
```

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

entity temall is
port        (clk : in std_logic;
                 nrst:in std_logic;
                 in_out: in std_logic;
                 led: out std_logic_vector(8 downto 0);
                 dq:  inout std_logic;            --ja[1]
                 data_recei :in std_logic;        --ja[2]
                 ledtest: out std_logic_vector(10 downto 0);
--                 data_outdoor :in std_logic_vector(8 downto 0);
                 finish:        out std_logic
--                 dataout: out std_logic_vector(6 downto 0)
                 );
end  temall;

architecture beh of temall is

signal finishin:                std_logic;
signal data_indoor:                std_logic_vector(8 downto 0);
signal finishout:        std_logic;
signal clk_1ms:                std_logic;
signal data_outdoor:                std_logic_vector(8 downto 0);
component temperature is
port (        clk: in std_logic;
               nrst:in std_logic;
               dq        :inout std_logic;
               ledtest: out std_logic_vector(10 downto 0);
               led: out std_logic;
               finish:        out std_logic;
               data:out std_logic_vector(8 downto 0));

end component temperature;

component clock_enable_1ms is
        port( clk: in std_logic;
                clk_1ms: out std_logic);
        end component;
```

```
        component decode is

                port(reset                  : in  std_logic;
                        clk                   : in  std_logic;
                        Din                    : in  std_logic;
                        finish                  : out std_logic;
                        Dout                   : out  std_logic_vector(8 downto 0));
        end component;


begin
        clk_com:component clock_enable_1ms
                port map ( clk=>clk, clk_1ms=>clk_1ms);

        decode_com: component decode
                port map(reset=>nrst,clk=>clk_1ms,
                        Din=>data_recei,finish=>finishout,
                         Dout=>data_outdoor);

tem: component temperature
        port map(clk=>clk, nrst=>nrst, dq=>dq,
                ledtest=>ledtest,
                finish=>finishin, data=>data_indoor);

led<= data_indoor when in_out='1'        else
          data_outdoor;

finish<= finishin when in_out='1' else
                 finishout;
end beh;
```

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

entity temperature is
port (        clk: in std_logic;
              nrst:in std_logic;
              dq        :inout std_logic;
              ledtest: out std_logic_vector(10 downto 0);
              led: out std_logic;
              finish: out std_logic;
--              ledtest: out std_logic_vector( 8 downto 0);
--              sw:  in std_logic_vector(1 downto 0);
              data:out std_logic_vector(8 downto 0));
end temperature;

architecture beh of temperature is
component clock_enable_1us is
port( clk: in std_logic;
         clk_1us: out std_logic);
end component;


-----------------signal---------------------------------------
type statetype is (RESET1,RESET2,PRE1,PRE2, CMD_CC,
       CONV_44, READ_BE, GET_TEMP, READ_BIT,WRITE_LOW,
        WRITE_HIGH,CMD_WR,CMD_BE,DELAY,Delay_50ms);
signal clk_1us:         std_logic;
signal cnt:                natural range 0 to 1000 :=0;
signal datatemp:         std_logic;
signal write_low_cnt,
        write_high_cnt:        natural range 0 to 2:=0;
signal read_low_cnt,
        read_high_cnt:        natural range 0 to 9:=0;
signal cnt_wr:        natural range 0 to 8:=0;
signal writetemp:std_logic_vector(7 downto 0);
signal write_flag,cnt_rd:natural range 0 to 3:=0;
signal get_temp_cnt:natural range 0 to 13:=0;
signal dataout:        std_logic_vector(8 downto 0);
SIGNAL state:        statetype;
signal rst_cnt:        natural;
```

```
signal dqtemp:        std_logic;
signal cnt_delay:natural range 0 to 750010:=0;
signal cnt_50ms: natural range 0 to 50010:=0;
signal finished: std_logic;
begin

--------------------component----------------------------------
CLKCOMP: component clock_enable_1us
port map ( clk=>clk, clk_1us=>clk_1us);

-------------------state machine -------------------------------
process( clk_1us,nrst)
begin
        if(nrst='0') then
                state<=RESET1;
                write_high_cnt<=0;
                write_low_cnt<=0;
                rst_cnt<=0;
                read_high_cnt<=0;
                read_low_cnt<=0;
                cnt_wr<=0;
                writetemp<="00000000";
                write_flag<=0;
                get_temp_cnt<=0;
                cnt_rd<=0;
                datatemp<='0';
                finished<='0';
                cnt_50ms<=0;

                cnt<=0;
                led<='1';
                ledtest(10)<='1';
                dqtemp<='1';
                cnt_delay<=0;
        elsif(rising_edge(clk_1us)) then
                case state is
when RESET1          =>        if (cnt> 499) then
        -- controller => ds18s20
                        led<='0';--low reset signal
                        state<=RESET2;
```

```
                else
                        cnt<=cnt+1;
             dq<='0';
             led<='1';
                END IF;
          ledtest(0)<='1';
--                                                  ledtest(10)<='0';
          WHEN RESET2          =>          if (cnt<515) then
                                 --wait 15 us
                        dq<='Z';
                        cnt<=cnt+1;
                        state<=PRE1;
                end if;
                ledtest(1)<='1';
          WHEN PRE1                  =>          if (cnt<560) then
                        -- ds18s20 => controller
                if(dq='0') then
                        led<='1';
                --shows the presence of ds18s20 to controller
                                state<=PRE2;
                        else
                                dq<='Z';
                        end if;
                        cnt<=cnt+1;
                else
                        led<='0';
                        state<=RESET1;
                        cnt<=0;
                end if;
                ledtest(2)<='1';
          when PRE2                  =>          cnt<=cnt+1;
                -- finish the reset
                led<='0';
                if(finished='0')then
                        if (cnt>759) then
                                -- initial finish
                                state<=CMD_CC;
                                -- find the appropriate ds18s20
                                cnt<=0;
                        end if;
```

```
        else
                state<=PRE2;
                dq<='Z';
        end if;
        ledtest(3)<='1';
when CMD_CC         =>
                writetemp<="11001100";
                state<=CMD_WR;
                ledtest(4)<='1';
when CMD_WR                 =>
                        case cnt_wr is
        when 0 to 7         =>
        if(writetemp(cnt_wr)='0') then
                        state<=WRITE_LOW;
                else
                        state<=WRITE_HIGH;
                end if;
                cnt_wr<=cnt_wr+1;
        when 8                  =>  cnt_wr<=0;
                if(write_flag=0) then
                --THE FIRST TIME FOR CC
                        state<= CONV_44;
                        write_flag<=1;
                elsif(write_flag=1)then
                        --44
                        state<= DELAY;
                        write_flag<=2;
                elsif(write_flag=2) then
                --THE SECOND TIME FOR CC
                        state<=CMD_BE;
                        write_flag<=3;
                elsif(write_flag=3) then
                        --BE
                        state<=GET_TEMP;
                        -- get temperature
                        write_flag<=0;
                end if;

        when others =>  cnt_wr<=0;
        end case;
```

```
when DELAY                =>
                if(cnt_delay=749999) then
                        cnt_delay<=0;
                        state<=RESET1;
                        ledtest(5)<='1';
                else
                        cnt_delay<=cnt_delay+1;
                end if;
WHEN WRITE_LOW  =>
        case write_low_cnt is
        when 0                  =>  dq<='0';
                if (cnt=60) then
                        write_low_cnt<=1;
                        cnt<=0;
                else
                        cnt<=cnt+1;
                end if;
        when 1                  =>        dq<='Z';
                if (cnt=10) then
                        write_low_cnt<=2;
                        cnt<=0;
                else
                        cnt<=cnt+1;
                end if;
        when 2                  =>  state<= CMD_WR;
                                        write_low_cnt<=0;
        when others =>  write_low_cnt<=0;
        end case;

when WRITE_HIGH         =>
                case write_high_cnt is
                when 0                  =>        dq<='0';
                        if (cnt=5) then
                                write_high_cnt<=1;
                                cnt<=0;
                        else
                                cnt<=cnt+1;
                        end if;
                when 1                  =>  dq<='Z';
                        if (cnt=60) then
```

```
                                write_high_cnt<=2;
                                cnt<=0;
                        else
                                cnt<=cnt+1;
                        end if;
                when 2                  =>  state<=CMD_WR;
                                            write_high_cnt<=0;
                when others =>  write_high_cnt<=0;
                end case;

when CONV_44        =>          writetemp<="01000100";
          --            dataout<=(others=>'0');
                                state<=CMD_WR;
                                ledtest(6)<='1';
when CMD_BE                 =>  writetemp<="10111110";
                                state<=CMD_WR;
                                ledtest(7)<='1';
when GET_TEMP        =>
         case get_temp_cnt is
        --read temperature from dq
        when 0                          =>          state<=READ_BIT;
                        get_temp_cnt<=get_temp_cnt+1;
        when 1 to 9        =>--0 to 8: 9 bits
                        dataout(get_temp_cnt-1)<=datatemp;
                        state<=READ_BIT;
                        get_temp_cnt<=get_temp_cnt+1;
        when 10                         =>      dataout(8)<=dataout(8);
                        get_temp_cnt<=0;
                        state<=Delay_50ms ;
                        ledtest(8)<='1';
                        cnt<=0;
        WHEN OTHERS                 =>      get_temp_cnt<=0;
        end case;

when Delay_50ms  =>
         if(cnt_50ms=49999) then
           cnt_50ms<=0;
           finished<='0';
           state<=RESET1;
           ledtest(9)<='1';
```

```
        else
              cnt_50ms<=cnt_50ms+1;
              finished<='1';
        end if;

when READ_BIT        =>
case cnt_rd is
when 0                =>        dq<='0';
                if (cnt=3) then
                        cnt_rd<=1;
                        cnt<=0;
                else
                        cnt<=cnt+1;
                end if;
when 1                =>  dq<='Z';
                if (cnt=3) then
                        cnt_rd<=2;
                        cnt<=0;
                else
                        cnt<=cnt+1;
                end if;
when 2                =>        dq<='Z';
                datatemp<=dq;
                if(cnt=5) then
                --13us read out data from dq;
                        cnt_rd<=3;
                        cnt<=0;
                else
                        cnt<=cnt+1;
                end if;
when 3                =>  dq<='Z';
                if(cnt=60) then
                        --delay 60 us
                        cnt<=0;
                        cnt_rd<=0;
                        state<=GET_TEMP;
                else
                        cnt<=cnt+1;
                end if;
when others        =>         cnt_rd<=0;
```

```
        end case;
        when others     =>          state<=RESET1;
        END CASE;
        END IF;
end process;
finish<=finished;
data<=dataout;
end beh;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.math_real."log2";
use IEEE.math_real."ceil";

entity top_humidity_sensors_interface is
    Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           enable : in  STD_LOGIC;
           pwm_out : out  STD_LOGIC;
           sclk : inout  STD_LOGIC;
           sdata : inout  STD_LOGIC);
end top_humidity_sensors_interface;

architecture Behavioral of top_humidity_sensors_interface is

component counter_wd is
generic ( N : integer := 8;
                        MAX_VAL: integer :=200);
    Port ( clk : in  STD_LOGIC;
                                enable: in std_logic;
           reset : in  STD_LOGIC;
           tc : out  STD_LOGIC);
end component counter_wd;


component humidity_interface IS
PORT ( clk,reset,enable: IN std_logic;
                data_out: out std_logic_vector(13 DOWNTO 0);
                        ld_data,enable_cnt: out std_logic;
                        sclk,sdata: inout std_logic);
end component humidity_interface;

COMPONENT  pwm_generator IS
GENERIC (N:integer:=14;
           pwm_freq        : INTEGER := 1000     --PWM switching frequency in Hz
                        );
PORT ( clk,reset,enable,ld_val_duty: IN std_logic;
                data_in: IN std_logic_vector(N-1 DOWNTO 0);  -- humidity percentage
                pwm_out: OUT std_logic
```

```
                );
END COMPONENT  pwm_generator;

SIGNAL data_out_to_pwm: std_logic_vector(13 DOWNTO 0);
SIGNAL enable_read,ld_val,enable_from_i2c,enable_count: std_logic;

begin

enable_count<=enable and enable_from_i2c;
-- every 100 ms  read a new humidity value
counter: counter_wd
         GENERIC MAP (N=> 1+integer(ceil(log2(real(10000)))),
                                        MAX_VAL=>10000)
                                        PORT MAP(clk=>clk,reset=>reset,
                                        enable=>enable_count,tc=>enable_read);
i2c_interface: humidity_interface
        PORT MAP (clk=>clk,reset=>reset,
            enable=>enable,
                        enable_cnt=> enable_from_i2c,
                        ld_data=>ld_val,
                        data_out=> data_out_to_pwm,
                        sclk=>sclk, sdata=>sdata);
pwm: pwm_generator GENERIC MAP(N=>14,pwm_freq=>1000)
                PORT MAP (enable=>enable_count,clk=>clk,
                        reset=>reset,ld_val_duty=>ld_val,
                        data_in=>data_out_to_pwm,pwm_out=>pwm_out);


end Behavioral;
```

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


entity comparison is
GENERIC (N:integer:= 8);
    Port ( clk,reset: in STD_LOGIC;
                                in_out_sel: in std_logic;
                                data_in : in  STD_LOGIC_VECTOR (N-1 downto 0);
           data_outmax : out  STD_LOGIC_VECTOR (N-1 downto 0);
                   data_outmin : out  STD_LOGIC_VECTOR (N-1 downto 0);
           start_comparison : in  STD_LOGIC; -- aka enable
           done_comparison : out  STD_LOGIC);
end comparison;

architecture structural of comparison is
signal data_max_in: STD_LOGIC_VECTOR (N-1 downto 0);
signal data_min_in: STD_LOGIC_VECTOR (N-1 downto 0);
signal data_max_out: STD_LOGIC_VECTOR (N-1 downto 0);
signal data_min_out: STD_LOGIC_VECTOR (N-1 downto 0);
signal start_comparison_in: STD_LOGIC;
signal start_comparison_out: STD_LOGIC;
signal done_comparison_in: STD_LOGIC;
signal done_comparison_out: STD_LOGIC;

component comparison_com is
GENERIC (N:integer:= 8);
    Port ( clk,reset: in STD_LOGIC;
--                                  in_out_sel: in std_logic;
                                data_in : in  STD_LOGIC_VECTOR (N-1 downto 0);
           data_outmax : out  STD_LOGIC_VECTOR (N-1 downto 0);
                   data_outmin : out  STD_LOGIC_VECTOR (N-1 downto 0);
           start_comparison : in  STD_LOGIC; -- aka enable
           done_comparison : out  STD_LOGIC);
end component comparison_com;

begin
process (clk)
begin
```

```
        if(rising_edge (clk)) then
           if(in_out_sel='1') then
              start_comparison_in<=start_comparison;
            else
               start_comparison_out<=start_comparison;
            end if;

        end if;
end process;
comparison_indoor:comparison_com GENERIC MAP (9) PORT MAP(clk=>clk,reset=>reset,
                        data_in=>data_in ,
                        data_outmax=>data_max_in ,
                    data_outmin=>data_min_in,
            start_comparison=>start_comparison_in,
            done_comparison=>done_comparison_in
                   );
comparison_outdoor:comparison_com GENERIC MAP (9) PORT MAP(clk=>clk,reset=>reset,
                        data_in=>data_in ,
                        data_outmax=>data_max_out ,
                    data_outmin=>data_min_out,
            start_comparison=>start_comparison_out ,
            done_comparison=>done_comparison_out
                   );
done_comparison<= done_comparison_in when in_out_sel='1' else
                                        done_comparison_out;
data_outmax<=data_max_in when in_out_sel='1' else
                         data_max_out;
data_outmin<=data_min_in when in_out_sel='1' else
                        data_min_out;
end structural;
```

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



entity counter_wd is
generic ( N : integer := 8;
                        MAX_VAL: integer :=200);
    Port ( clk : in  STD_LOGIC;
                                enable: in std_logic;
           reset : in  STD_LOGIC;
           tc : out  STD_LOGIC);
end counter_wd;

architecture Behavioral of counter_wd is

SIGNAL cnt: std_logic_vector(N-1 DOWNTO 0);
begin


PROCESS(clk,reset)
BEGIN
IF(reset='1') THEN

cnt<=(OTHERS=>'0');
tc<='0';
ELSE
IF(clk='1' and clk'event) THEN
IF (enable='1') THEN
        IF( to_integer(unsigned(cnt))>=MAX_VAL-1) THEN
        tc<='1';
        cnt<=(OTHERS=>'0');
        ELSE
        cnt<=std_logic_vector(unsigned(cnt)+1);
        tc<='0';
        END IF;
ELSE

cnt<=(OTHERS=>'0');
```

```
END IF;

END IF;


END IF;
END PROCESS;


end Behavioral;
```

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

entity decode is
        port(reset : in std_logic;
                clk : in std_logic;
                Din : in std_logic;
                finish : out std_logic;
                Dout : out std_logic_vector(8 downto 0));
end entity;



architecture arc of decode is

--000011110=>010101011010101001
        signal init_seq : std_logic_vector(17 downto 0);
        signal start_flag : std_logic;
        signal data_in : std_logic;
        signal phase : natural range 0 to 2 := 0;
        signal cnt : natural range 0 to 9 := 9;
        --        signal finish          : std_logic;
        --detect the start sequence,
        --decode the data

begin

        process(clk)
        begin
                if(reset='0')then
                        cnt <= 0;
                        init_seq <= (others => '1');
                        finish <= '0';
                        --        Dout<=(others=>'1');
                else
                if(falling_edge(clk)) then
                        if(init_seq/="010101011010101001")then
                        init_seq(17 downto 1) <= init_seq(16 downto 0);
                        init_seq(0) <= Din;
                        cnt <= 9;
```

```
                        phase <= 0;
                        else
                        if(cnt > 0)then
                        finish <= '0';
                                case phase is
                                when 0 => if (Din='0') then
                                        data_in <= '0';
                                else
                                        data_in <= '1';
                                        end if;
                                        phase <= 1;
                                when 1 => if (data_in='0' and Din='1') then
                                        Dout(cnt-1) <= '0';
                                elsif(data_in='1' and Din='0') then
                                        Dout(cnt-1) <= '1';
                                else
                                        Dout(cnt-1) <= '1';
                                        end if;
                                        phase <= 0;
                                        cnt   <= cnt-1;
                                when others => Dout(cnt-1) <= '1';
                                        phase <= 0;
                                end case;
                        else
                                finish   <= '1';
                                init_seq <= (others => '0');
                        end if;
                        end if;
                end if;
                end if;
        end process;

end architecture arc;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use IEEE.math_real."log2";
use IEEE.math_real."ceil";

entity  pwm_generator IS
GENERIC (N:integer:=14;
   --PWM switching frequency in Hz
           pwm_freq        : INTEGER := 1000
                       );
PORT ( clk,reset,enable,ld_val_duty: IN std_logic;
   -- raw data that must be processed
           data_in: IN std_logic_vector(N-1 DOWNTO 0);
                pwm_out: OUT std_logic
                );
END entity  pwm_generator;

architecture beh of pwm_generator is

CONSTANT max_val: integer:= 100000000/pwm_freq;
CONSTANT num_bits: integer:=integer(ceil(log2(real((max_val-1)))));

signal counter: std_logic_vector( num_bits-1  downto 0);
signal duty: std_logic_vector(num_bits-1 downto 0);
  begin

PROCESS(clk, reset)
  BEGIN
    IF(reset = '1') THEN
           counter<=(OTHERS=>'0');
           pwm_out<='0';
    ELSIF(clk'EVENT AND clk = '1') THEN
              IF(enable='1')THEN
                                  counter<=std_logic_vector(unsigned(counter)+1);
                                  IF(unsigned(counter)<unsigned(duty))THEN
                                 pwm_out<='0';
                                 ELSE
                                 pwm_out<='1';
                                  END IF;
```

```
            ELSE
            pwm_out<='0';
            END IF;
    END IF;
  END PROCESS;


-- register for the duty cycle
PROCESS(clk,reset)
BEGIN
IF(reset='1') THEN
 -- 50 % of default duty cycle
duty<=std_logic_vector(to_unsigned((max_val)/2,num_bits)-1);
ELSIF( rising_edge(clk)) THEN
        IF(ld_val_duty='1' and enable='1') THEN
        duty<=std_logic_vector(to_unsigned(
            to_integer(unsigned(data_in))*max_val*100/(2**14)-1,num_bits )) ;
         END IF;

END IF;
END PROCESS;


end beh;
```

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


entity translator is
GENERIC ( n: integer:= 9);
    Port ( reset : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           start : in  STD_LOGIC;
           data_in : in  STD_LOGIC_VECTOR (N-1  downto 0);
                  -- minus - / plus +
                  dataout1: out std_logic_vector (N-2  downto 0);
                  -- most significant bit
                  dataout2: out std_logic_vector (N-2  downto 0);
                         -- second significant bit
                  dataout3: out std_logic_vector (N-2  downto 0);
                         -- least significant bit
                  dataout4: out std_logic_vector (N-2  downto 0);
                       -- dot
                  dataout5: out std_logic_vector (N-2  downto 0);
                  -- decimal
                  dataout6: out std_logic_vector (N-2  downto 0));
end translator;

architecture Behavioral of translator is



--signal dot: std_logic_vector(7 downto 0):=x"2E";
--signal x1,x2,x3,decimal_value,sign_value: std_logic_vector(7 DOWNTO 0);

signal datatem: natural range 0 to 128:=0;
signal datax1, datax2, datax3: natural range 0 to 10:=0;
begin


translation:process (data_in)
begin
IF(data_in(N-1)='1')THEN
```

```
-- ascii of minus
dataout1<=x"2D";
--N=9                  7-1 7bits
datatem<=(to_integer(unsigned(not(data_in(N-2 DOWNTO 1)))))+1;

ELSE
-- ascii of plus
dataout1<=x"2B";                                          -- +
datatem<=to_integer(unsigned(data_in(N-2 DOWNTO 1)));

END IF;

-- x1x2x3

datax1<=datatem / 100;
datax2<=(datatem mod 100) / 10;
datax3<=(datatem mod 100) mod 10;
dataout4(7 downto 4)<= "0011";

-- less significan value
dataout4(3 downto 0)<=std_logic_vector(to_unsigned (datax3,4));


dataout3(7 downto 4)<= "0011";
-- most significan value -1
dataout3(3 downto 0)<=std_logic_vector(to_unsigned (datax2,4));


dataout2(7 downto 4)<= "0011";
-- most significan value
dataout2(3 downto 0)<=std_logic_vector(to_unsigned (datax1,4));

dataout5<="00101110";

-- decimal part
IF(data_in(0)='1')THEN
-- .5
dataout6<="00110101";--x"35";                                  --5
ELSE
--  .0
```

```
dataout6<="00110000";--x"30";                                    --0
END IF;

end process translation;




end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;
use IEEE.math_real."log2";
use IEEE.math_real."ceil";




entity humidity_interface is
        PORT ( clk,reset,enable : IN std_logic;
                data_out           : out std_logic_vector(13 DOWNTO 0);
                ld_data,enable_cnt : out   std_logic;
                sclk,sdata         : inout std_logic);
end humidity_interface;

architecture Behavioral of humidity_interface is

        TYPE state_t IS (power_up , idle,
                        measure_request_1,measure_request_2,
                        measure_request_3,measure_request_4,
                        measure_request_5,measure_request_6,
                        measure_request_7,measure_request_8,
                        measure_request_9,measure_request_10,
                        measure_request_11,measure_request_12,
                        measure_request_13,measure_request_14,
                        data_fetch_1 ,data_fetch_2 ,data_fetch_3,
                        data_fetch_4 ,data_fetch_5 ,data_fetch_6 ,
                        data_fetch_7 ,data_fetch_8 ,data_fetch_9 ,
                        data_fetch_10 ,data_fetch_11 ,data_fetch_12,
                        data_fetch_13 ,data_fetch_14 ,data_fetch_15 ,
                        data_fetch_16 ,data_fetch_17 ,data_fetch_18 ,
                        data_fetch_19 ,data_fetch_20 ,data_fetch_21 ,
                        data_fetch_22 ,data_fetch_23 ,data_fetch_24 ,
                        data_fetch_25 ,data_fetch_26 ,data_fetch_27 ,
                        data_fetch_28 ,        data_fetch_29 ,data_fetch_30 ,
                        data_fetch_31 ,data_fetch_32 ,
                        wait_measure,update_duty_cycle_pwm,hang,gen);
        CONSTANT freq_sclk : integer := 1000; --- 100 kHz
```

```
        SIGNAL curr_state,next_state : state_t;
        SIGNAL counter_cl :
                std_logic_vector( 1+integer(ceil(log2(real(100000)))) DOWNTO 0);
        SIGNAL counter : std_logic_vector( 8 DOWNTO 0);
        SIGNAL counter_wait:
                 std_logic_vector( 1+integer(ceil(log2(real(6000000)))) DOWNTO 0);

        SIGNAL check_ack,check_collision,sclk_read,
                                read_sclk,generate_data,
                                reset_cnt_clk,sclk_rd,tc_wait,
                                tc_cl,enable_wait,shf_en,clk_rd,clk_wr,
                                clk_wr_neg,sclk_collision,reset_counter_cl,
                                generate_clk,sdata_i,ack : std_logic;
        SIGNAL sclk_i : STD_LOGIC := '0';
        -- first two bits are the status registers
        SIGNAL data : std_logic_vector(15 DOWNTO 0);
begin


        sclk_i_gen : PROCESS(clk,reset)
        BEGIN
                IF (reset='1')THEN
                        counter <= (OTHERS => '0');
                ELSIF(rising_edge(clk))THEN

                        IF(to_integer(unsigned(counter))= freq_sclk/2-1) then
                                sclk_i  <= not(sclk_i);
                                counter <= (OTHERS => '0');
                        ELSE
                                counter <= std_logic_vector(unsigned(counter)+1);
                        END IF;
                END IF;
        END PROCESS sclk_i_gen;


        state_reg : PROCESS(reset,sclk_i)
        BEGIN
IF(reset='1' ) THEN
        data         <= (OTHERS => '0');
        counter_wait <= (OTHERS => '0');
```

```
        curr_state   <= power_up;
        counter_cl   <= (OTHERS => '0');
ELSIF (rising_edge(sclk_i))THEN
        IF( reset_counter_cl='1' ) THEN
                counter_cl <= (OTHERS => '0');
        ELSE
                IF( to_integer(unsigned(counter_cl))=100000-1 ) THEN
                        tc_cl      <= '1';
                        counter_cl <= (OTHERS => '0');
                ELSE
                        tc_cl      <= '0';
                counter_cl <= std_logic_vector(unsigned(counter_cl)+1);
                END IF;
        END IF;
        curr_state <= next_state;
        IF(enable_wait='1') THEN -- 60 s
                IF(to_integer(unsigned(counter_wait))= 6000000-1) THEN
                        counter_wait <= (OTHERS => '0');
                        tc_wait      <= '1';
                ELSE
                        tc_wait      <= '0';
                counter_wait <= std_logic_vector(unsigned(counter_wait)+1);
                END IF;
        ELSE
                counter_wait <= (OTHERS => '0');
        END IF;
        -- shift register for the data
        IF (shf_en='1') THEN
                data(15 DOWNTO 1) <= data(14 downto 0);
                data(0)           <= sdata;
        END IF;
        IF(generate_clk='1' and check_ack='1' and generate_data='0' ) THEN
                ack <= sdata;
                else
                ack <= '1';
        END IF;
                END IF;

        END PROCESS state_reg;
```

```
        cl : PROCESS(curr_state,tc_cl,ack,enable,data,tc_wait)
        BEGIN
        -- default values
        data_out         <= (OTHERS => '0');
        ld_data          <= '0';
        reset_counter_cl <= '0';
        generate_clk     <= '0';
        enable_cnt       <= '0';
        enable_wait      <= '0';
        shf_en           <= '0';
        check_ack        <= '0';
        sdata_i          <= '1';
        generate_data    <= '0';
CASE curr_state IS
        WHEN power_up => IF ( tc_cl='1' ) THEN -- 1 s
                next_state <= idle;
        ELSE
                next_state <= curr_state;
                END IF;
        WHEN idle => reset_counter_cl <= '1';
                enable_cnt <= '1';
                IF(enable='1') THEN
                        next_state <= measure_request_1;
                ELSE
                        next_state <= curr_state;
                END IF;
        WHEN measure_request_1 => --  start bit
                generate_data <= '1';
                sdata_i    <= '0';
                next_state <= measure_request_3;
        WHEN measure_request_3 =>
                -- write address
                sdata_i       <= '0'; -- first address bit msb
                generate_clk  <= '1';
                next_state    <= measure_request_4;
                generate_data <= '1';
        WHEN measure_request_4 => sdata_i <= '1';
                generate_clk  <= '1';
                next_state    <= measure_request_5;
                generate_data <= '1';
```

```
WHEN measure_request_5 => sdata_i <= '0';
        generate_clk  <= '1';
        generate_data <= '1';
        next_state <= measure_request_6;
WHEN measure_request_6 => sdata_i <= '1';
        generate_clk <= '1';
        generate_data <= '1';
        next_state <= measure_request_7;
WHEN measure_request_7 => sdata_i <= '0';
        generate_clk  <= '1';
        generate_data <= '1';
        next_state <= measure_request_8;
WHEN measure_request_8 => sdata_i <= '0';
        generate_data <= '1';
        generate_clk <= '1';
        next_state <= measure_request_9;
WHEN measure_request_9 => sdata_i <= '0';
        generate_clk  <= '1';
        generate_data <= '1';
        next_state <= measure_request_10;
WHEN measure_request_10 => -- write
        sdata_i       <= '0';
        generate_clk  <= '1';
        generate_data <= '1';
        next_state    <= measure_request_11;
WHEN measure_request_11 => -- ack
        generate_clk  <= '1';
        generate_data <= '0';
        check_ack     <= '1';
        IF ( ack='1' ) THEN
                next_state <= measure_request_11;
        ELSE
                next_state   <= measure_request_12;
                generate_clk <= '0';
        END IF;
WHEN measure_request_12 => -- stop bit
        sdata_i       <= '1';
        generate_data <= '1';
        generate_clk  <= '0';
        next_state    <= wait_measure;
```

```
WHEN wait_measure => -- wait for 90 sec
        enable_wait <= '1';
        enable_cnt  <= '0';
        sdata_i     <= '1';
        --data_out<="00"&x"00F";
        generate_clk     <= '0';
        generate_data    <= '0';
        reset_counter_cl <= '1';
        IF ( tc_wait='1') THEN
                next_state <= data_fetch_1;
        ELSE
                next_state <= curr_state;
        END IF;
WHEN data_fetch_1 => --  start bit
        sdata_i      <= '0';
        generate_clk <= '0';
        next_state <= data_fetch_3;
WHEN data_fetch_3 => -- write address
-- first address bit msb
        sdata_i       <= '0';
        generate_clk  <= '1';
        generate_data <= '1';
        next_state <= data_fetch_4;
WHEN data_fetch_4 => sdata_i <= '1';
        generate_clk  <= '1';
        generate_data <= '1';
        next_state <= data_fetch_5;
WHEN data_fetch_5 =>
        sdata_i       <= '0';
        generate_clk  <= '1';
        generate_data <= '1';
        next_state <= data_fetch_6;
WHEN data_fetch_6 =>
        sdata_i       <= '1';
        generate_clk  <= '1';
        generate_data <= '1';
        next_state <= data_fetch_7;
WHEN data_fetch_7 =>
        sdata_i       <= '0';
        generate_clk  <= '1';
```

```
        generate_data <= '1';
        next_state <= data_fetch_8;
WHEN data_fetch_8 => sdata_i <= '0';
        generate_clk  <= '1';
        generate_data <= '1';
        next_state <= data_fetch_9;
WHEN data_fetch_9 => sdata_i <= '0';
        generate_clk  <= '1';
        next_state    <= data_fetch_10;
        generate_data <= '1';
WHEN data_fetch_10 => sdata_i <= '0';
        generate_clk  <= '1';
        generate_data <= '1';
        next_state <= data_fetch_11;
WHEN data_fetch_11 => -- read
        sdata_i       <= '1';
        generate_clk  <= '1';
        next_state    <= data_fetch_12;
        generate_data <= '1';
WHEN data_fetch_12 => -- ack
        generate_clk <= '1';
        check_ack    <= '1';
        IF ( ack='1' ) THEN
                next_state <= data_fetch_12;
        ELSE
                next_state <= data_fetch_14;
        END IF;
WHEN data_fetch_14 => -- msb status bit
        generate_clk <= '1';
        shf_en       <= '1';
        next_state <= data_fetch_15;
WHEN data_fetch_15 => -- lsb  status bit
        generate_clk <= '1';
        shf_en       <= '1';
        next_state <= data_fetch_16;
WHEN data_fetch_16 => -- humidity 13
        generate_clk <= '1';
        shf_en       <= '1';
        next_state <= data_fetch_17;
WHEN data_fetch_17 => -- humidity 12
```

```
        generate_clk <= '1';
        shf_en       <= '1';
        next_state <= data_fetch_18;
WHEN data_fetch_18 => --  humidity 11
        generate_clk <= '1';
        shf_en       <= '1';
        next_state <= data_fetch_19;
WHEN data_fetch_19 => -- humidity 10
        generate_clk <= '1';
        shf_en       <= '1';
        next_state <= data_fetch_20;
WHEN data_fetch_20 => -- humidity 9
        generate_clk <= '1';
        shf_en       <= '1';
        next_state <= data_fetch_21;
WHEN data_fetch_21 => -- humidity 8
        generate_clk <= '1';
        shf_en       <= '1';
        next_state <= data_fetch_22;
WHEN data_fetch_22 => -- master ack
        sdata_i       <= '1';
        generate_clk  <= '1';
        generate_data <= '1';
        next_state <= data_fetch_23;
WHEN data_fetch_23 => -- humidity 7
        generate_clk <= '1';
        shf_en       <= '1';
        next_state <= data_fetch_24;
WHEN data_fetch_24 => -- humidity 6
        generate_clk <= '1';
        shf_en       <= '1';
        next_state <= data_fetch_25;
WHEN data_fetch_25 => -- humidity 5
        generate_clk <= '1';
        shf_en       <= '1';
        next_state <= data_fetch_26;
WHEN data_fetch_26 => -- humidity 4
        generate_clk <= '1';
        shf_en       <= '1';
        next_state <= data_fetch_27;
```

```
        WHEN data_fetch_27 => --  humidity 3
                generate_clk <= '1';
                shf_en       <= '1';
                next_state <= data_fetch_28;
        WHEN data_fetch_28 => -- humidity 2
                generate_clk <= '1';
                shf_en       <= '1';
                next_state <= data_fetch_29;
        WHEN data_fetch_29 => -- humidity 1
                generate_clk <= '1';
                shf_en       <= '1';
                next_state <= data_fetch_30;
        WHEN data_fetch_30 => -- humidity 0
                generate_clk <= '1';
                shf_en       <= '1';
                next_state <= data_fetch_31;
        WHEN data_fetch_31 =>
-- master n ack
-- ( stop communication otherwise slave will send the temp )
                sdata_i       <= '0';
                generate_clk  <= '0';
                generate_data <= '1';
                next_state <= data_fetch_32;
        WHEN data_fetch_32 => -- stop bit
                sdata_i       <= '1';
                generate_clk  <= '0';
                generate_data <= '1';
                next_state <= update_duty_cycle_pwm;
        WHEN update_duty_cycle_pwm => -- for 1 cc
                -- data 15  at 1 it is in command mode
                -- data 14 at 1  no new value
                IF(data(15)='0' AND data(14)='0') THEN
                        -- new value and no command mode
                        ld_data    <= '1';
                        -- raw data for the pwm generator
                        data_OUT   <= data(13 downto 0);
                        next_state <= idle;
                ELSE
                -- default value 50 %
                        data_OUT <= "01"&x"FFF";
```

```
          --data_OUT<=data(15)& data(14)&x"FF0";
                        next_state <= idle;
                END IF;
        WHEN hang => next_state <= curr_state;
                -- data_OUT<="11"&x"FFF"; -- debug
                data_OUT         <= (OTHERS => '0');
                sdata_i          <= '0';
                ld_data          <= '0';
                reset_counter_cl <= '0';
                generate_clk     <= '0';
                enable_cnt       <= '0';
        WHEN OTHERS => next_state <= power_up;
                data_out         <= (OTHERS => '0');
                sdata_i          <= '1';
                ld_data          <= '0';
                reset_counter_cl <= '0';
                generate_clk     <= '0'; generate_data <= '0';
                enable_cnt       <= '0';
END CASE;

        END PROCESS cl;


        -- tristate output

        sclk <= '0' when (generate_clk='1' and sclk_i='1')
                        ELSE 'Z';


        sdata <= '0'
        when (sdata_i='0' and generate_data='1' and check_ack='0' )
                 ELSE 'Z';



end Behavioral;
```

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

entity clock_enable_1ms is
port( clk: in std_logic;
          clk_1ms: out std_logic);
end clock_enable_1ms;

architecture beh of clock_enable_1ms is
signal cnt: natural range 0 to 99999 :=0;
begin
clk1us: process(clk)
begin
        IF(RISING_EDGE(CLK))THEN
                if(cnt= 99999) then
                        clk_1ms<='1';
                        cnt<=0;
                else
                        clk_1ms<='0';
                        cnt<=cnt+1;
                END IF;
        END IF;
end process;
end beh;
```

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

entity clock_enable_10s is
port( clk: in std_logic;
          clk_10s: out std_logic);
end clock_enable_10s;

architecture beh of clock_enable_10s is
signal cnt: natural range 0 to 1000000000 :=0;

begin
clk1us: process(clk)
begin
        IF(RISING_EDGE(CLK))THEN
                if(cnt= 999999999) then
                        clk_10s<= '1';
                        cnt<=0;
                else
                        clk_10s<='0';
                        cnt<=cnt+1;
                END IF;
        END IF;

end process;
end beh;
```

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

entity clock_enable_1us is
port( clk: in std_logic;
          clk_1us: out std_logic);
end clock_enable_1us;

architecture beh of clock_enable_1us is
signal cnt: natural range 0 to 99 :=0;
begin
clk1us: process(clk)
begin
        IF(RISING_EDGE(CLK))THEN
                if(cnt= 99) then
                        clk_1us<='1';
                        cnt<=0;
                else
                        clk_1us<='0';
                        cnt<=cnt+1;
                END IF;
        END IF;
end process;
end beh;
```

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

entity clock_enable_2ms is
port( clk: in std_logic;
          clk_2ms: out std_logic);
end clock_enable_2ms;

architecture beh of clock_enable_2ms is
signal cnt: natural range 0 to 50000 :=0;
signal clk_e: std_logic:='0';
begin
clk1us: process(clk)
begin
        IF(RISING_EDGE(CLK))THEN
                if(cnt= 49999) then
                        clk_e<= not clk_e;
                        cnt<=0;
                else
--                         clk_1us<='0';
                        cnt<=cnt+1;
                END IF;
        END IF;
clk_2ms<=clk_e;
end process;
end beh;
```

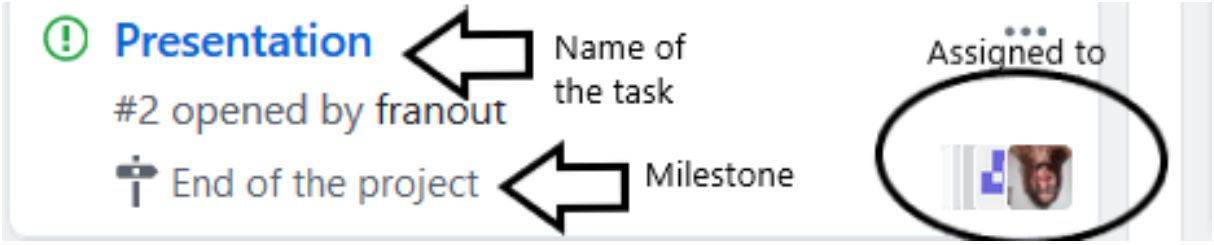
14 Appendix - Utilization table of FPGA resources, power consumption and static timing analysis

Resource	Utilization	Available	Utilization %
LUT	702	63400	1.11
FF	564	126800	0.44
DSP	2	240	0.83
IO	34	210	16.19
BUFG	5	32	15.63

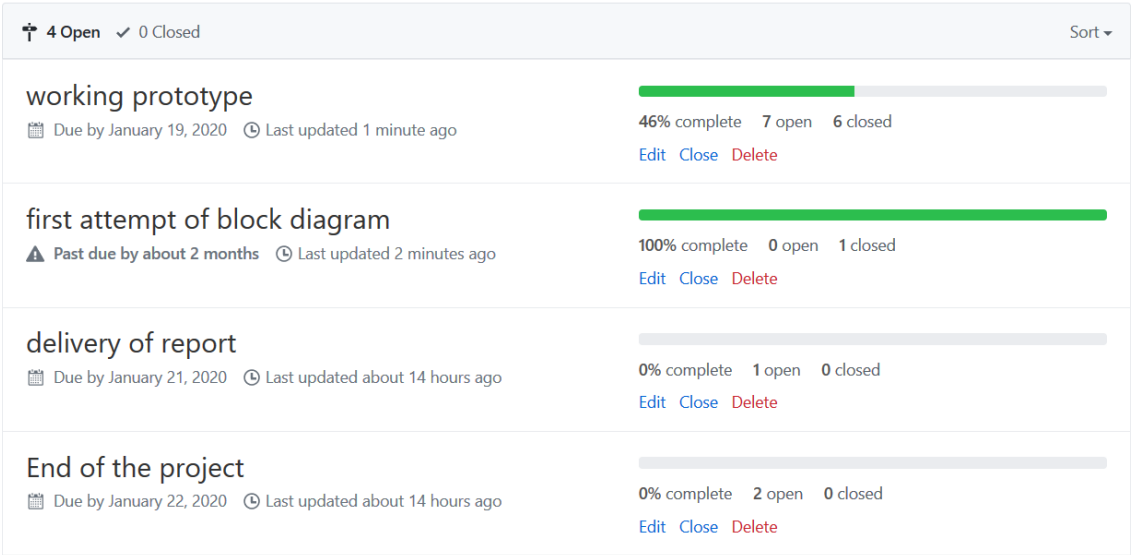
15 Appendix - Project Management

The project management has been handled on GitHub through the feature *Project*.
In the following, various step of the project will be presented as screenshots.
At the beginning of the project all the task were into the *TODO* list. As soon as the design started, the task moved from *TODO* to *In Progress* section.

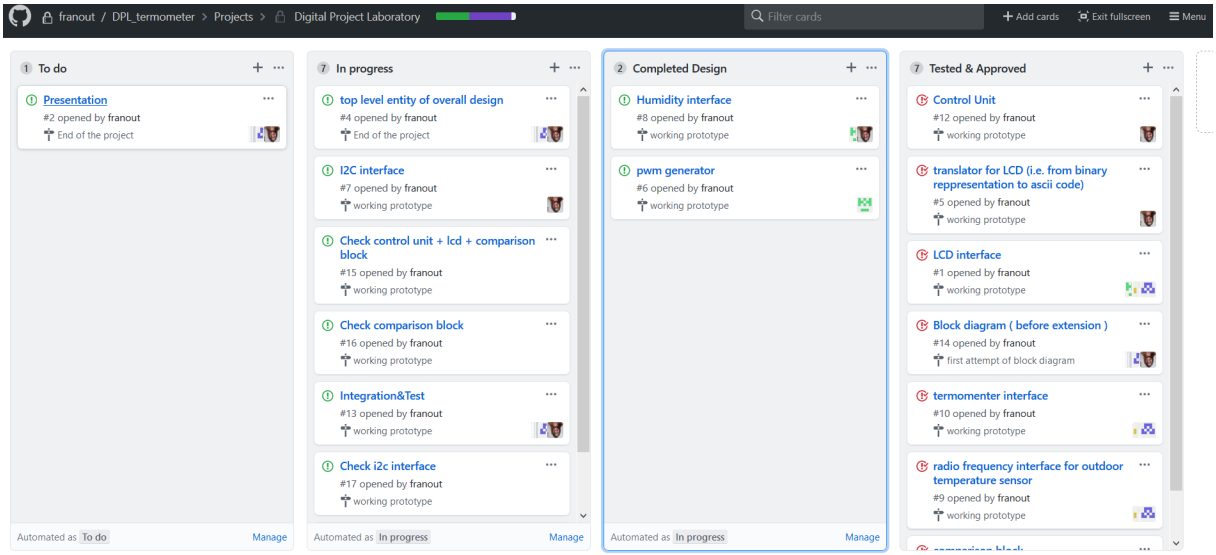
Legend



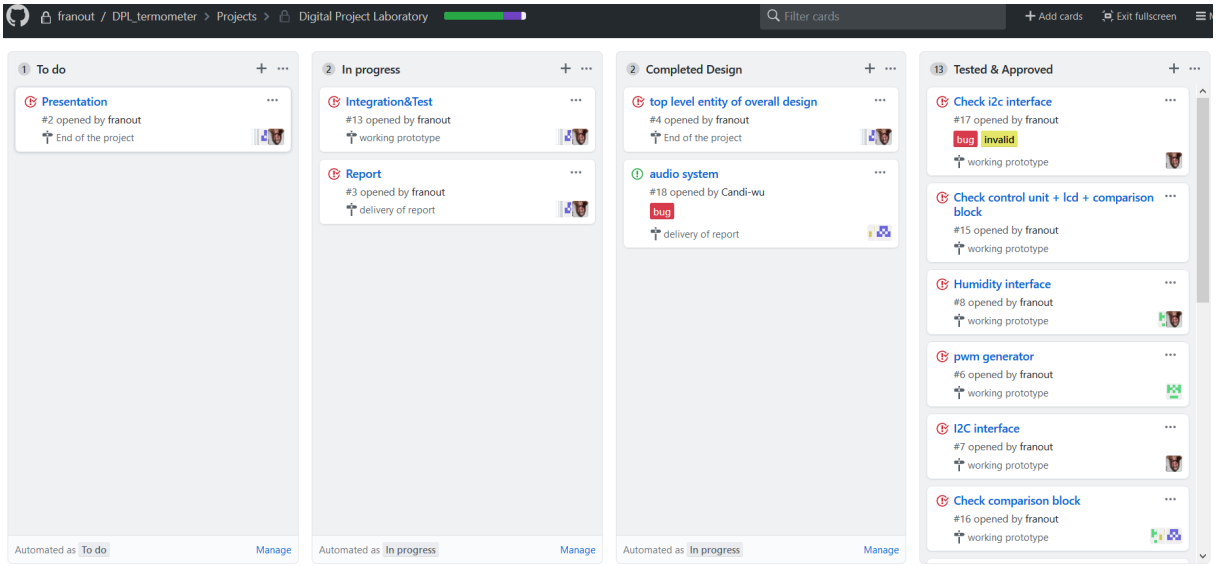
Milestones



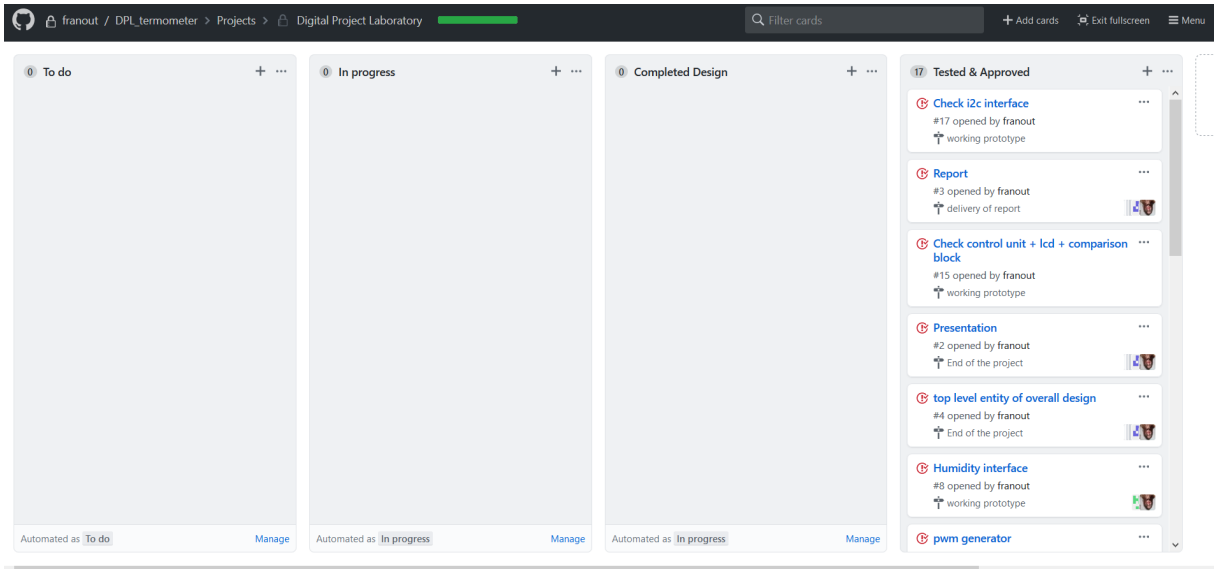
Beginning of January



Middle of January



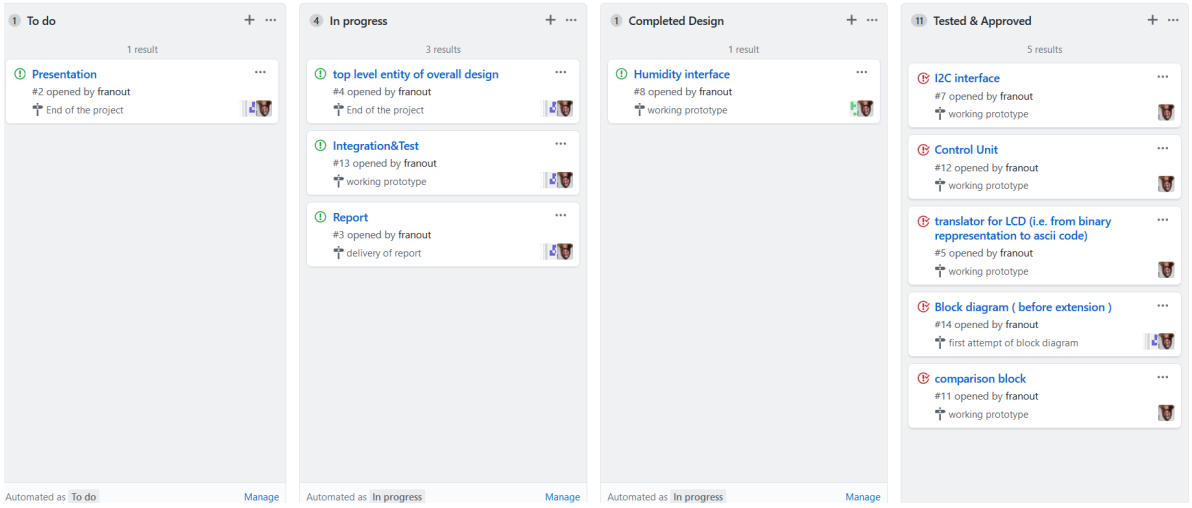
End of the Road



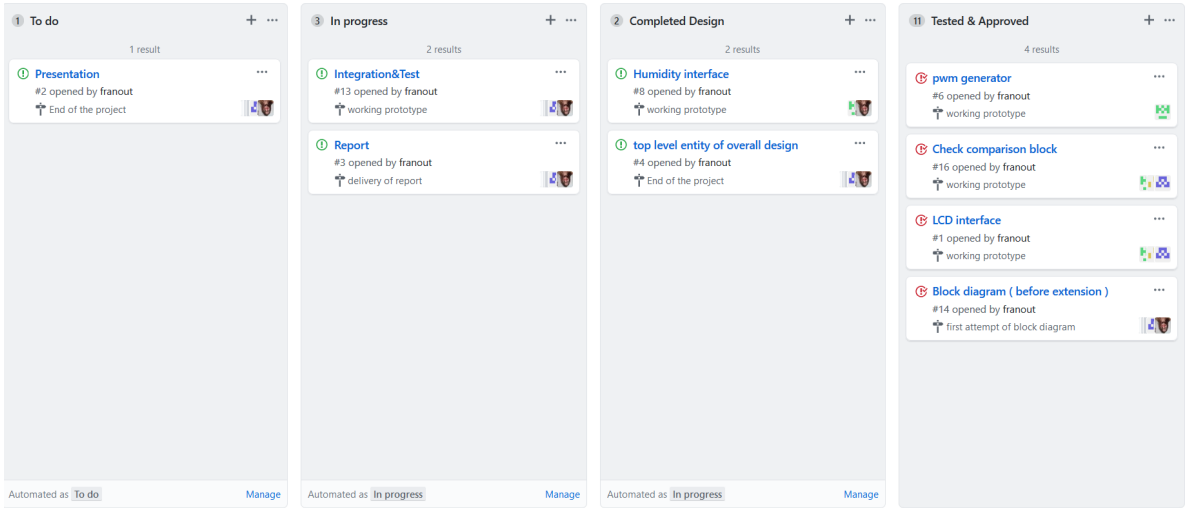
16 Personal Reports

Work done by person

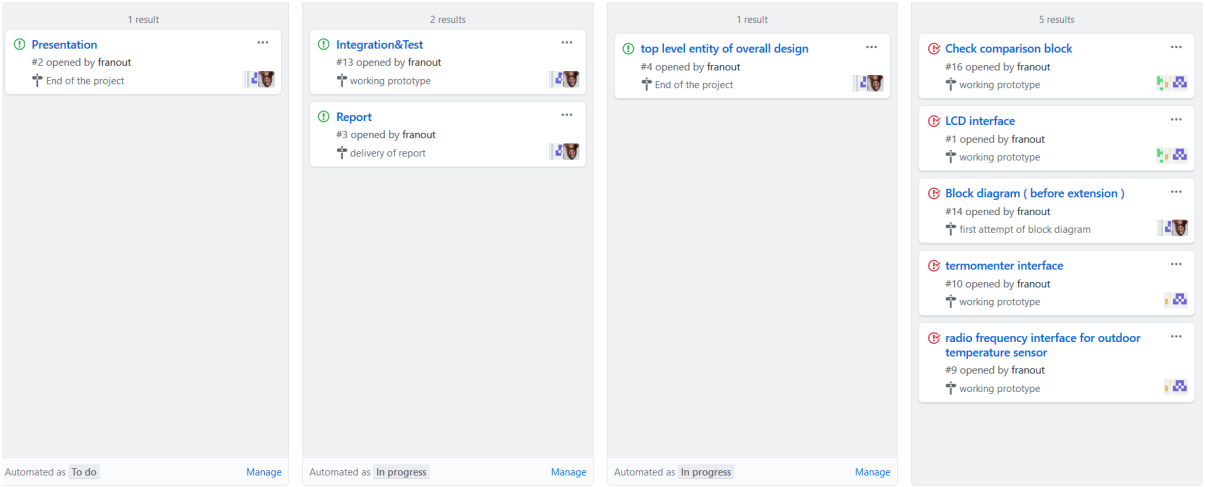
- Francesco Angione:



- Ernesto Lozano Calvo:



- Candi Wu:



- Hong Zhou: