Design specification and Test report for design:  
Temperature and humidity sensor

By:

Ibraheem Al - Nuaimi

Emil Andersson

# Table of contents

[Table of contents 2](#_Toc501103951)

[1 Introduction 4](#_Toc501103952)

[2 Top Level Implementation 5](#_Toc501103953)

[2.1 Block Diagram 5](#_Toc501103954)

[2.2 Interface 6](#_Toc501103955)

[3 Sub components 6](#_Toc501103956)

[3.1 Sensor\_ctrl 6](#_Toc501103957)

[3.2 Calculation 7](#_Toc501103958)

[3.3 Binary\_to\_bcd 7](#_Toc501103959)

[3.4 Print\_values 8](#_Toc501103960)

[3.5 lcd\_controller 8](#_Toc501103961)

[3.6 update 9](#_Toc501103962)

[4 Testbench 10](#_Toc501103963)

[4.1 Block Diagram 10](#_Toc501103964)

[4.2 Test cases 10](#_Toc501103965)

[4.3 Specification 10](#_Toc501103966)

[4.4 Testbench results 11](#_Toc501103967)

[5 Test report 12](#_Toc501103968)

[5.1 Negative temperature test 12](#_Toc501103969)

[5.2 Positive temperature test 12](#_Toc501103970)

# Introduction

In this VHDL course as our project we have chosen to implement a program that shows you the temperature and relative humidity in the current environment. This information is given to you through an OLED display.

This is possible by using the SI7006 sensor which contains functionality that measures temperature and humidity. To be able to implement this design we have downloaded the i2c master from the web, i.e. we have not written the i2c master ourselves.

We want to create this design to test ourselves, to see if we can solve the problem and also we want to use all the tools that we have learned in this course, all in one project.

# Top Level Implementation

## Block Diagram

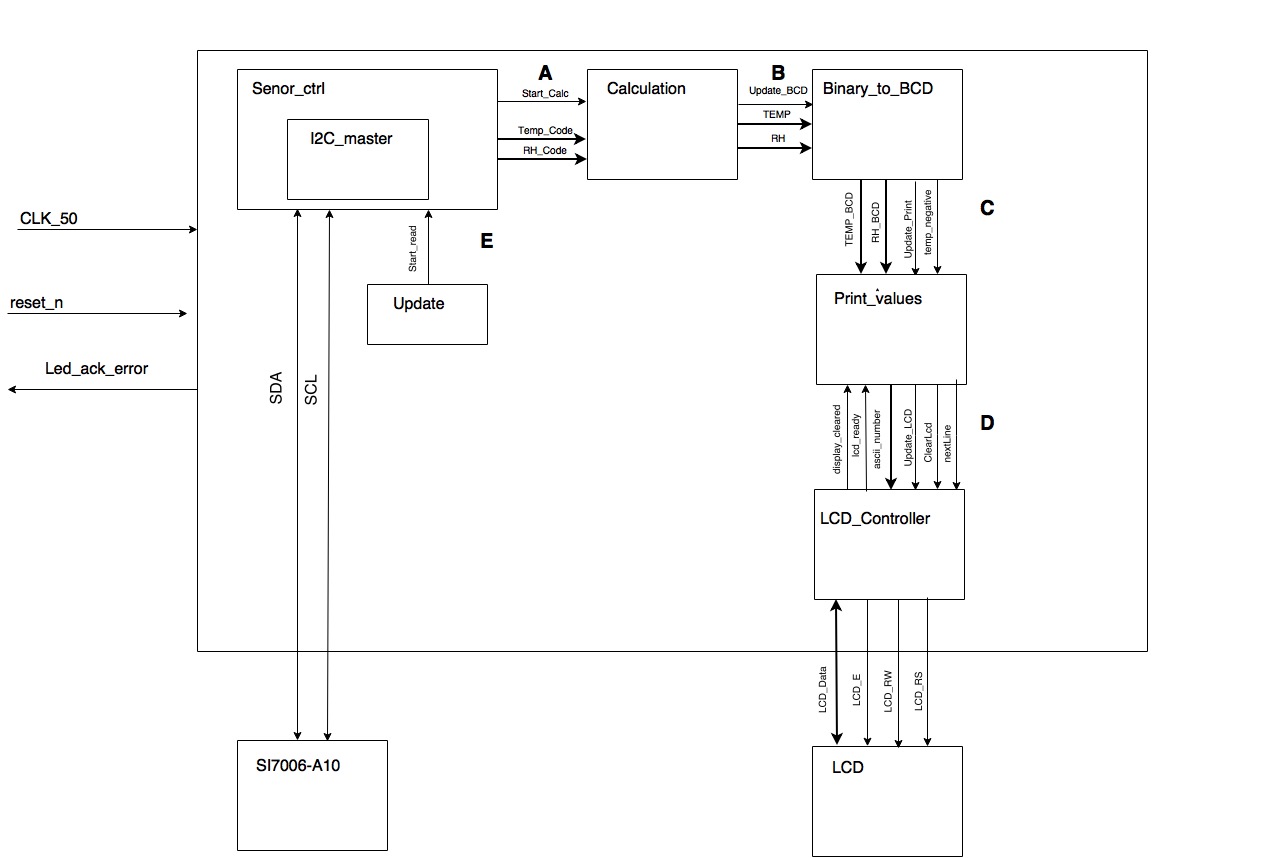


Figure 1 Top level Block Diagram

| **Figure ref.** | **Source** | **Signal name** | **Description** |
| --- | --- | --- | --- |
| A | Sensor\_ctrl | start\_calc, temp\_code, rh\_code, | Temp\_Code and RH\_Code are vectors which contains the temp/humidity code we receive from sensor slave. start\_calc is telling calculation to start |
| B | Calculation | TEMP, RH, update\_bcd | TEMP is converted into degree celcius. RH is converted into percents. When update\_bcd goes high, a new BCD conversion can start |
| C | Binary\_to\_bcd | temp\_bcd, rh\_bcd, temp\_negative, update\_print | Temp\_bcd and rh\_bcd contains value in BCD. Temp\_negative indicates if there is a negative temperature. Update\_print tells print\_values component to start |
| D | Print\_Values | Update\_lcd, lcd\_ready, print\_ascii, clearLcd, display\_cleared, nextLine | Different commands to the lcd\_controller component |
| E | Update | Start\_read | Tells sensor\_ctrl to make a new measurement |

Table 1 Signal descriptions for signals in the figure above.

## Interface

### Generics

| **Name** | **Type** | **Range** | **Description** |
| --- | --- | --- | --- |
|  |  |  |  |

Table 2 Generics for top Level entity.

### Signals

| **Name** | **Direction** | **Type** | **Description** |
| --- | --- | --- | --- |
| Clk\_50 | in | std\_logic | 50MHz clock |
| SDA | bidirectional | Std\_logic | Serial Data Line |
| SCL | bidirectional | std\_logic | Serial Clock Line |
| LCD\_Data | Inout | 8 bit std\_logic\_vector | Data bus |
| LCD\_E | out | std\_logic | Enable |
| LCD\_RW | out | std\_logic | Read / Write |
| LCD\_RS | out | std\_logic | Register select signal, either Data or Command |
| Led\_ack\_arror | out | Std\_logic | Connected to led that goes high if ack error is detected |
| Reset\_n | in | Std\_logic | Resets display |

Table 3 Signals for top Level entity.

# Sub components

## Sensor\_ctrl

### Block diagram

The sensor is using the i2c master as a component (see block diagram 2.1)

### Specification

The sensor\_ctrl is using i2c master as a component to communicate between master and slaves. This Is possible by using two bidirectional open drain lines – SCL and SDA which is the interface towards the i2c bus. The i2c master is downloaded from eewikis webpage, i.e. we have not created this component ourselves. The speed on the i2c bus will be set to 100 000 Hz and the clock frequency is set to 50MHz (20ns period time). Two commands will be sent to the slave, one command for temperature and one for relative humidity. The slave will then return two 8 bit vectors for each command which will be put into the 16 bit vectors called rh\_code and temp\_code. This component sends the code from the two measurements to Calculation.

### Error handling

If i2c master component detects an ack error a led will be turned on.

## Calculation

### Block diagram

None. (Very simple implementation.)

### Specification

The result of the measurements needs to be converted into either degree celcius or percent depending on which measurment has been done. We will use two calculation expressions (see below) in order to convert the temp- and humidity code into its actual binary value. Since calculations in VHDL is quite tricky to handle, we have choosed to “up scale” the different constants we need for the calculation, for instance 175,72 is up scaled into 1757 and so on (Since the temperature has an accuracy of +-0,5 this wont be a problem). These values needs to be converted into BCD in order to display it on the LCD. Therefore both values is sent the Binary\_to\_BCD component.

The temp code which is received from the SI7006 sensor will be converted into degrees celcius by using the following expression: Temperature( °C ) = ((175,72 \* temp\_code)/65536) – 46,85

The humidity code received from the sensor will be converted into percent by using the following expression: Relative humidity( % ) = ((125 \* rh\_code)/65536) - 6

### Error handling

A humidity measurement will always return “XXXXXX10” in the LSB field and “XXXXXX00” for the temperature measurement. Therefore, an error check is done by reading the bits to see that they are correct. Otherwise we return to idle state.

## Binary\_to\_bcd

### Block diagram

None. (Very simple implementation.)

### Specification

The Binary\_to\_BCD component receives the temperatur/humidity binary value and to be able to write that value on the lcd we need to convert it into binary coded decimals (BCD). In this component we use a “negative number” signal which is set high if the component recieves a negative value. It then “flips” the negative value into a positive and then starts to shift. This is done by catergorizing the binary value into hundreds, tens and units. In our case if we receive e.g. the value 159 (10011111 in binary) the value is actually 15,9 since we have up scaled it in the previous component. However 159 is represented by 0001 0101 1001 in BCD, and as you can see each 4bit category represents a value (1 5 9). When the shifting is in progress it shifts one bit at a time and if any of the 4bit category is greater than four we add three. On the web you can find many different examples on how to implement a binary value into BCD so for this component we followed one of those examples(nandland.com) and changed it to fit our design.

### Error handling

None.

## Print\_values

### Block Diagram

None. (Very simple implementation.)

### Specification

An update is received when bcd conversion is done, the print values component then starts by telling the lcd controller to clear the current printout. When the lcd controller is done, print values will begin to send one character at a time each 700us. When it’s done with the first line print values tells the lcd controller to switch to second line.

### Error handling

**None**

## lcd\_controller

### Block diagram

None. (Very simple implementation.)

### Specification

The lcd display needs an initialization sequence with different commands such as function set, display clear and so on, where each command in this sequence requires a specific delay. This is done by using a state machine and a count delay process. The count delay process counts clock cycles to create the different delays that is necessary for all the commands and signals. The state machine then handles all the commands. When it’s done with the initialization it’ll wait in the idle state until we either want to send a character or clear the display.

### Error handling

We have increased all the delays to ensure that enough time passes for all sequences/commands.

## update

### Block diagram

None. (Very simple implementation.)

### Specification

A new measurement will be made every 200ms. The update component handles this by counting clock cycles and when its done an output is set high.

### Error handling

None.

# Testbench

Note that since we didn’t have time for the test bench we have not changed its block diagram or specification.

## Block Diagram../../Downloads/test_bench_diagram.png

Figure 2 Testbench Block Diagram

## Test cases

|  |  |
| --- | --- |
| **Test case number** | **Description** |
| 1 | A positive temperature and 80% humidity is received from the p\_tests process  Output is checked to see that everything works in a correct way |
| 2 | A negative temperature and 40% humidity is received from the p\_tests process  Output is checked to see that everything works in a correct way |
| 3 | We send some sort of data with an inserted error, we check the output so see if there is an error detection. |

Table 4 Test cases

## Specification

### p\_generate\_clk

The p\_generate\_clk process generates 50 MHz clock signal.

### p\_in\_stimuli

This process generates input stimuli to the top\_level. In order to fully test the design, we need a slave component of some sort to see that when we send a command to the sensor, we actually get a 16 bit measurement value back to process further.

### p\_tests

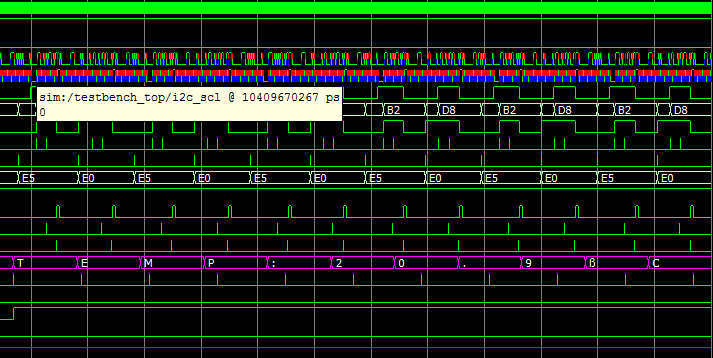
The tests process shall control all tests that we want to do. We get input stimuli and the output signals we get from the p\_out\_check process decides if the test is OK or some error has occurred.

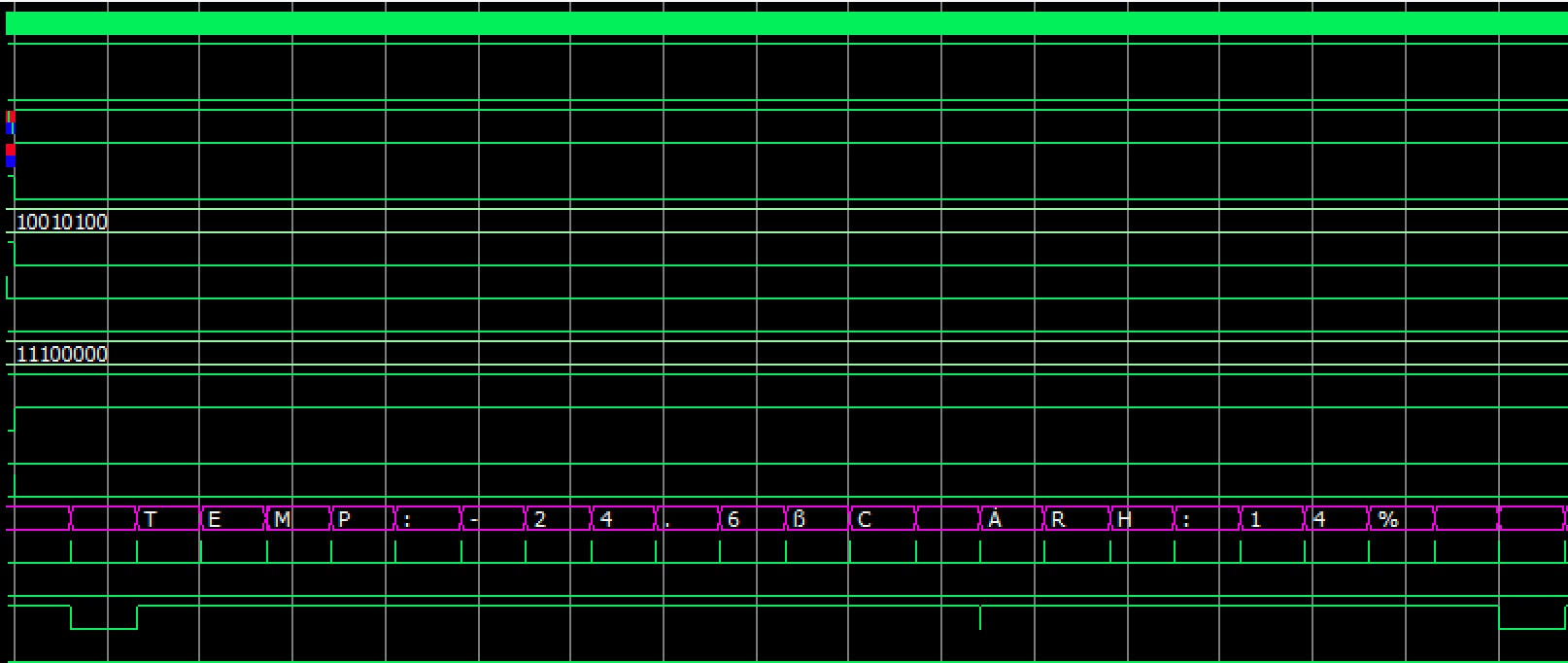
### p\_out\_check

This process receives data from top\_level and gives the result to the p\_tests process.

## Testbench results

Unfortunately, we didn’t have time to write a test bench that fully tested the functionality per to the test cases. Our ambition was from the start to do this, but along the way problems arose and our main priority was always first and foremost to deliver a stable program. However, we did have time to try out Kent Abrahamssons i2c slave with our design in simulation (see images below).





As you can see both positive and negative temperature is shown in a correct way on the data to the display. The first image shows TEMP:20.9°C, the second image shows TEMP: -24.6°C and the command for setting the address to second line (0x40) is shown as “Á” in the second image followed by RH:14%

# Test report

Several tests have been in hardware to fully test the functionality. The design is supposed to be able to show both negative and positive temperature, therefore we have tried to push these limits to ensure we have a fully functioning program.

## Negative temperature test

We have used two different methods to represent a negative value on the display. These methods are ice and cooling spray. When we used ice in a plastic bag we got a smooth transition starting from 25 degrees dropping down to -1,3 degrees (see image below). When cooling spray was used, the temperature declined to a negative value very fast. Even though both tests resulted in a negative value the transition was different between the methods and that’s what we wanted. One test to show a slow declining value and one test to show a quick and major difference in temperature.



## Positive temperature test

We also wanted to test high temperature values. This was done by using a hairdryer. As you can see in the image below we managed to create a quite high temperature.

