# **Tutorial :** How to create a driver for a I2C distance sensor matching a *DEO-nano-SoC FPGA* chip with *Quartus*

Tatiana Dehon & Charles Moulin

May 2021

## 1 Introduction

This project course aims to build a hardware/software driver by combining programmable architectures like **PIC** - or **Arduino** in this case - and re-configurable ones like **FPGA Altera** interfaces with a specific sensor.

This requires basic knowledge in digital hardware design, tools and architectures (**VHDL**, **FPGA**), programming languages (here we have learned by ourselves and used as correctly as possible a lot **VHDL** and also a little bit of **C**, which we had already used before) and also a minimum understanding of the I2C sensor basic protocol.

We have made some research inside and outside the platforms that were given to us to make our project work the optimal way. This document exposes the steps we have followed to build our program, the issues we had to face and the way we have corrected them, every time it was possible.

All the files necessary to run and understand the codes we have written an explained here are available on our **GitHub** through clicking this link[1]. The demonstration video is there as well.

## 2 Sensor characteristics

We picked a distance sensor : *Ultra Sonic SRF02* provided by the SEMI, and shown in figure 1 left (click on the number of a figure to see it).

Its behavior is described in the figures 1 right and 2 and its technical interesting specifications are described in the tab below.

Note that we have set the command to send to the command register to 0x51 so that we can get the measure result in centimeters and we have not changed the device address even if it was possible because it was not useful for us to accomplish our work.

| Feature | Value |
|---|---|
| Voltage | 5 V |
| Current | 4 mA Typ. |
| Frequency | 40 kHz |
| Maximum Range | 6 m |
| Minimum Range | 15 cm |
| Size | 24 mm x 20 mm x 17 mm |
| Address | 0xE0 |
| Type | slave I2C |
| # registers | 6 |
| LED | Yes - 1 red |

## 3 Sensor modeling

We have created an artificial sensor - with an **input clock**, a **sda** and a **scl** and **three registers** + the **output register** - inside the project folder to simulate our distance sensor activity. We can do that since we know the basic functioning of an I2C sensor : ours follows exactly the basic one, described at the figure 3.

This can be seen in the `I2C_m.vhd` file which describes the whole component described above, with the functioning of the module from the figure 3, through the several states.

## 4 Testbench

A testbench is a file created to simulate the sensor through incoming signals and to see if everything works well in the driver. The file we will talk about in this section is the `I2C_M_TB.vhd` file.

[1] https://github.com/Titania7/Distance_sensor_I2C

## 4.1 Driver imposition

Since our sensor type is slave, we need a master I2C code as a driver in order to control it.

We have simply based our coding on the files available on the SEMI lab computers, which was a default master-slave I2C code. We have modified it to match our sensor.

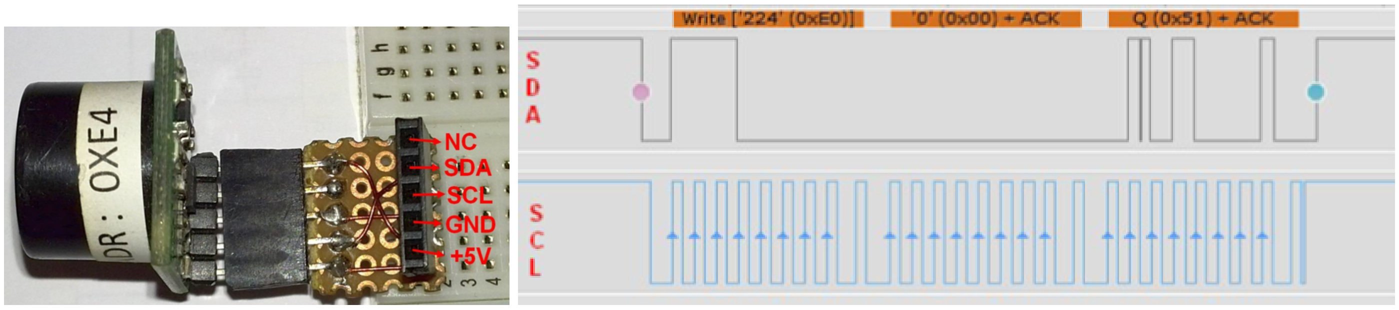


Figure 1: Our distance sensor + behaviour

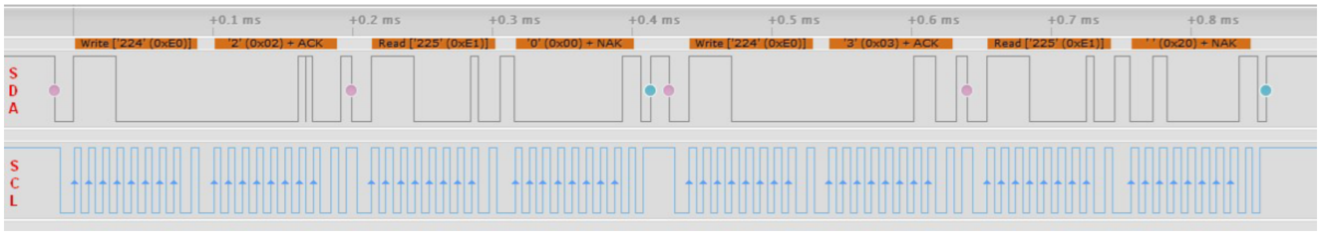


Figure 2: Our sensor behaviour

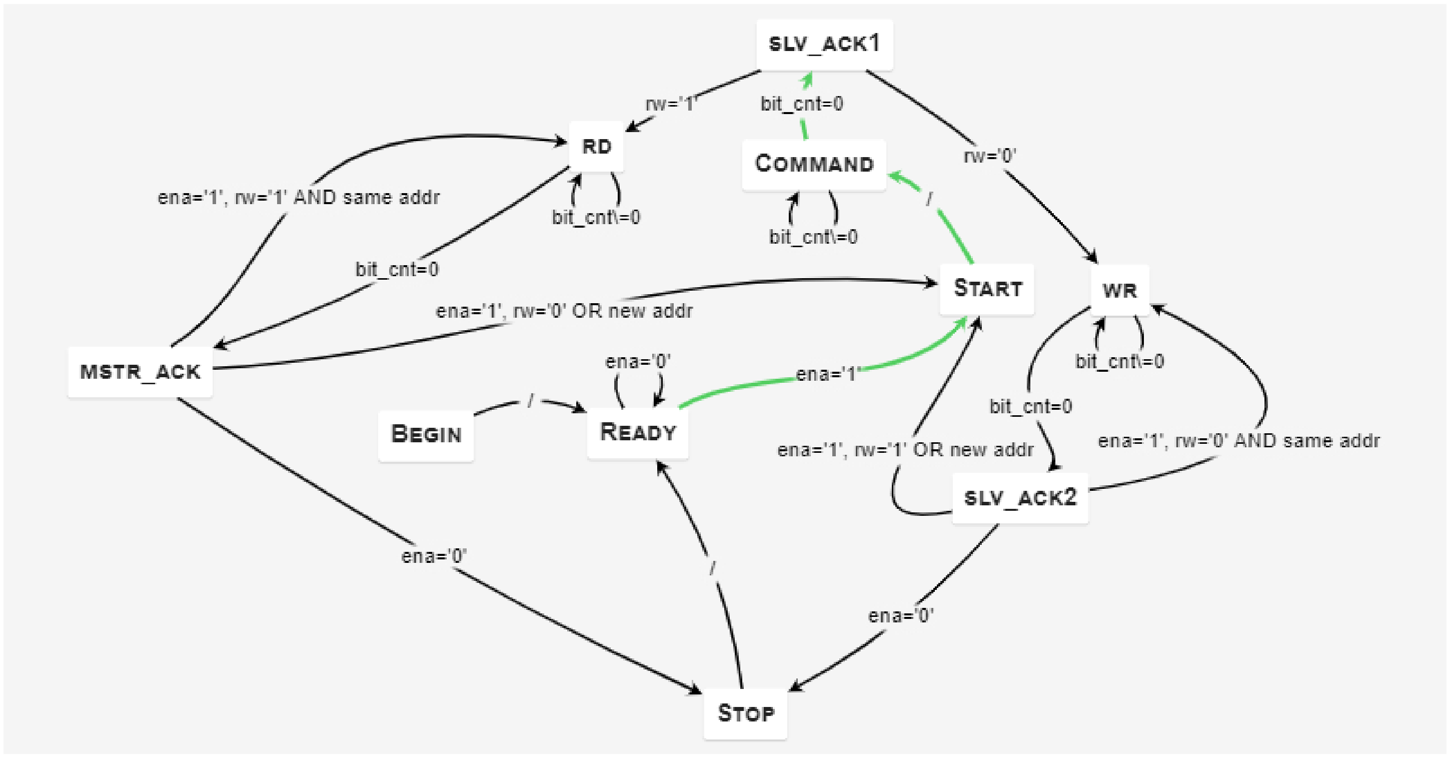


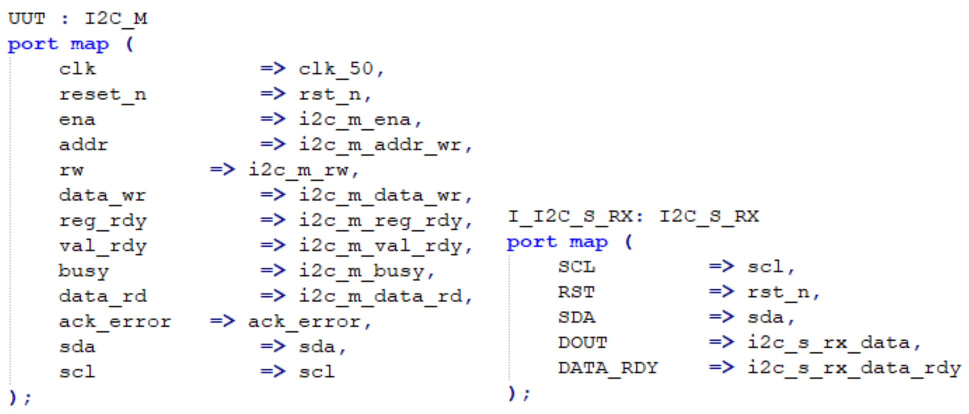
Figure 3: I2C classical behaviour

If you take a look at the lines 80 to 85 of this file you will see that the addresses of the device and the registers have been specified.

Then from line 117 to line 145, the master driver and the slave sensor are wired with their respective data connected altogether.

Finally, there is also a trick we made for the `rw` value, remember, this value determined whether the I2C goes to *read* mode or *write* mode. Since our sensor is read-only, we have forced it to '1' to be sure to never get in an accidental write mode.

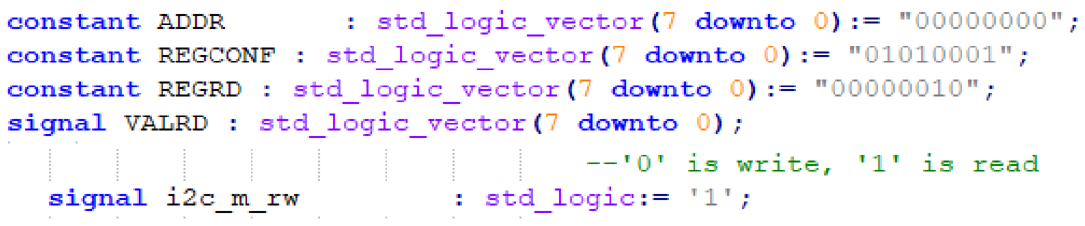
These three particularities are shown in figures 4 and 5.



```
UUT : I2C_M
port map (
    clk             => clk_50,
    reset_n         => rst_n,
    ena             => i2c_m_ena,
    addr            => i2c_m_addr_wr,
    rw          => i2c_m_rw,
    data_wr         => i2c_m_data_wr,
    reg_rdy         => i2c_m_reg_rdy,
    val_rdy         => i2c_m_val_rdy,
    busy            => i2c_m_busy,
    data_rd         => i2c_m_data_rd,
    ack_error   => ack_error,
    sda             => sda,
    scl             => scl
);
```

```
I_I2C_S_RX: I2C_S_RX
port map (
    SCL         => scl,
    RST         => rst_n,
    SDA         => sda,
    DOUT        => i2c_s_rx_data,
    DATA_RDY    => i2c_s_rx_data_rdy
);
```

Figure 4: I2C_M_TB.vhd particularities



```
constant ADDR       : std_logic_vector(7 downto 0):= "00000000";
constant REGCONF : std_logic_vector(7 downto 0):= "01010001";
constant REGRD : std_logic_vector(7 downto 0):= "00000010";
signal VALRD : std_logic_vector(7 downto 0);
                                    --'0' is write, '1' is read
  signal i2c_m_rw          : std_logic:= '1';
```

Figure 5: I2C_M_TB.vhd particularities

We have also modified some of the scripts of the original working of the master driver so that it never gets in the write mode either.

Its behavior can be modeled this way :

1. **s0** : Addressing for the configuration :
   s0 if `busy = 1`
   s1 if `busy = 0` ;

2. **s1** : Configuration :
   s1 if `reg_rdy = 0`
   s2 if `reg_rdy = 1` ;

3. **s2** : Awaiting of the measure :
   s2 if `val_rdy = 0`
   s3 if `val_rdy = 1` ;

4. **s3** : Choice of the register for the measure :
   s3 if `busy = 1`
   s4 if `busy = 0` ;

5. **s4** : Awaiting for the master to be ready :
   s4 if `reg_rdy = 0`
   s5 if `reg_rdy = 1` ;

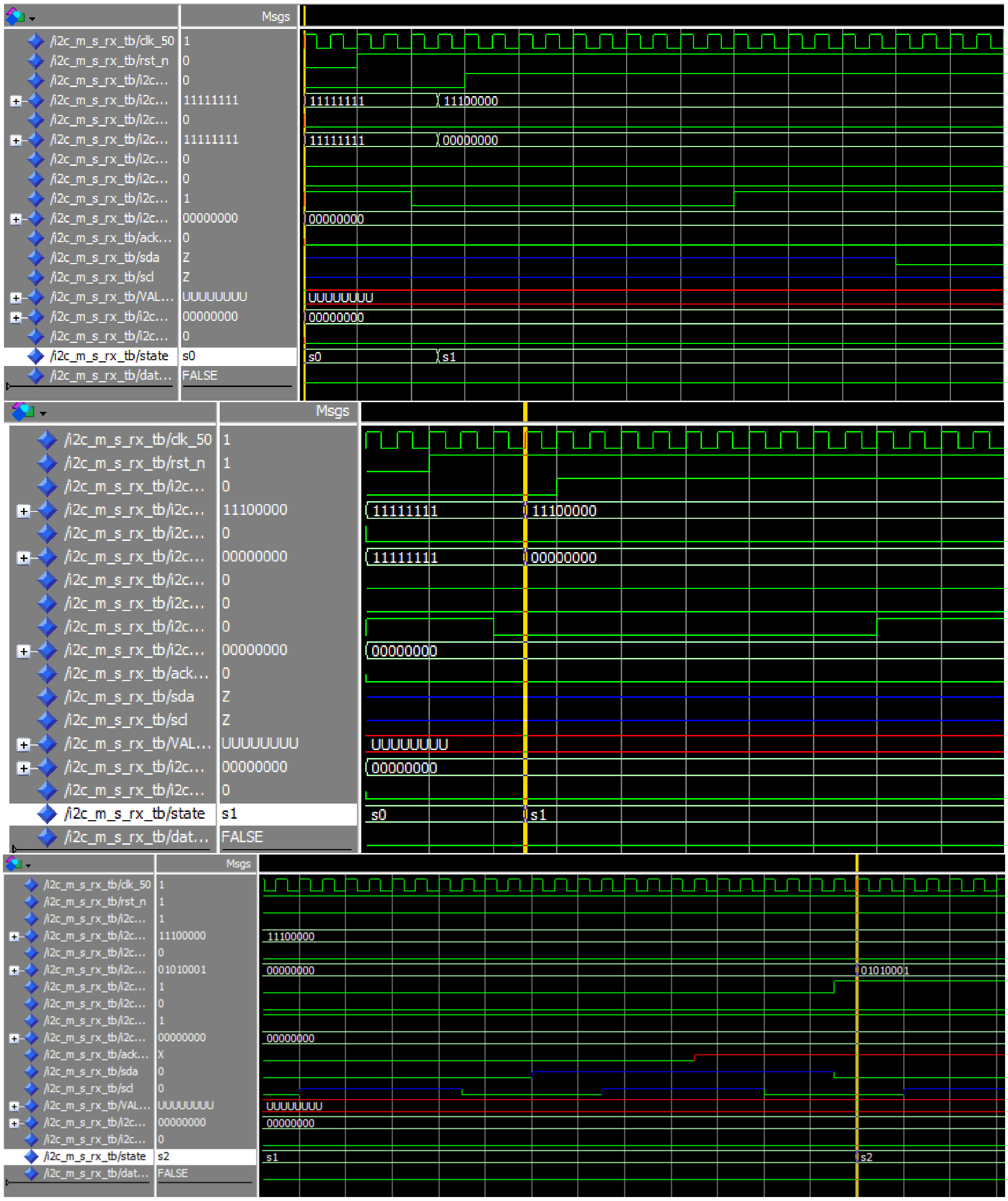
6. **s5** : The master reads the measure :
   s5 if `val_rdy = 0`
   s3 if `val_rdy = 1`.

## 4.2 Sensor simulations using ModelSim

Once these arrangements have been noted, the expected series we will watch as sda output shall be s0 - s1 - s2 - s3 - s4 - s5 - s3 - s4 - s5 - s3 - s4 - s5 - etc. We have run the simulation and the good news is that it follows exactly this scheme, this you can see at figures 6, 7 and 8.
The values of s3, s4 and s5 alternate so quickly at the end that if the zoom is not big enough, s3 and s5 are not visible but are there anyway (see last figure 8).

# 5 C coding to display results with DEO board

Once the toughest part (last point) has been done, it is time to prepare the display on the Arduino DEO-nano-SoC chip. We have simply used the serial output data to display the

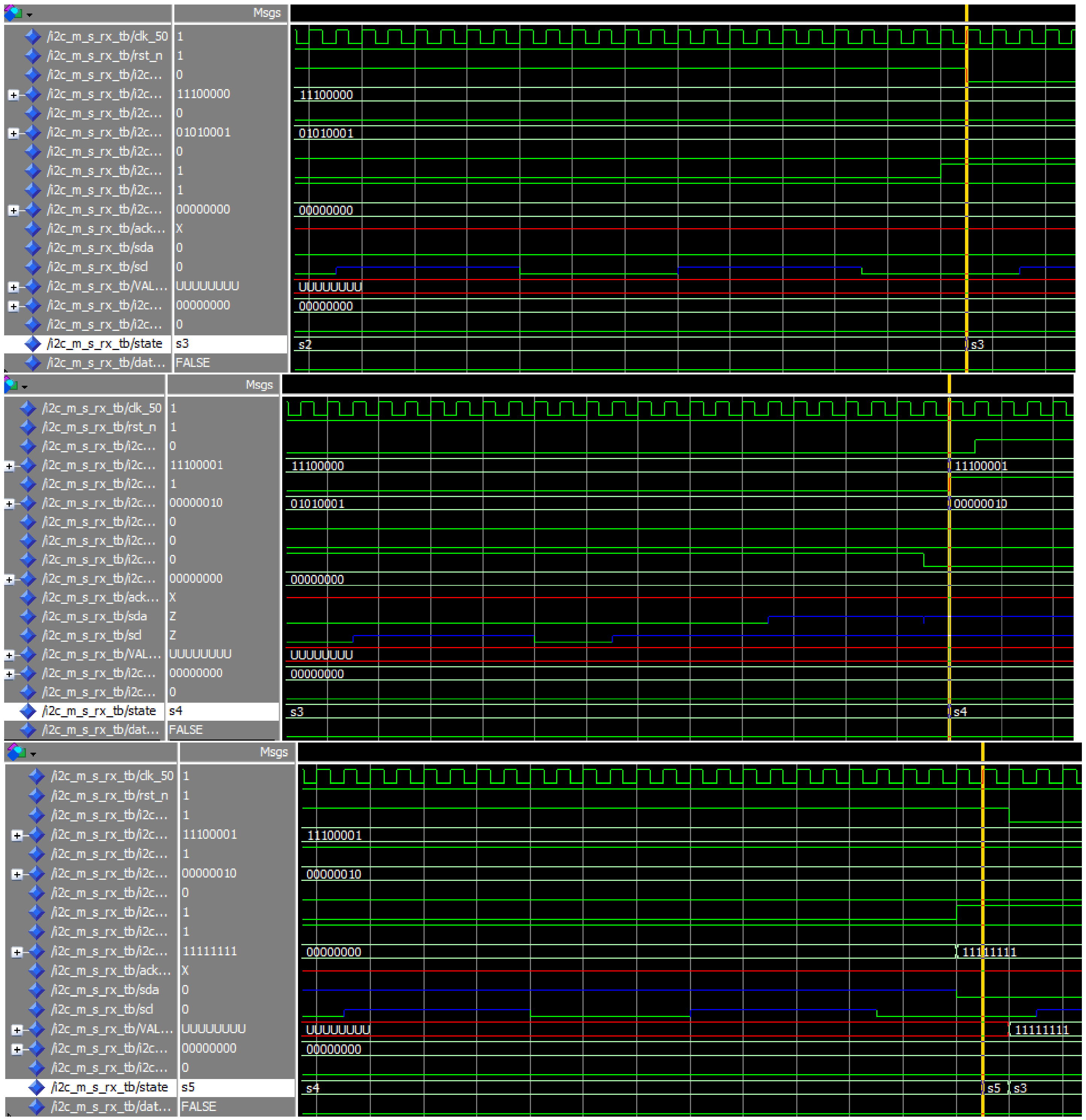


```
Msgs
/i2c_m_s_rx_tb/clk_50   1
/i2c_m_s_rx_tb/rst_n    0
/i2c_m_s_rx_tb/i2c...   0
/i2c_m_s_rx_tb/i2c...   11111111
/i2c_m_s_rx_tb/i2c...   0
/i2c_m_s_rx_tb/i2c...   11111111
/i2c_m_s_rx_tb/i2c...   0
/i2c_m_s_rx_tb/i2c...   0
/i2c_m_s_rx_tb/i2c...   1
/i2c_m_s_rx_tb/i2c...   00000000
/i2c_m_s_rx_tb/ack...   0
/i2c_m_s_rx_tb/sda      Z
/i2c_m_s_rx_tb/scl      Z
/i2c_m_s_rx_tb/VAL...   UUUUUUUU
/i2c_m_s_rx_tb/i2c...   00000000
/i2c_m_s_rx_tb/i2c...   0
/i2c_m_s_rx_tb/state    s0
/i2c_m_s_rx_tb/dat...   FALSE
```

```
Msgs
/i2c_m_s_rx_tb/clk_50   1
/i2c_m_s_rx_tb/rst_n    1
/i2c_m_s_rx_tb/i2c...   0
/i2c_m_s_rx_tb/i2c...   11100000
/i2c_m_s_rx_tb/i2c...   0
/i2c_m_s_rx_tb/i2c...   00000000
/i2c_m_s_rx_tb/i2c...   0
/i2c_m_s_rx_tb/i2c...   0
/i2c_m_s_rx_tb/i2c...   0
/i2c_m_s_rx_tb/i2c...   00000000
/i2c_m_s_rx_tb/ack...   0
/i2c_m_s_rx_tb/sda      Z
/i2c_m_s_rx_tb/scl      Z
/i2c_m_s_rx_tb/VAL...   UUUUUUUU
/i2c_m_s_rx_tb/i2c...   00000000
/i2c_m_s_rx_tb/i2c...   0
/i2c_m_s_rx_tb/state    s1
/i2c_m_s_rx_tb/dat...   FALSE
```

```
Msgs
/i2c_m_s_rx_tb/clk_50   1
/i2c_m_s_rx_tb/rst_n    1
/i2c_m_s_rx_tb/i2c...   1
/i2c_m_s_rx_tb/i2c...   11100000
/i2c_m_s_rx_tb/i2c...   0
/i2c_m_s_rx_tb/i2c...   01010001
/i2c_m_s_rx_tb/i2c...   1
/i2c_m_s_rx_tb/i2c...   0
/i2c_m_s_rx_tb/i2c...   1
/i2c_m_s_rx_tb/i2c...   00000000
/i2c_m_s_rx_tb/ack...   X
/i2c_m_s_rx_tb/sda      0
/i2c_m_s_rx_tb/scl      0
/i2c_m_s_rx_tb/VAL...   UUUUUUUU
/i2c_m_s_rx_tb/i2c...   00000000
/i2c_m_s_rx_tb/i2c...   0
/i2c_m_s_rx_tb/state    s2
/i2c_m_s_rx_tb/dat...   FALSE
```

Figure 6: Simulations measures with ModelSim


```
Msgs
/i2c_m_s_rx_tb/clk_50     1
/i2c_m_s_rx_tb/rst_n      1
/i2c_m_s_rx_tb/i2c...     0
/i2c_m_s_rx_tb/i2c...     11100000
/i2c_m_s_rx_tb/i2c...     0
/i2c_m_s_rx_tb/i2c...     01010001
/i2c_m_s_rx_tb/i2c...     0
/i2c_m_s_rx_tb/i2c...     1
/i2c_m_s_rx_tb/i2c...     1
/i2c_m_s_rx_tb/i2c...     00000000
/i2c_m_s_rx_tb/ack...     X
/i2c_m_s_rx_tb/sda        0
/i2c_m_s_rx_tb/scl        0
/i2c_m_s_rx_tb/VAL...     UUUUUUUU
/i2c_m_s_rx_tb/i2c...     00000000
/i2c_m_s_rx_tb/i2c...     0
/i2c_m_s_rx_tb/state      s3
/i2c_m_s_rx_tb/dat...     FALSE
```

```
Msgs
/i2c_m_s_rx_tb/clk_50     1
/i2c_m_s_rx_tb/rst_n      1
/i2c_m_s_rx_tb/i2c...     0
/i2c_m_s_rx_tb/i2c...     11100001
/i2c_m_s_rx_tb/i2c...     1
/i2c_m_s_rx_tb/i2c...     00000010
/i2c_m_s_rx_tb/i2c...     0
/i2c_m_s_rx_tb/i2c...     0
/i2c_m_s_rx_tb/i2c...     0
/i2c_m_s_rx_tb/i2c...     00000000
/i2c_m_s_rx_tb/ack...     X
/i2c_m_s_rx_tb/sda        Z
/i2c_m_s_rx_tb/scl        Z
/i2c_m_s_rx_tb/VAL...     UUUUUUUU
/i2c_m_s_rx_tb/i2c...     00000000
/i2c_m_s_rx_tb/i2c...     0
/i2c_m_s_rx_tb/state      s4
/i2c_m_s_rx_tb/dat...     FALSE
```

```
Msgs
/i2c_m_s_rx_tb/clk_50     1
/i2c_m_s_rx_tb/rst_n      1
/i2c_m_s_rx_tb/i2c...     1
/i2c_m_s_rx_tb/i2c...     11100001
/i2c_m_s_rx_tb/i2c...     1
/i2c_m_s_rx_tb/i2c...     00000010
/i2c_m_s_rx_tb/i2c...     0
/i2c_m_s_rx_tb/i2c...     1
/i2c_m_s_rx_tb/i2c...     1
/i2c_m_s_rx_tb/i2c...     11111111
/i2c_m_s_rx_tb/ack...     X
/i2c_m_s_rx_tb/sda        0
/i2c_m_s_rx_tb/scl        0
/i2c_m_s_rx_tb/VAL...     UUUUUUUU
/i2c_m_s_rx_tb/i2c...     00000000
/i2c_m_s_rx_tb/i2c...     0
/i2c_m_s_rx_tb/state      s5
/i2c_m_s_rx_tb/dat...     FALSE
```

Figure 7: Simulations measures with ModelSim

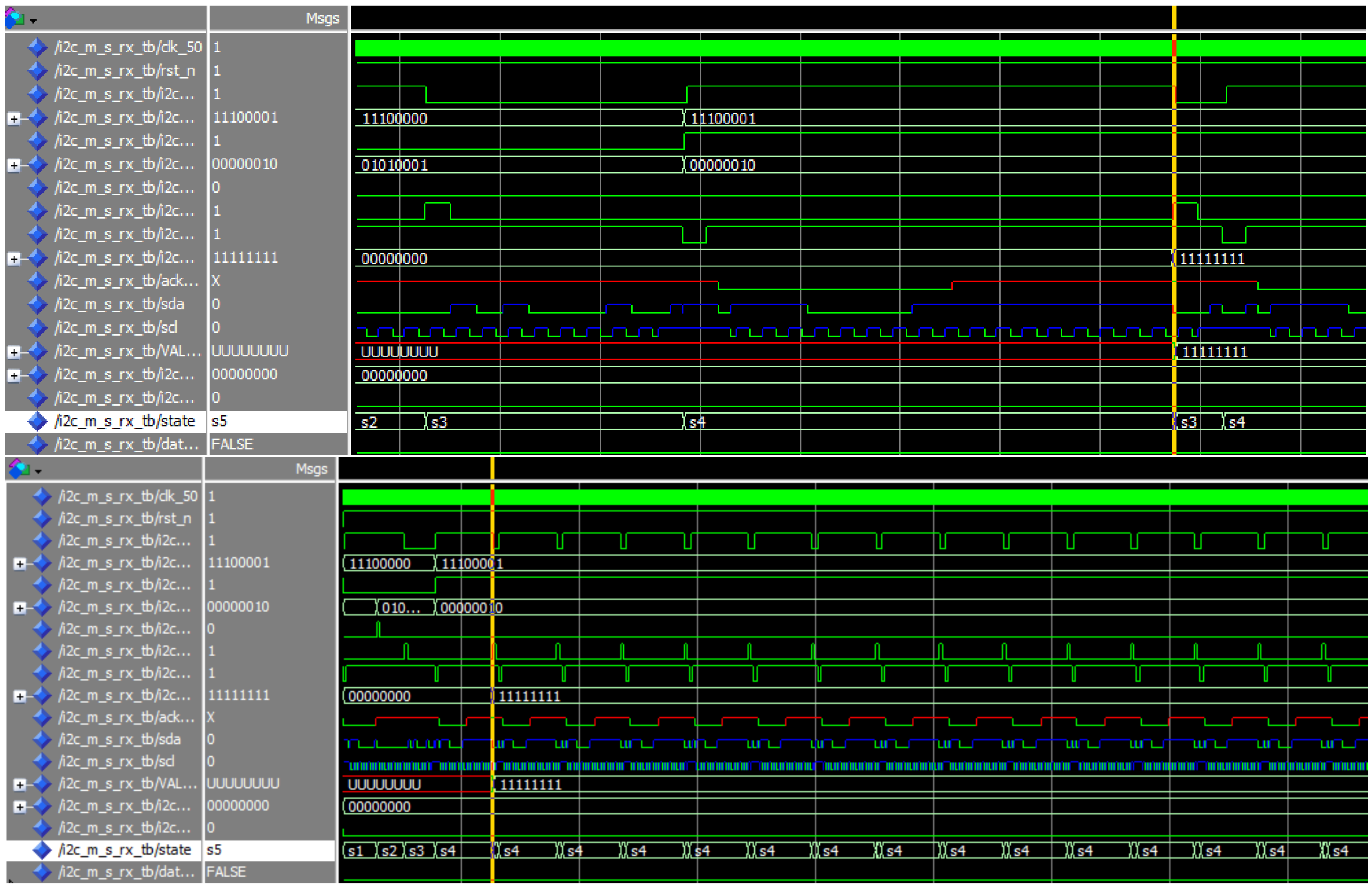


Figure 8: Simulations measures with ModelSim

result of the measure.

The file to look at is `main.c` and the sda output of the sensor is connected to the driver, itself connected to the pin of the board.

Note that we have used a file coming from a similar project and we have simply modified it to match our goal. At line 44 we have defined the pointer to the output of the sensor (sda) so it means the input of the card.

This variable receives the measured value and then displays it inside the serial port, at line 107.

## 6 Implementation on the board using Putty

Once all of this has been completed, the last thing left to do is to compile and push an executable file inside the chip.

To do so, we used Putty with a guide provided by the SEMI and managed to install the executable file (see figure 9) on the board. This file is the one called `Distance_sensor-Dehon-Moulin` inside our GitHub repository.

### 6.1 Final test

After wiring the chip to the sensor (no need for pull-up resistances, as mentioned in the data sheet) - see figure 10 -, we just wire it to the computer using an Ethernet cable, we power it with its plug and we connect its output to a USB port (we chose COM15) to be able to receive the data.

The result is shown at figure 11.

### 6.2 Issues encountered

The obvious finding is that we have a problem in the data transfer because the output of the sensor is always 0 cm, at any distance we place it in relation to the obstacle (a wall).

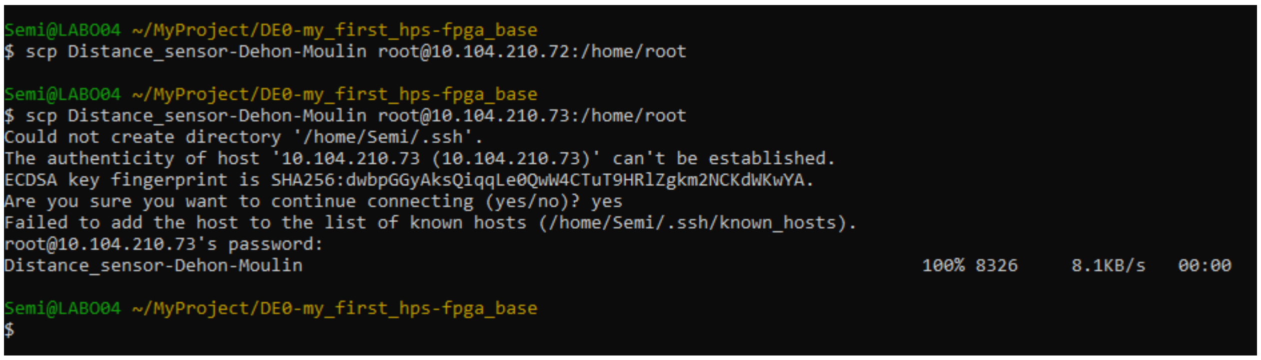
We tried to correct it by several ways :

- We first simply verified that our connections were correct : they were so the problem should be somewhere else ;
- Then we verified whether the code presented a malfunction somewhere but if it did we did not find it.

Plus, our testbench showed successful results so at first sight it might not be the source of the zero-data ;

- We tried to wire the connected pins to an oscilloscope to see if the signals were right but unfortunately the connecting extremities of the oscilloscope wires were far too big not to unplug the sensor at the same time.

Sadly, we did not have enough time to locate exactly the source of the malfunction but the ideal simulation we have made showed that, a priori, this should have worked.



```
Semi@LABO04 ~/MyProject/DE0-my_first_hps-fpga_base
$ scp Distance_sensor-Dehon-Moulin root@10.104.210.72:/home/root

Semi@LABO04 ~/MyProject/DE0-my_first_hps-fpga_base
$ scp Distance_sensor-Dehon-Moulin root@10.104.210.73:/home/root
Could not create directory '/home/Semi/.ssh'.
The authenticity of host '10.104.210.73 (10.104.210.73)' can't be established.
ECDSA key fingerprint is SHA256:dwbpGGyAksQiqqLe0QwW4CTuT9HRlZgkm2NCKdWKwYA.
Are you sure you want to continue connecting (yes/no)? yes
Failed to add the host to the list of known hosts (/home/Semi/.ssh/known_hosts).
root@10.104.210.73's password:
Distance_sensor-Dehon-Moulin                                                100% 8326     8.1KB/s   00:00

Semi@LABO04 ~/MyProject/DE0-my_first_hps-fpga_base
$
```

Figure 9: The code is pushed in the board

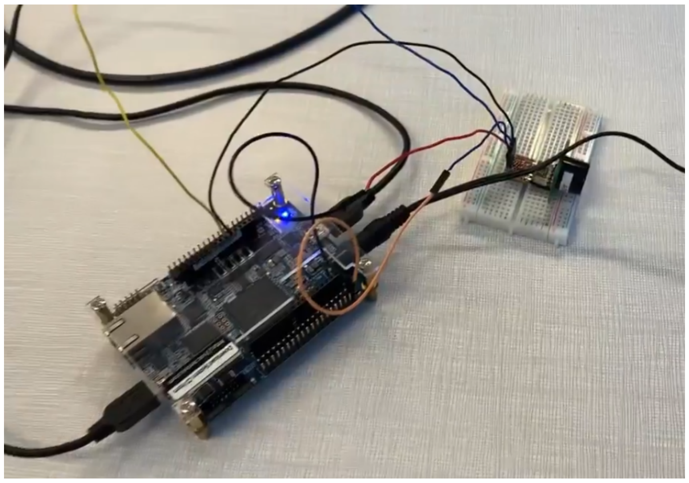
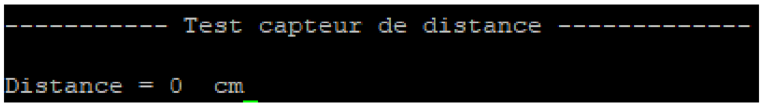


Figure 10: The chip is wired



```
----------- Test capteur de distance -------------

Distance = 0  cm
```

Figure 11: Output of the sensor through serial USB port

## 7 Conclusions

Even if this project led us to a final issue that we had not enough time to correct despite the many hours of work and the energy we have spent on it, the core of our work and understandings have been very instructive.

In the end, we have :

- Discovered the VHDL coding and exploited it inside a real application project ;
- Learned how to use Quartus meanwhile and how to simulate components within in order to use a real one ;

- Searched solutions by ourselves to solve our issues and communicated with the SEMI crew whenever our tasks were very challenging ;
- Worked with a team-spirit to tackle a support which still was unknown to both of us ;
- Learned to used a lot of tools in general, during the whole time this project lasted. In the end, even if the final result makes us quite frustrated, we know we could try it again and maybe with the same time, we could reach an effective sensor thanks to the skills we have acquired during this project.

Special thanks to Mohammed Chelkha who was always there during the lab sessions, was incredibly brave in front of the errors hidden inside our codes and who has helped us more than a lot !