Hardware Software Platforms Project Presentation

Control of light sensor with DE0-Nano-SoC

Deffrennes T. & Finet C.

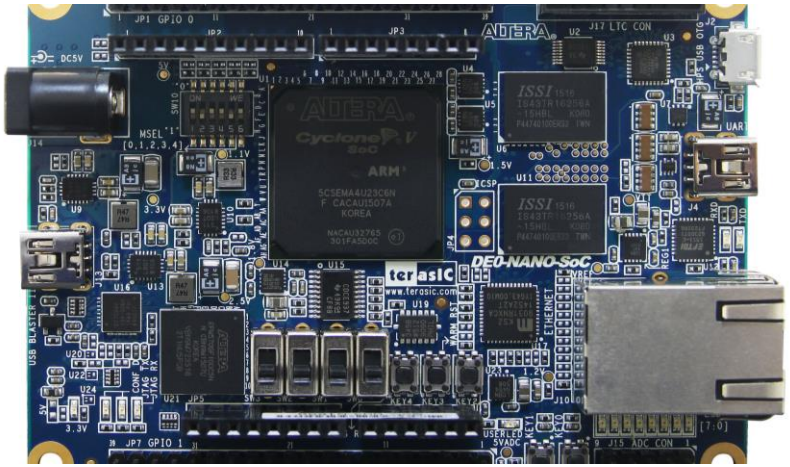
Plan

- Objective
- BH1710FVC light sensor
- I2C
  - -Protocol
  - -Driver
- Design steps
  - - Hardware part
  - - Software part
- Current results
- Expected results

Objective

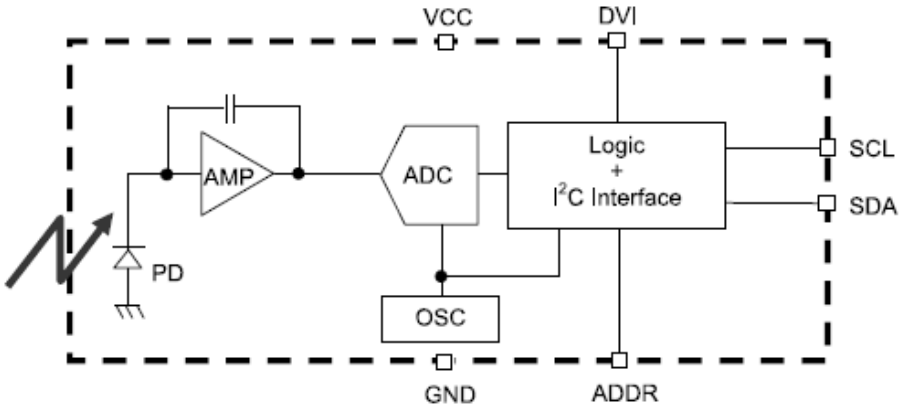
- Tutorial presentation to show how drive the BH1710FVC light sensor with a FPGA.

- FPGA used: The DE0-Nano-SoC board from Terasic. Hardware design platform built around the Altera System-on-Chip (SoC) FPGA.



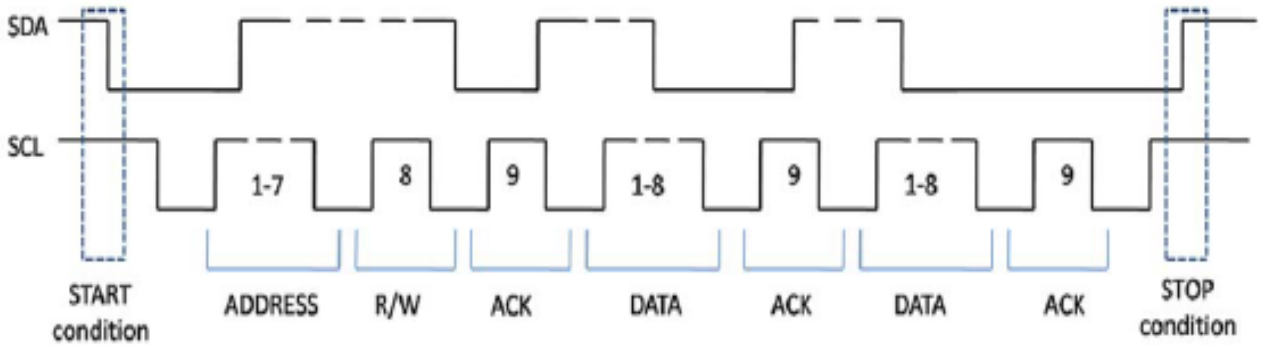
BH1710FVC light sensor

- Digital ambient light sensor on 16 bits (unit = lux)
- I2C bus interface:
  - SDA
  - SCL
- Master-slave communication with FPGA



Measurement Mode	Measurement Time.	Resolution
H-Resolution Mode	Typ. 120ms.	1 Lx.
M-Resolution Mode	Typ. 16ms.	4 Lx.
L-Resolution Mode	Typ. 2.9ms.	32 Lx.

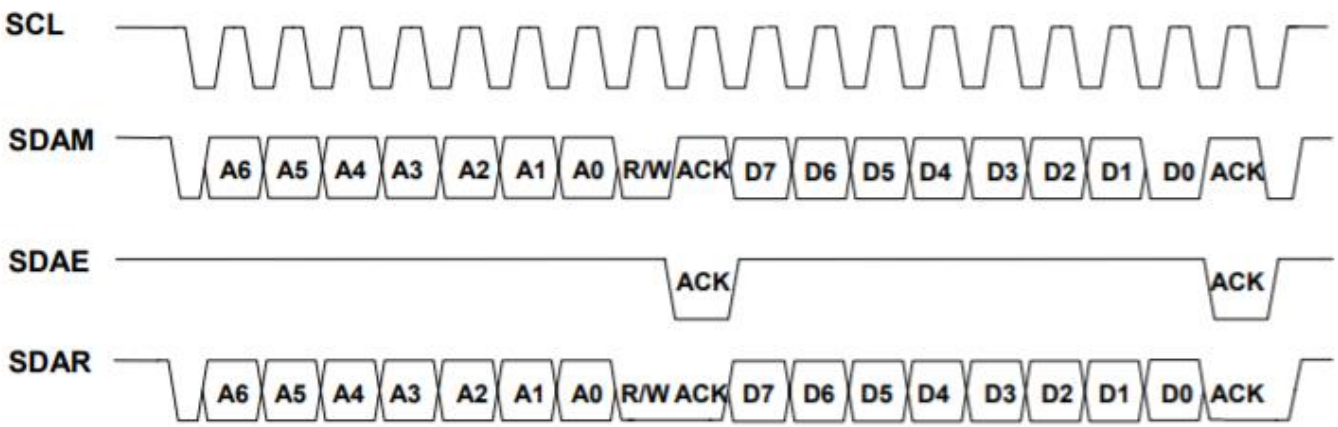
I2C protocol



- Start and stop conditions
- Slave address (7bits) + R/W bit (access mode)
- Data byte: byte by byte transfer of data on SDA line (register address, opecode or data read from slave)
- ACK and NACK

❑ Writing data

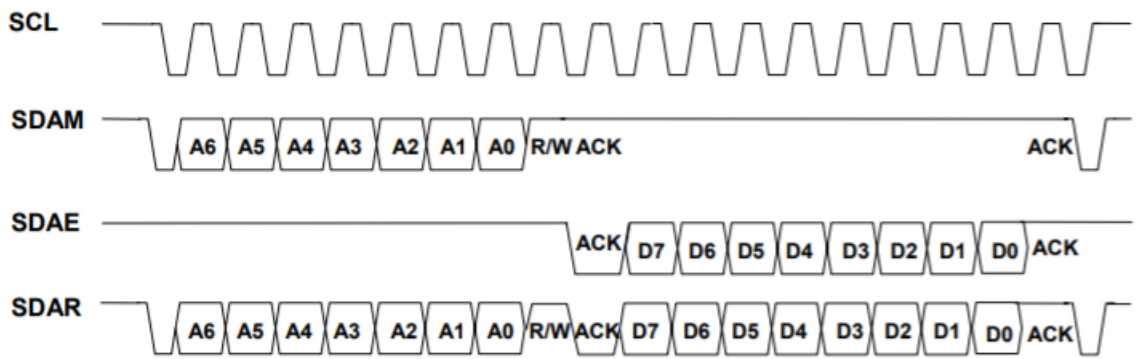
1) Master sends the slave address
2) Selection of the writing mode (R/W to ‘0’)
3) Master sends the 8 bits data



SCL: Master-imposed clock
SDAM: SDA levels imposed by the master
SDAE: SDA levels imposed by the slave
SDAR: Resulting actual SDA levels

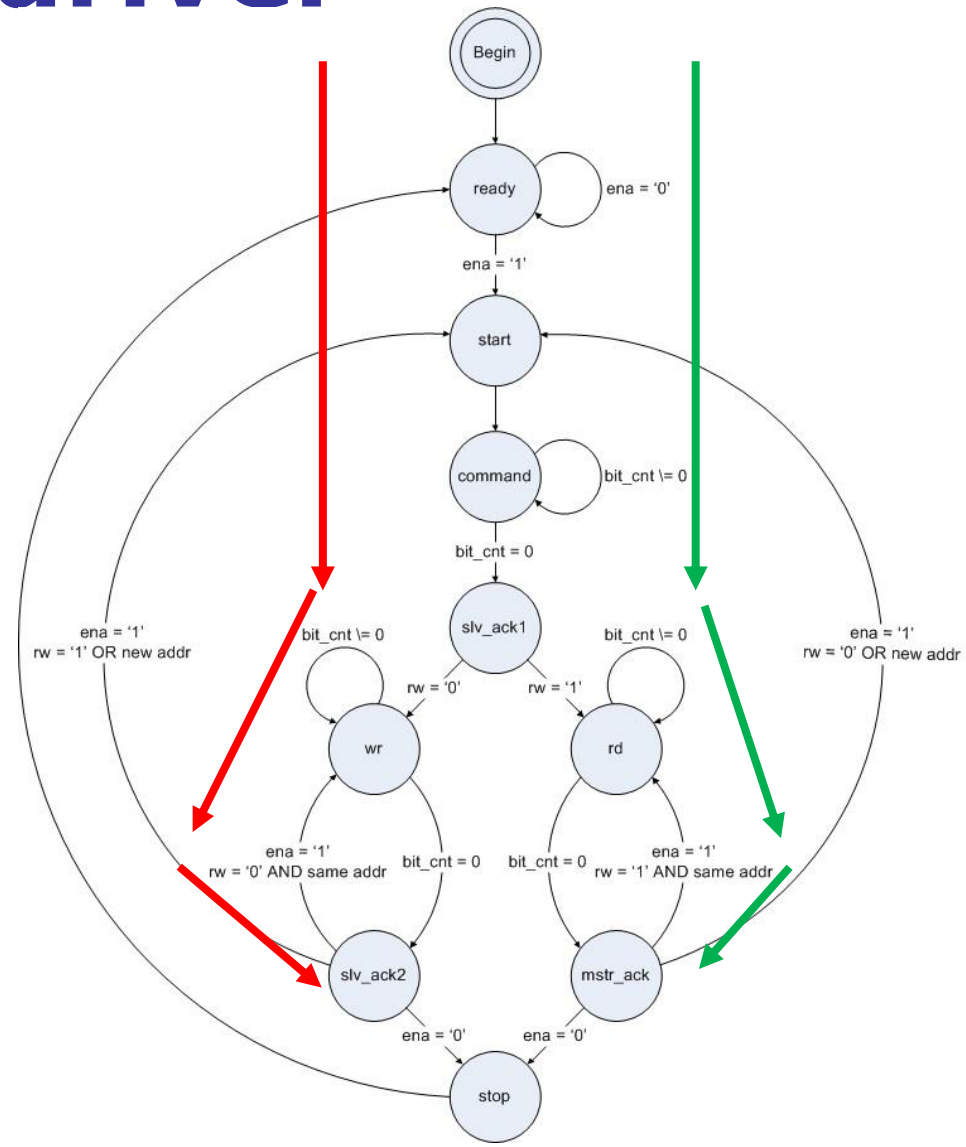
❑ Reading data

1) Master sends the slave address and R/W='1' and then waits for the ACK
2) ACK is set by the slave, then the slave sends the data to the SDA
3) Master sets the ACK to 0 to continue reading or 1 to stop transmission



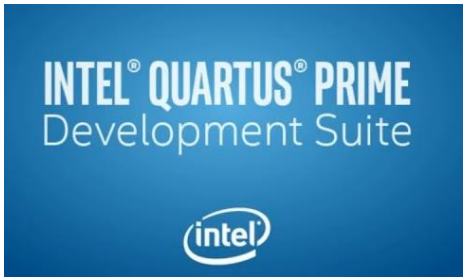
I2C driver

- State machine
  - Command state communicates the address and R/W bit
  - Write state waits for the 8 bits to write to slave
  - Read state waits for the 8 bits to read from slave



- VHDL code to perform the state machine operation

Hardware part



❑ Tools:

- ➢ Install the Digital design tool **Quartus Lite Edition** to design your RTL and implement on the FPGA Altera

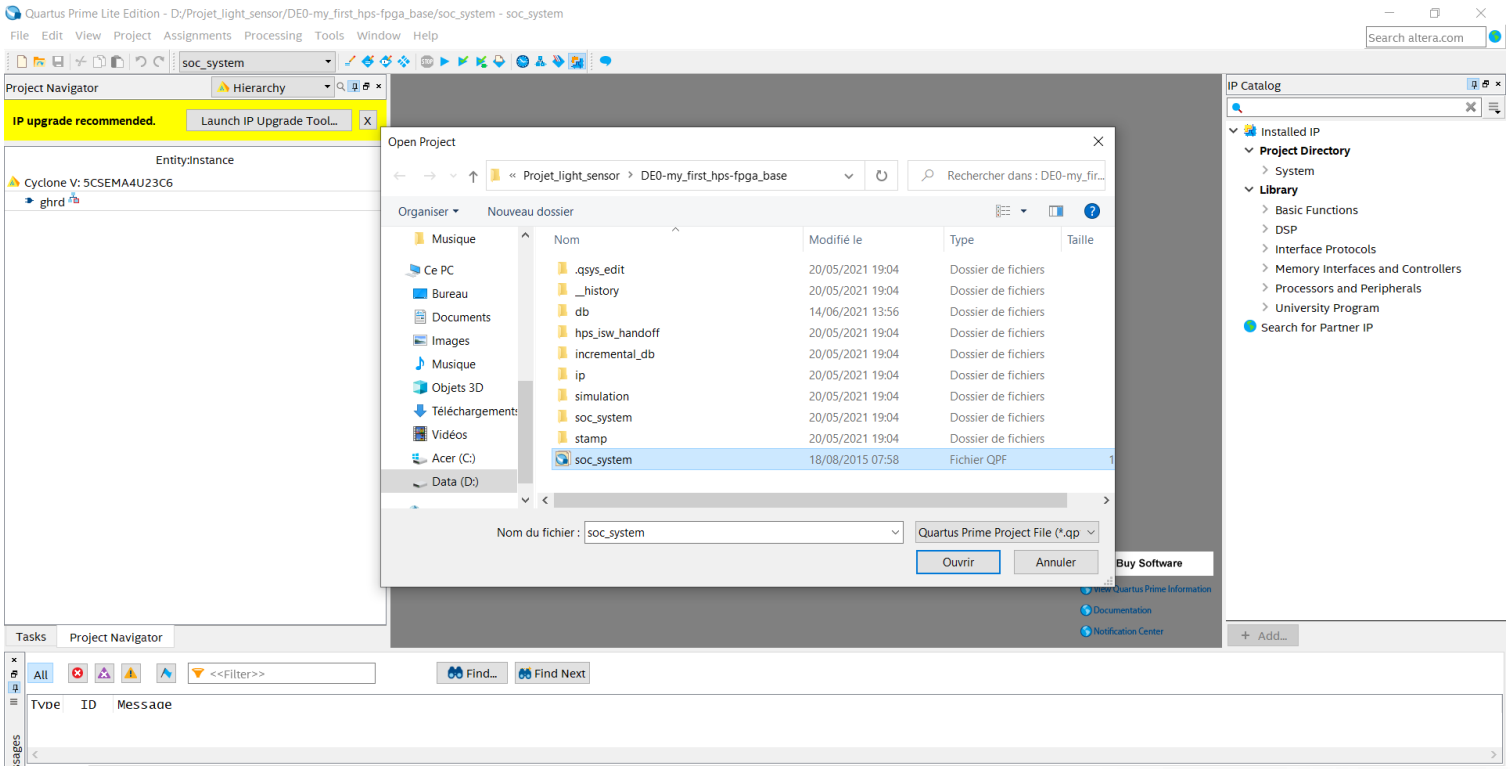
* It is free, no license is required you just have to register with an Intel account

* The tool comes with Cyclone V support already selected (but check it anyway)

- ➢ Install **ModelSim**, the signals simulator

❑Steps:

1) Download the DE0 nano SoC golden and open the file soc_system.qpf as project in Quartus (File -> Open Project):



2) Creation of a VHDL code in the project to use the light sensor:

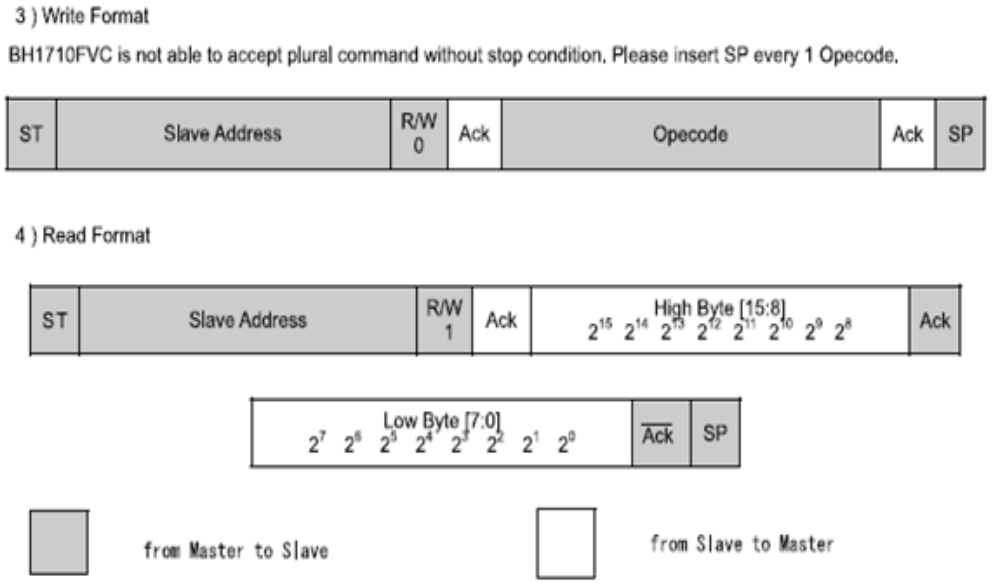
- The clock frequency of the board used (50MHz)
- A reset
- SDA /SCL as bidirectional
- 2 output registers of 8 bits (16 bits data)

```
entity light is
  port(

    --System
    FPGA_CLK1_50   : in std_logic;
    reset : in std_logic;

    --I2C signals.
    sda : inout std_logic;
    scl : inout std_logic;

    -- Two registers to read the sensor
    reg1 : out std_logic_vector(15 downto 8);
    reg2 : out std_logic_vector(7 downto 0)
  );
end light;
```

2) Creation of a VHDL code in the project to read the light sensor:

- The light bloc is based on the I2C master state machine (mapping)

- Behaviour: single reading repeated over time



- The operations of the VHDL code are:

- Writing operation (R/W = '0'): master defines the opecode for the type of measurement and resolution.
- Reading operation (R/W = '1') with first the high byte and then the low byte.

3) The routing (clock, reset, registers) configuration in the FPGA must be done with Platform Designer (Tools -> Platform Designer):

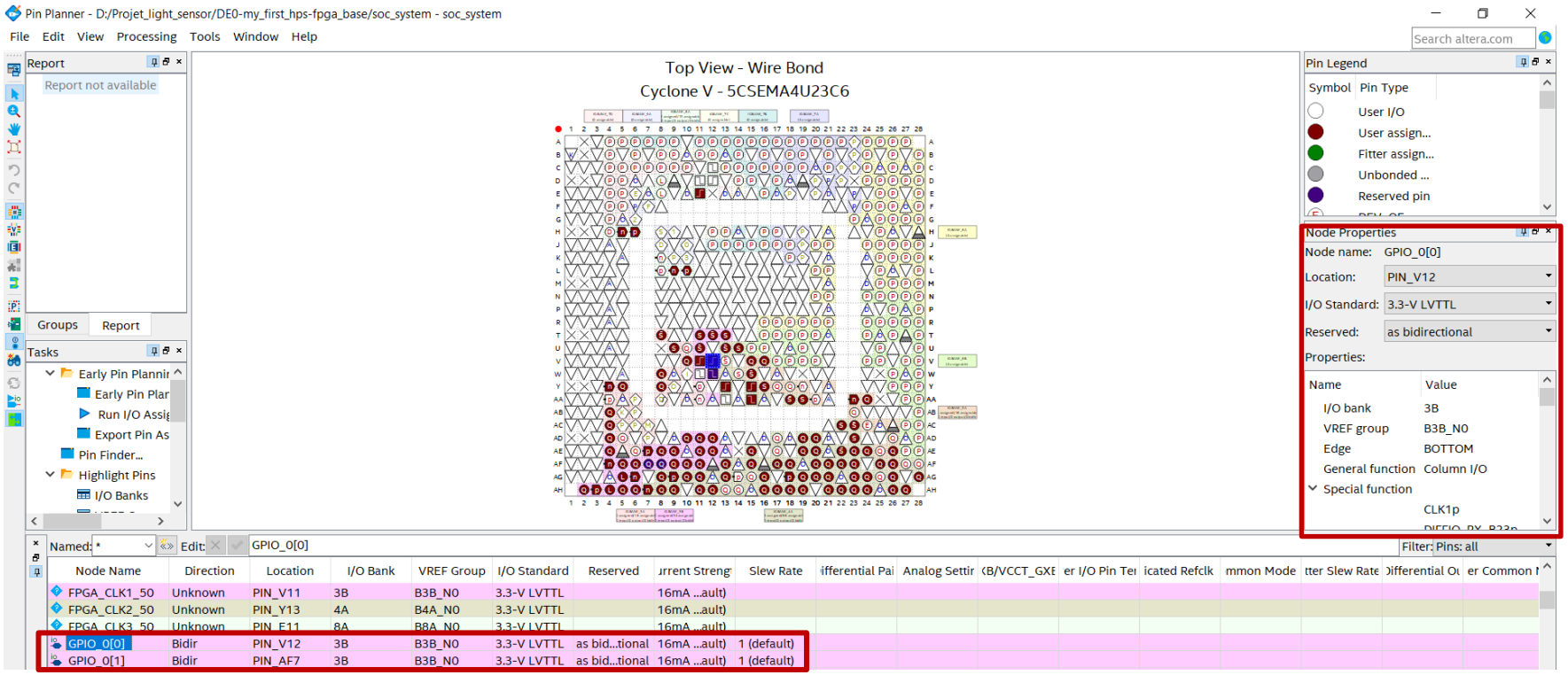
System Contents

System: soc_system **Path:** hps_0

Use	Connections	Name	Description	Export	Clock	Base	End
☑		**onchip_memory2_0**	On-Chip Memory (RAM or ROM)...				
		clk1	Clock Input	Double-click to export	**clk_0**		
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk1]	0x0000_0000	0x0000_ffff
		reset1	Reset Input	Double-click to export	[clk1]		
☑		**jtag_uart**	JTAG UART Intel FPGA IP				
		clk	Clock Input	Double-click to export	**clk_0**		
		reset	Reset Input	Double-click to export	[clk]		
		avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0002_0000	0x0002_0007
		irq	Interrupt Sender	Double-click to export	[clk]		
☑		**fpga_only_mas...**	JTAG to Avalon Master Bridge				
		clk	Clock Input	Double-click to export	**clk_0**		
		clk_reset	Reset Input	Double-click to export			
		master	Avalon Memory Mapped Master	Double-click to export	[clk]		
		master_reset	Reset Output	Double-click to export			
☑		**intr_capturer_0**	Interrupt Capture Module				
		clock	Clock Input	Double-click to export	**clk_0**		
		reset_sink	Reset Input	Double-click to export	[clock]		
		avalon_slave_0	Avalon Memory Mapped Slave	Double-click to export	[clock]	0x0003_0000	0x0003_0007
		interrupt_receiver	Interrupt Receiver	Double-click to export	[clock]	IRQ 0	IRQ 31
☑		**clk_0**	Clock Source				
		clk_in	Clock Input	**clk**	*exported*		
		clk_in_reset	Reset Input	**reset**			
		clk	Clock Output	Double-click to export	clk_0		
		clk_reset	Reset Output	Double-click to export			
☑		**reg1**	PIO (Parallel I/O) Intel FPGA IP				
		clk	Clock Input	Double-click to export	**clk_0**		
		reset	Reset Input	Double-click to export	[clk]		
		s1	Avalon Memory Mapped Slave	[illegible]	[clk]	0x0000_0000	0x0000_000f
		external_connection	Conduit	**reg1_external_conn...**			
☑		**reg2**	PIO (Parallel I/O) Intel FPGA IP				
		clk	Clock Input	Double-click to export	**clk_0**		
		reset	Reset Input	Double-click to export	[clk]		
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_0010	0x0000_001f
		external_connection	Conduit	**reg2_external_conn...**			

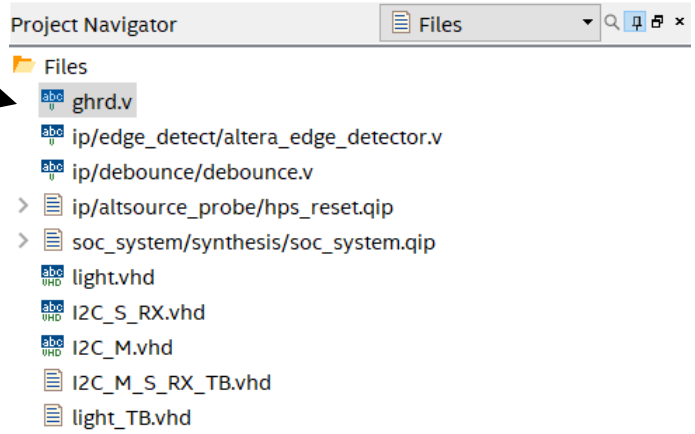
- Then click on generate HDL to create HDL design files for synthesis

4) Pin Planner has to be used to allocate the pin that will be used on the FPGA (Assignments -> Pin Planner):



GPIO_0[0] for SCL (bidirectional)
GPIO_0[1] for SDA (bidirectional)

5) Update of the ghrd.v file




```
127  //==========================================
128  //  REG/WIRE declarations
129  //==========================================
130  // internal wires and registers declaration
131    wire [1:0]  fpga_debounced_buttons;
132    wire [7:0]  fpga_led_internal;
133    wire        hps_fpga_reset_n;
134    wire [2:0]  hps_reset_req;
135    wire        hps_cold_reset;
136    wire        hps_warm_reset;
137    wire        hps_debug_reset;
138    wire [27:0] stm_hw_events;
139    wire [7:0] wire1;
140    wire [7:0] wire2;
```

```
146  //=====================================================
147  //  Structural coding
148  //=====================================================
149
150
151  soc_system u0 (
152        //Clock&Reset
153      .reg1_external_connection_export (wire1),
154      .reg2_external_connection_export (wire2),
155      .clk_clk                         (FPGA_CLK1_50 ),
156      .reset_reset_n                   (1'b1         ),
```

```
239  light Mylight(
240      .FPGA_CLK1_50(FPGA_CLK1_50),
241      .reset(hps_fpga_reset_n),
242      .reg1(wire1),
243      .reg2(wire2),
244      .scl(GPIO_0[0]),
245      .sda(GPIO_0[1]),
246  );
```

code to be added (verilog language)

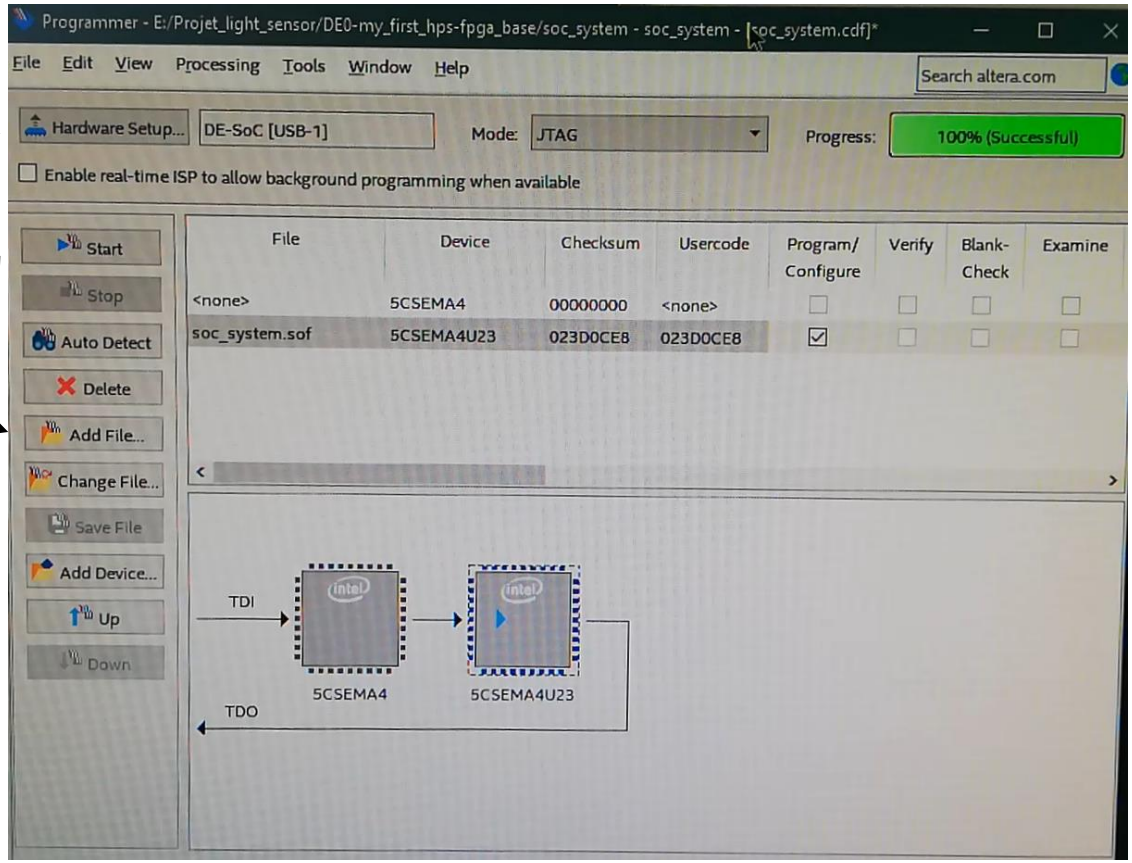
6) Transfer the hardware program on the FPGA with Programmer (Tools -> Programmer)

After connecting the Hardware part of the FPGA to the computer:

a) The DE0-Nano-SoC must be recognized

b) Add the soc_system.sof file (Obtained after compiling the entire project)

c)Start the uploading



Software part

❑Tools:

- For the software development the **Embedded Development Suite** EDS need to be installed. The same version as Qartus is mandatory.

- **PUTTY** can be installed to access the board using the USB port (serial communication) and can connect to the board and use the terminal to see the execution of the program running on the ARM processors. Any printf on the program will appear on that terminal.

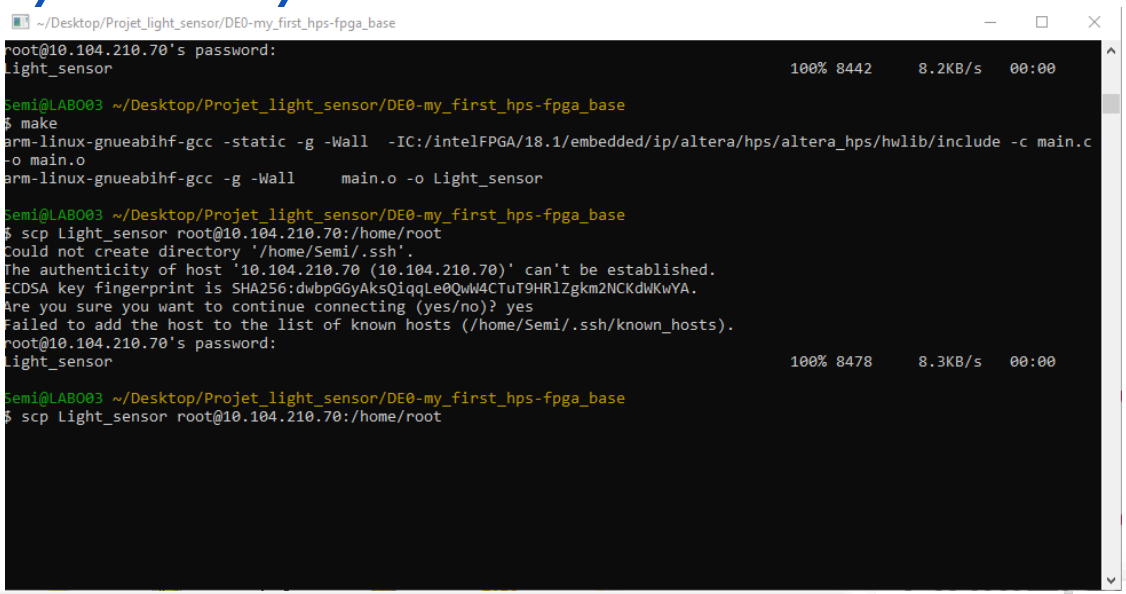
❑Steps:

1) Connect the software part of the FPGA to the computer with the usb cable and the FPGA to the local network with a RJ45 cable.

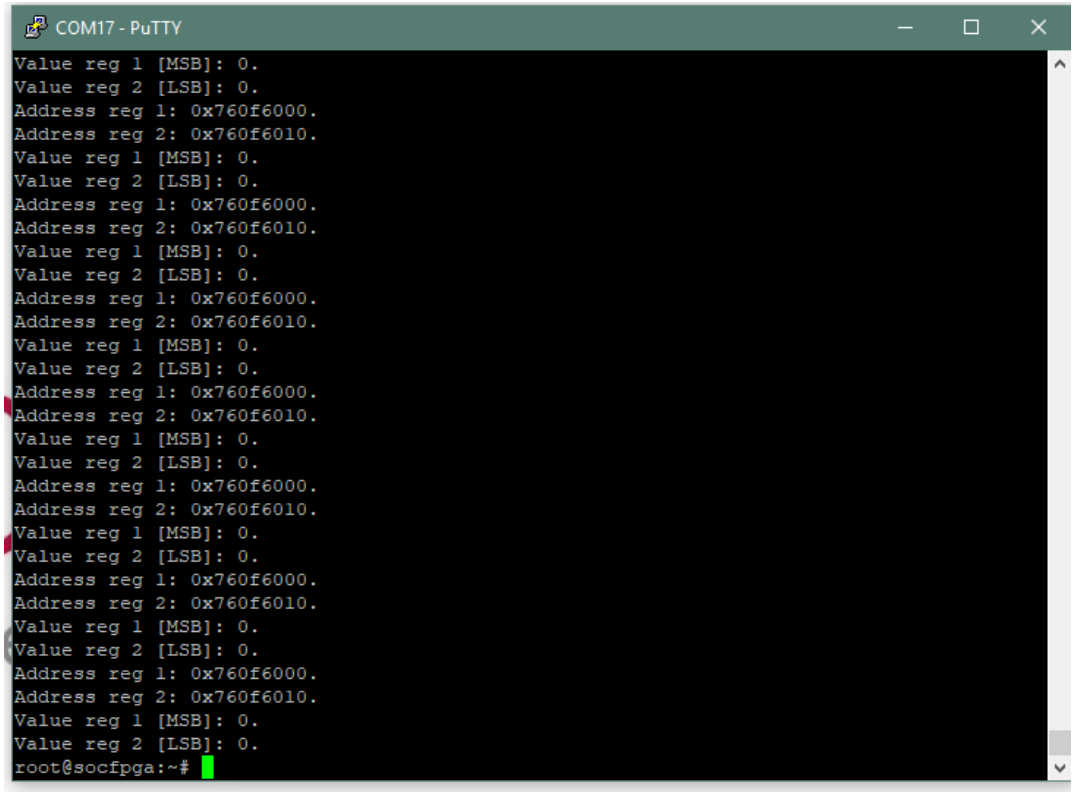
2) Connect to the FPGA with PUTTY (serial/port=COM…/baud=115200) and write the command *socfpga* (login =root) then *udhcpc* to obtain the @IP of the board.

3)With EDS Shell access the project place on the pc and run the command *make* (in the Makefile, modify the Target=Light_sensor).

4)Then run the command *scp* Light_sensor *root @IP:/home/root*



5) On PUTTY run the main.c file to see the result of the light sensor on the screen.



```
Value reg 1 [MSB]: 0.
Value reg 2 [LSB]: 0.
Address reg 1: 0x760f6000.
Address reg 2: 0x760f6010.
Value reg 1 [MSB]: 0.
Value reg 2 [LSB]: 0.
Address reg 1: 0x760f6000.
Address reg 2: 0x760f6010.
Value reg 1 [MSB]: 0.
Value reg 2 [LSB]: 0.
Address reg 1: 0x760f6000.
Address reg 2: 0x760f6010.
Value reg 1 [MSB]: 0.
Value reg 2 [LSB]: 0.
Address reg 1: 0x760f6000.
Address reg 2: 0x760f6010.
Value reg 1 [MSB]: 0.
Value reg 2 [LSB]: 0.
Address reg 1: 0x760f6000.
Address reg 2: 0x760f6010.
Value reg 1 [MSB]: 0.
Value reg 2 [LSB]: 0.
Address reg 1: 0x760f6000.
Address reg 2: 0x760f6010.
Value reg 1 [MSB]: 0.
Value reg 2 [LSB]: 0.
Address reg 1: 0x760f6000.
Address reg 2: 0x760f6010.
Value reg 1 [MSB]: 0.
Value reg 2 [LSB]: 0.
root@socfpga:~#
```

❑Code C:

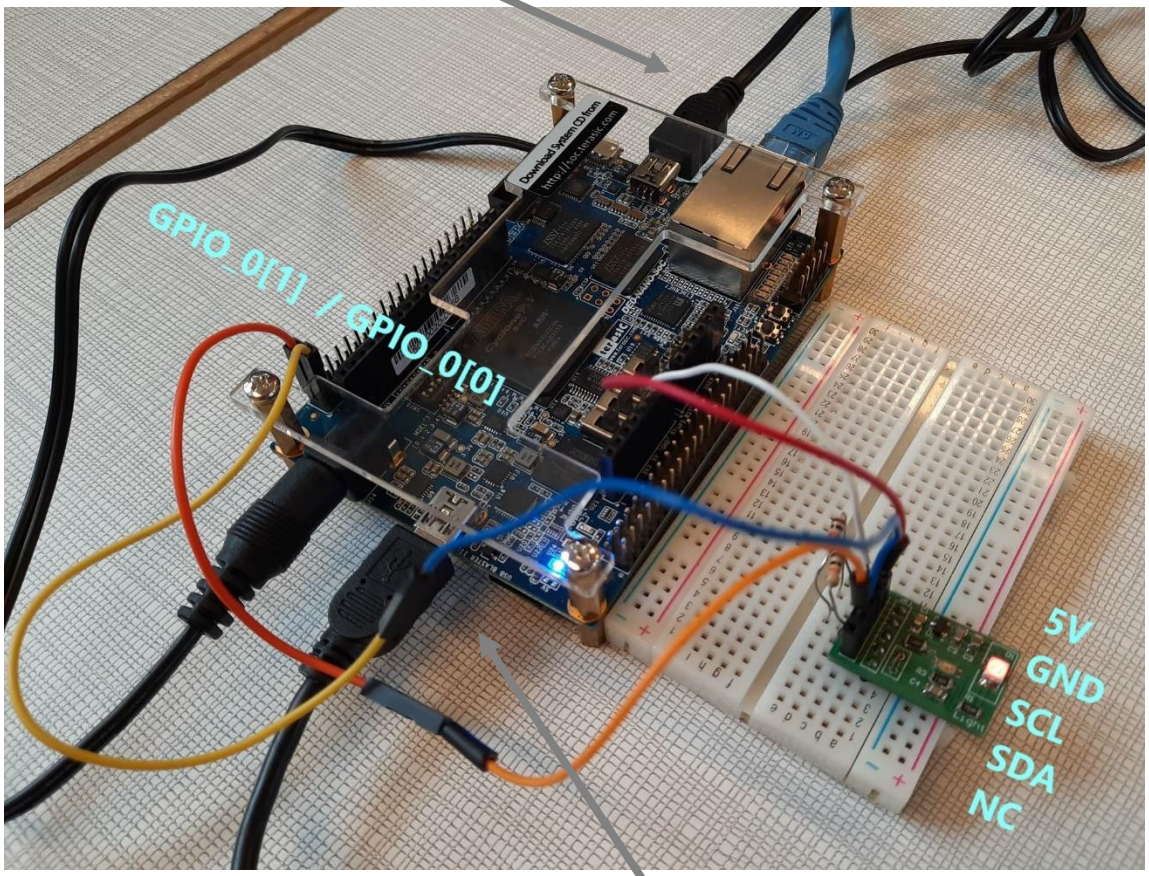
For the software aspect of the sensor reading, a code has been implemented in C (main.c).

It takes the values included in the two registers, the first one corresponding to the most significant byte (MSB) and the second one to the less significant byte (LSB), and concatenate them.

Once the code is run, the brigthness value is displayed on the user's screen using a printf().

Connections made on the board

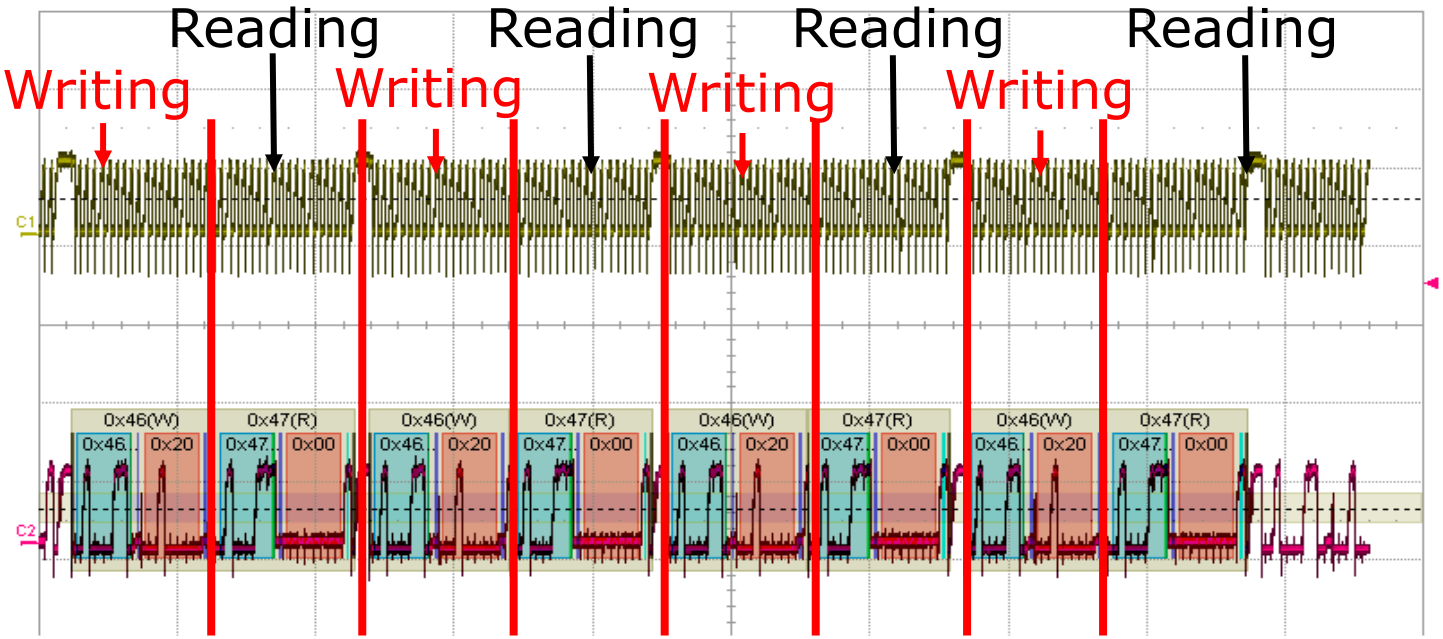
Software part



Hardware part

- 2 pull-up resistors of 10 kΩ between the 5V/SCL and 5V/SDA.

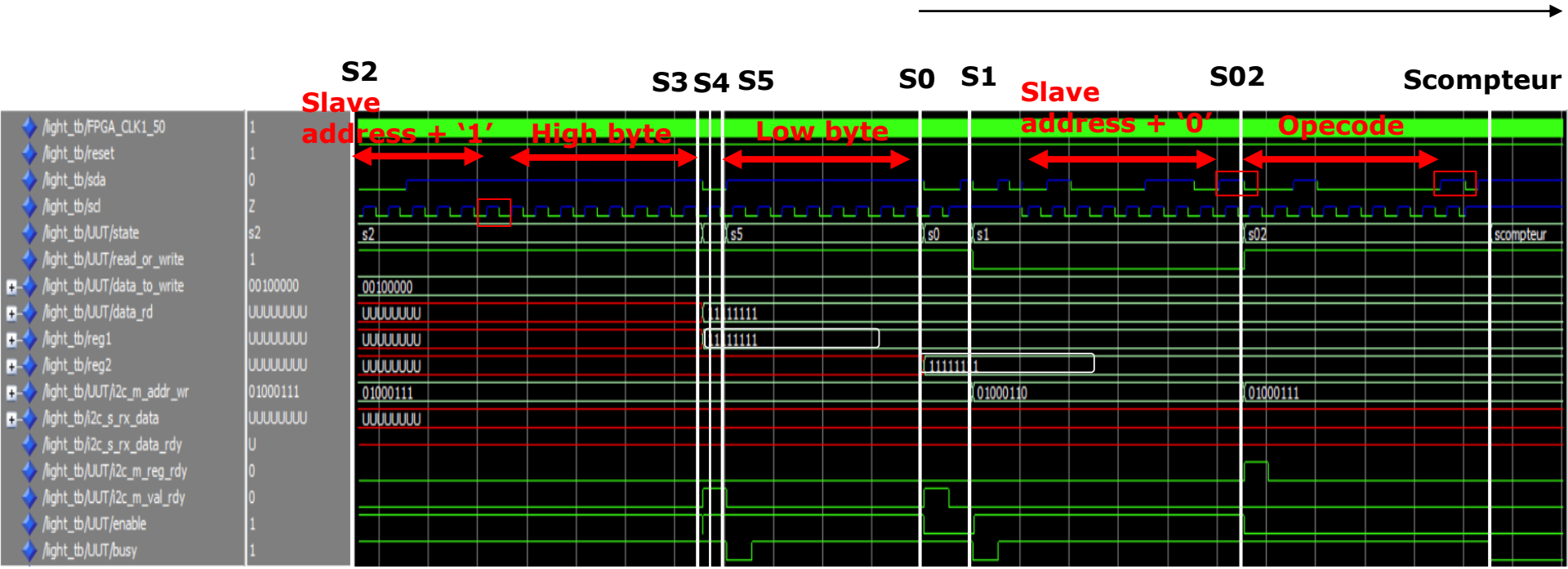
Results with digital oscilloscope



- The write and read addresses are correct(0x46 for R/W = '0' and 0x47 for R/W = '1')
- No data received is observed (0x00) because we go directly to the reading operation

=> need a counter while the sensor is measuring

Test bench of the block light



S0-> write mode, i2c_m_addr_wr = DEVICE & '0' and data_to_write = opecode (resolution)
S1-> read mode, i2c_m_addr_wr <= DEVICE & '1'
S02-> if busy = '0' then state = Scompteur
Scompteur -> wait 120 ms
S2-> put enable to '0' if i2c_m_val_rdy = '1'
S3-> put the read data in register 1
S4-> wait the i2c_m_val_rdy = '0' to pass to state 5
S5-> put the read data in register 2

ACK error

Conclusion

- Expected results
  - No more NACKs
  - View the 2nd register reading on the oscilloscope
  - Have the correct brightness display on PUTTY

⇒ Modify the position of the counter state ?

⇒ Review the steps of the I2C driver ?