Hardware Software platforms : Project presentation

I2C receiver temperature sensor

Denis Pierre

Liégeois Loïc

Rasic Cyril

rasic.cyril@umons.ac.be

Table of contents

- Project presentation
- Reminders (FPGA, I2C drivers)
- Hardware
- Software
- Tests and results

Table of contents

- Project presentation
- Reminders (FPGA, I2C drivers)
- Hardware
- Software
- Tests and results

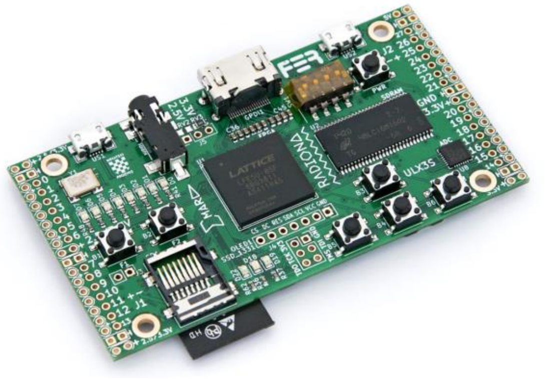
Project presentation

- Steps :
  - Create an I2C driver on a FPGA to receive data from a T sensor = goal of the project
  - Create a test bench to check if what we built is working well
  - Check the results

Table of contents

- Project presentation
- Reminders (FPGA, I2C drivers)
- Hardware
- Software
- Tests and results

FPGA

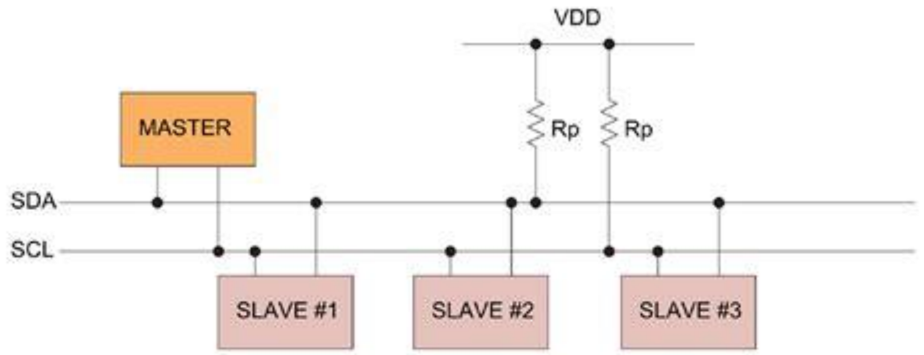


- Field-Programmable Gate Arrays
  - Reconfigurable integrated circuit
  - Enables the creation of customized digital circuits for specific applications.

  - How to use it with Quartus ?
    - VHDL code (behaviour of the circuit)
    - Synthesis & implementation (translate the code in a hardware representation suitable for the FPGA => specific config file .sof)
    - FPGA programming (load the file in the FPGA)

I2C

- Inter-Integrated Circuit
  - Serial communication protocol (allows communication between electronic devices)
  - Consists of 2 lines :
    - SCL (serial clock) : Only the master uses this line to generate the clock which is followed by the slave to send the synchronized data at the speed of the clock
    - SDA (serial data) : Bidirectionnal line carrying the data bit per bit

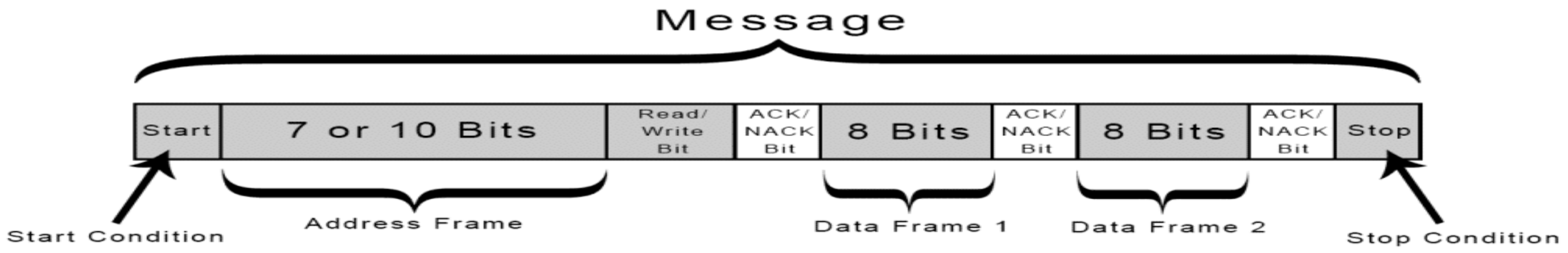


I2C

- Frame :
  - Start condition (high->low while clk is high)
  - Adress of the device (slave in 7 or 10 bits)
  - Control bit (1 to read, 0 to write)
  - Acknowledgment (ack=0, nack=1)
  - Data (slave or master sends the data)
  - Stop condition (low->high while clk is high)

I2C

- Frame :


Message
Start
7 or 10 Bits
Read/ Write Bit
ACK/ NACK Bit
8 Bits
ACK/ NACK Bit
8 Bits
ACK/ NACK Bit
Stop
Start Condition
Address Frame
Data Frame 1
Data Frame 2
Stop Condition

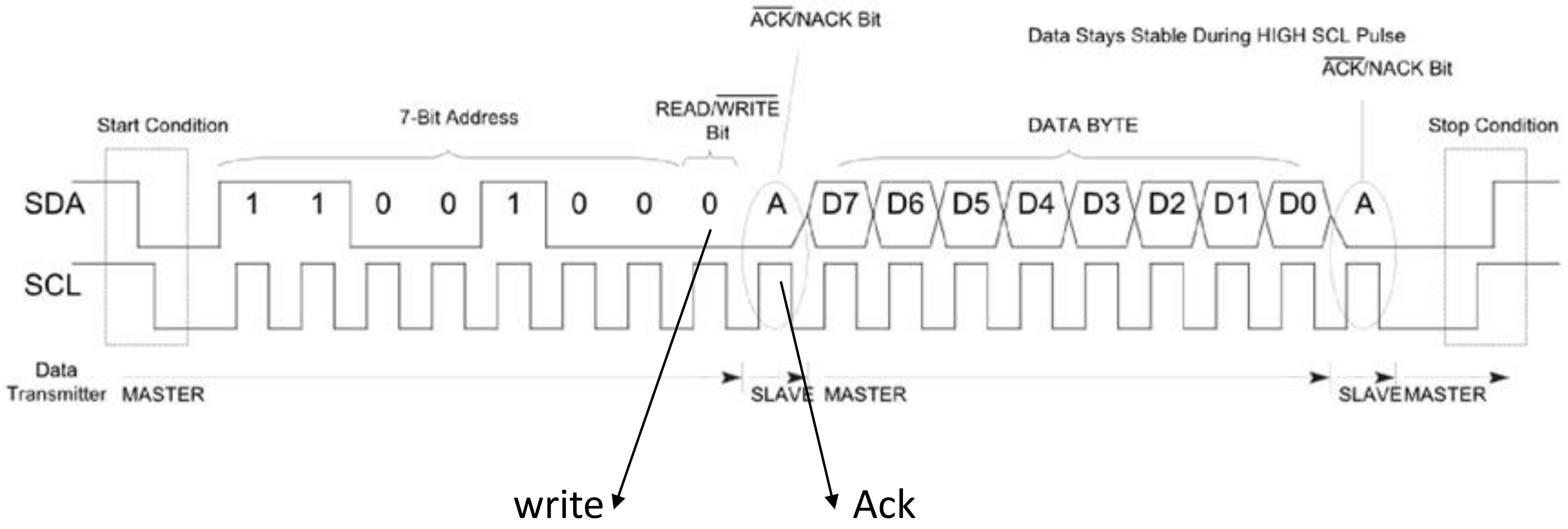

ACK/NACK Bit
Data Stays Stable During HIGH SCL Pulse
ACK/NACK Bit
Start Condition
7-Bit Address
READ/WRITE Bit
DATA BYTE
Stop Condition
SDA
1 1 0 0 1 0 0 0 A D7 D6 D5 D4 D3 D2 D1 D0 A
SCL
Data Transmitter
MASTER
SLAVE
MASTER
SLAVE
MASTER
write
Ack

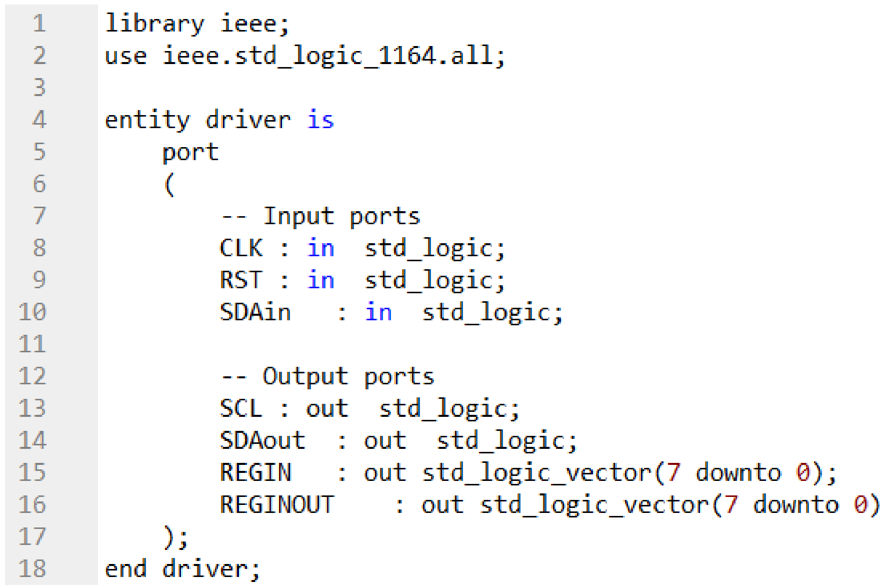
Table of contents

- Project presentation
- Reminders (FPGA, I2C drivers)
- Hardware
- Software
- Tests and results

Hardware

- Built on Quartus
  - Definition of the architecture
  - Description of the behaviour
  - Synthesis of the code
  - Configuration of the FPGA using the .sof file

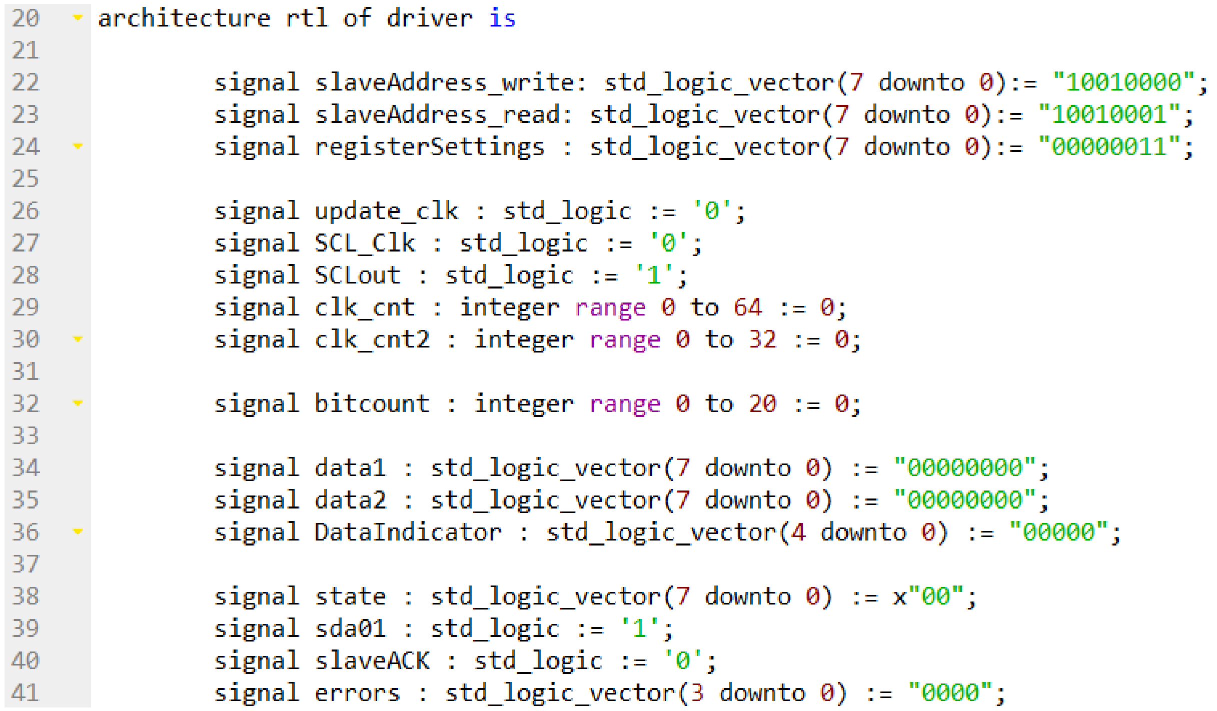
Definition of the entity



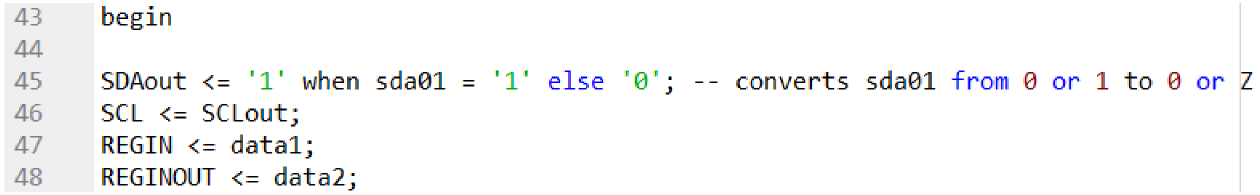
```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity driver is
    port
    (
        -- Input ports
        CLK : in  std_logic;
        RST : in  std_logic;
        SDAin   : in  std_logic;

        -- Output ports
        SCL : out  std_logic;
        SDAout  : out  std_logic;
        REGIN   : out std_logic_vector(7 downto 0);
        REGINOUT    : out std_logic_vector(7 downto 0)
    );
end driver;
```

- All the in/out ports
- No SCLin as the slave can't send on this line

Definition of the architecture



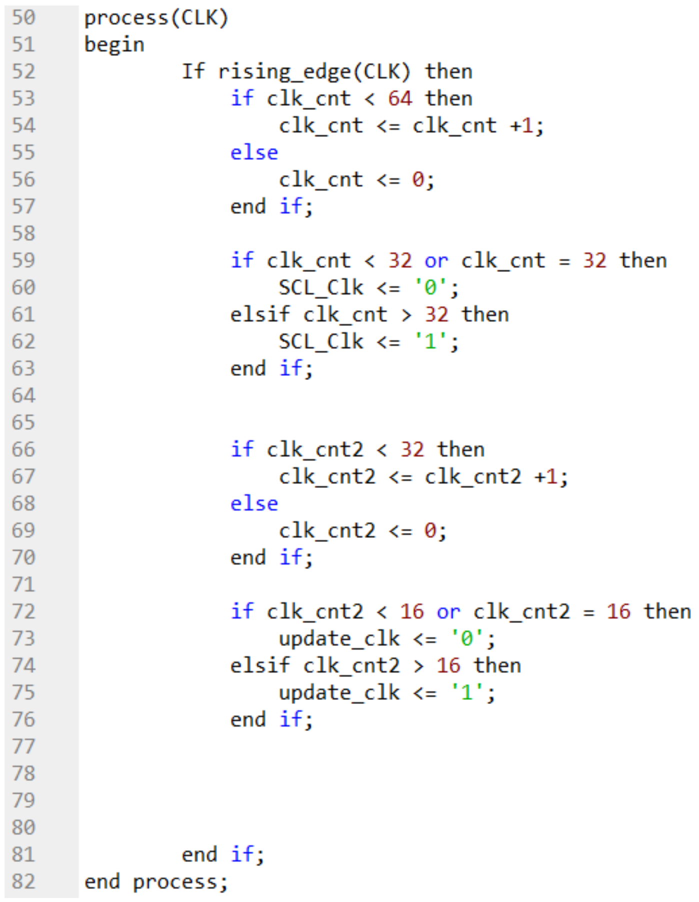
```
architecture rtl of driver is

        signal slaveAddress_write: std_logic_vector(7 downto 0):= "10010000";
        signal slaveAddress_read: std_logic_vector(7 downto 0):= "10010001";
        signal registerSettings : std_logic_vector(7 downto 0):= "00000011";

        signal update_clk : std_logic := '0';
        signal SCL_Clk : std_logic := '0';
        signal SCLout : std_logic := '1';
        signal clk_cnt : integer range 0 to 64 := 0;
        signal clk_cnt2 : integer range 0 to 32 := 0;

        signal bitcount : integer range 0 to 20 := 0;

        signal data1 : std_logic_vector(7 downto 0) := "00000000";
        signal data2 : std_logic_vector(7 downto 0) := "00000000";
        signal DataIndicator : std_logic_vector(4 downto 0) := "00000";

        signal state : std_logic_vector(7 downto 0) := x"00";
        signal sda01 : std_logic := '1';
        signal slaveACK : std_logic := '0';
        signal errors : std_logic_vector(3 downto 0) := "0000";
```



```
begin

SDAout <= '1' when sda01 = '1' else '0'; -- converts sda01 from 0 or 1 to 0 or Z
SCL <= SCLout;
REGIN <= data1;
REGINOUT <= data2;
```

- Description of the logic of the driver
- Definition of the signals
- Assignments
- NB : RTL stands for register transfer level

Creation of the clock



```
process(CLK)
begin
        If rising_edge(CLK) then
            if clk_cnt < 64 then
                clk_cnt <= clk_cnt +1;
            else
                clk_cnt <= 0;
            end if;

            if clk_cnt < 32 or clk_cnt = 32 then
                SCL_Clk <= '0';
            elsif clk_cnt > 32 then
                SCL_Clk <= '1';
            end if;


            if clk_cnt2 < 32 then
                clk_cnt2 <= clk_cnt2 +1;
            else
                clk_cnt2 <= 0;
            end if;

            if clk_cnt2 < 16 or clk_cnt2 = 16 then
                update_clk <= '0';
            elsif clk_cnt2 > 16 then
                update_clk <= '1';
            end if;




        end if;
end process;
```

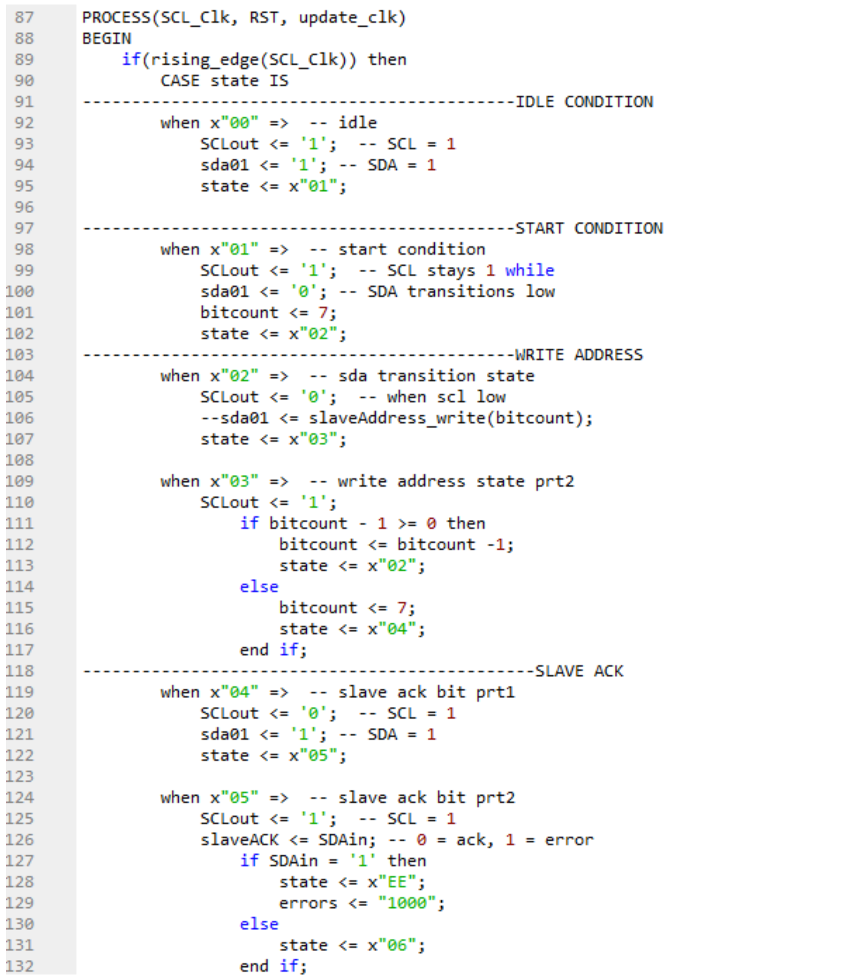
- Creation of a clock used for the communications on the driver
- f_{clk}=400kHz=$f_{process}$/2= $f_{processor}$/2*64
- $f_{processor}$=50MHz

Processes description

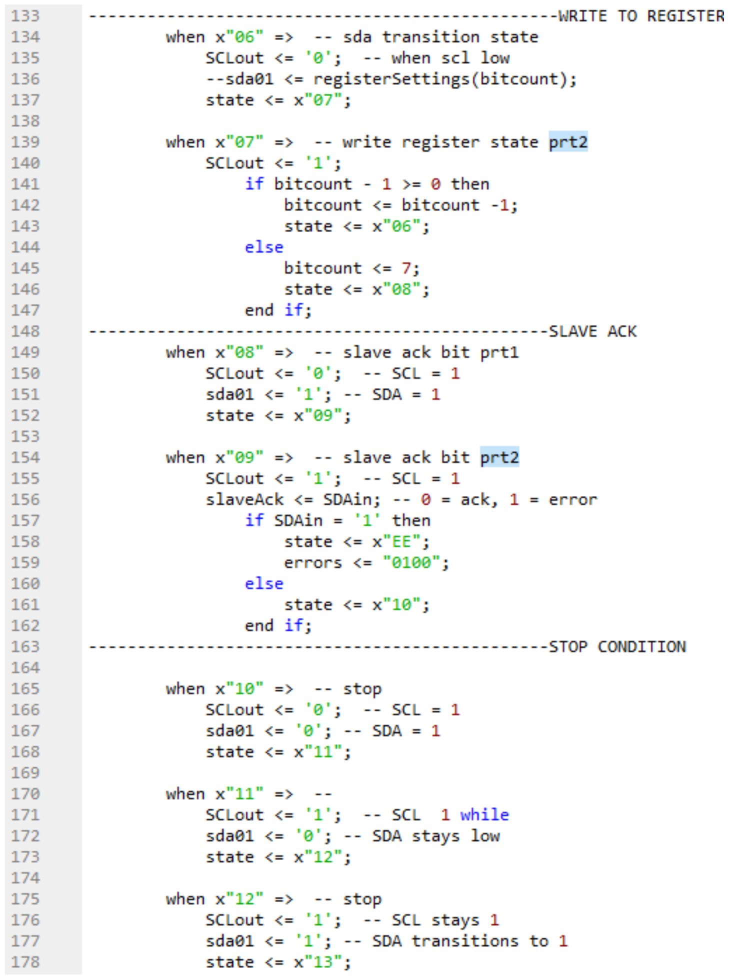
- Two communications on the bus
  1. The master says to the slave which register configuration he will have to send (cf. sensor datasheet)
  2. The master says to the slave to send the data

- When no communication on the bus, state=idle

Communication 1

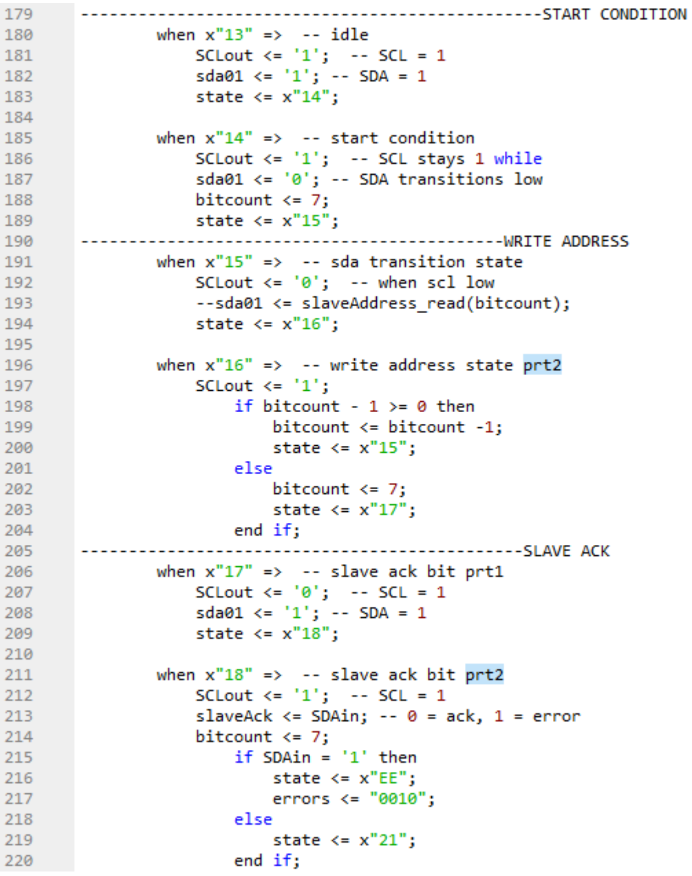


```
PROCESS(SCL_Clk, RST, update_clk)
BEGIN
    if(rising_edge(SCL_Clk)) then
        CASE state IS
--------------------------------------------IDLE CONDITION
        when x"00" =>  -- idle
            SCLout <= '1';  -- SCL = 1
            sda01 <= '1'; -- SDA = 1
            state <= x"01";

--------------------------------------------START CONDITION
        when x"01" =>  -- start condition
            SCLout <= '1';  -- SCL stays 1 while
            sda01 <= '0'; -- SDA transitions low
            bitcount <= 7;
            state <= x"02";
--------------------------------------------WRITE ADDRESS
        when x"02" =>  -- sda transition state
            SCLout <= '0';  -- when scl low
            --sda01 <= slaveAddress_write(bitcount);
            state <= x"03";

        when x"03" =>  -- write address state prt2
            SCLout <= '1';
                if bitcount - 1 >= 0 then
                    bitcount <= bitcount -1;
                    state <= x"02";
                else
                    bitcount <= 7;
                    state <= x"04";
                end if;
----------------------------------------------SLAVE ACK
        when x"04" =>  -- slave ack bit prt1
            SCLout <= '0';  -- SCL = 1
            sda01 <= '1'; -- SDA = 1
            state <= x"05";

        when x"05" =>  -- slave ack bit prt2
            SCLout <= '1';  -- SCL = 1
            slaveACK <= SDAin; -- 0 = ack, 1 = error
                if SDAin = '1' then
                    state <= x"EE";
                    errors <= "1000";
                else
                    state <= x"06";
                end if;
```

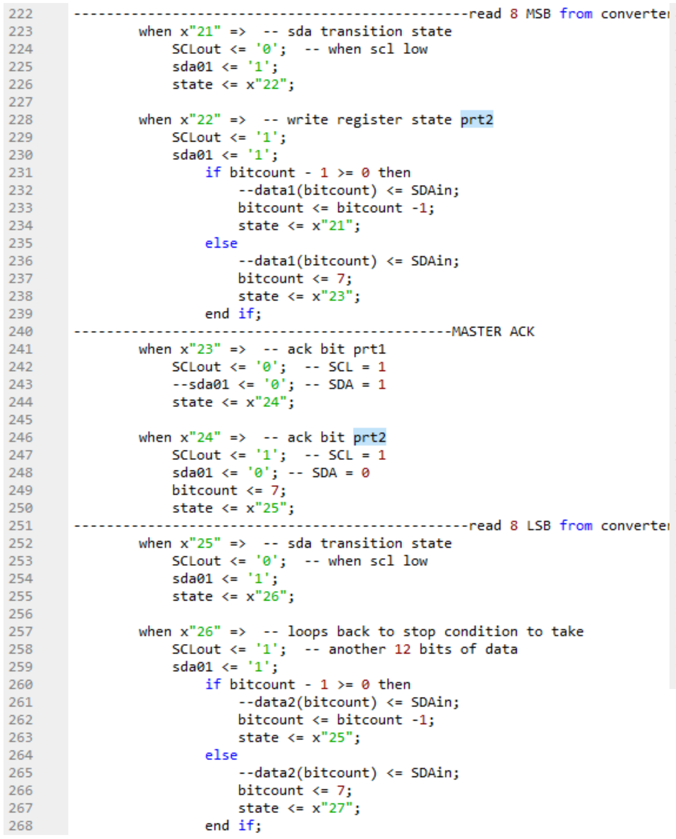


```
-----------------------------------------------WRITE TO REGISTER
        when x"06" =>  -- sda transition state
            SCLout <= '0';  -- when scl low
            --sda01 <= registerSettings(bitcount);
            state <= x"07";

        when x"07" =>  -- write register state prt2
            SCLout <= '1';
                if bitcount - 1 >= 0 then
                    bitcount <= bitcount -1;
                    state <= x"06";
                else
                    bitcount <= 7;
                    state <= x"08";
                end if;
----------------------------------------------SLAVE ACK
        when x"08" =>  -- slave ack bit prt1
            SCLout <= '0';  -- SCL = 1
            sda01 <= '1'; -- SDA = 1
            state <= x"09";

        when x"09" =>  -- slave ack bit prt2
            SCLout <= '1';  -- SCL = 1
            slaveAck <= SDAin; -- 0 = ack, 1 = error
                if SDAin = '1' then
                    state <= x"EE";
                    errors <= "0100";
                else
                    state <= x"10";
                end if;
----------------------------------------------STOP CONDITION

        when x"10" =>  -- stop
            SCLout <= '0';  -- SCL = 1
            sda01 <= '0'; -- SDA = 1
            state <= x"11";

        when x"11" =>  --
            SCLout <= '1';  -- SCL  1 while
            sda01 <= '0'; -- SDA stays low
            state <= x"12";

        when x"12" =>  -- stop
            SCLout <= '1';  -- SCL stays 1
            sda01 <= '1'; -- SDA transitions to 1
            state <= x"13";
```

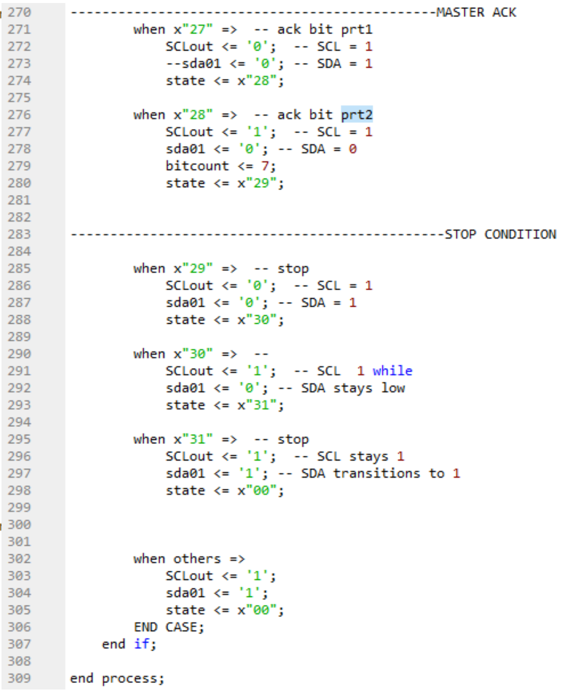
Communication 1



```vhdl
-------------------------------------------------START CONDITION
        when x"13" =>  -- idle
            SCLout <= '1';  -- SCL = 1
            sda01 <= '1'; -- SDA = 1
            state <= x"14";

        when x"14" =>  -- start condition
            SCLout <= '1';  -- SCL stays 1 while
            sda01 <= '0'; -- SDA transitions low
            bitcount <= 7;
            state <= x"15";
---------------------------------------------WRITE ADDRESS
        when x"15" =>  -- sda transition state
            SCLout <= '0';  -- when scl low
            --sda01 <= slaveAddress_read(bitcount);
            state <= x"16";

        when x"16" =>  -- write address state prt2
            SCLout <= '1';
                if bitcount - 1 >= 0 then
                    bitcount <= bitcount -1;
                    state <= x"15";
                else
                    bitcount <= 7;
                    state <= x"17";
                end if;
-----------------------------------------------SLAVE ACK
        when x"17" =>  -- slave ack bit prt1
            SCLout <= '0';  -- SCL = 1
            sda01 <= '1'; -- SDA = 1
            state <= x"18";

        when x"18" =>  -- slave ack bit prt2
            SCLout <= '1';  -- SCL = 1
            slaveAck <= SDAin; -- 0 = ack, 1 = error
            bitcount <= 7;
                if SDAin = '1' then
                    state <= x"EE";
                    errors <= "0010";
                else
                    state <= x"21";
                end if;
```

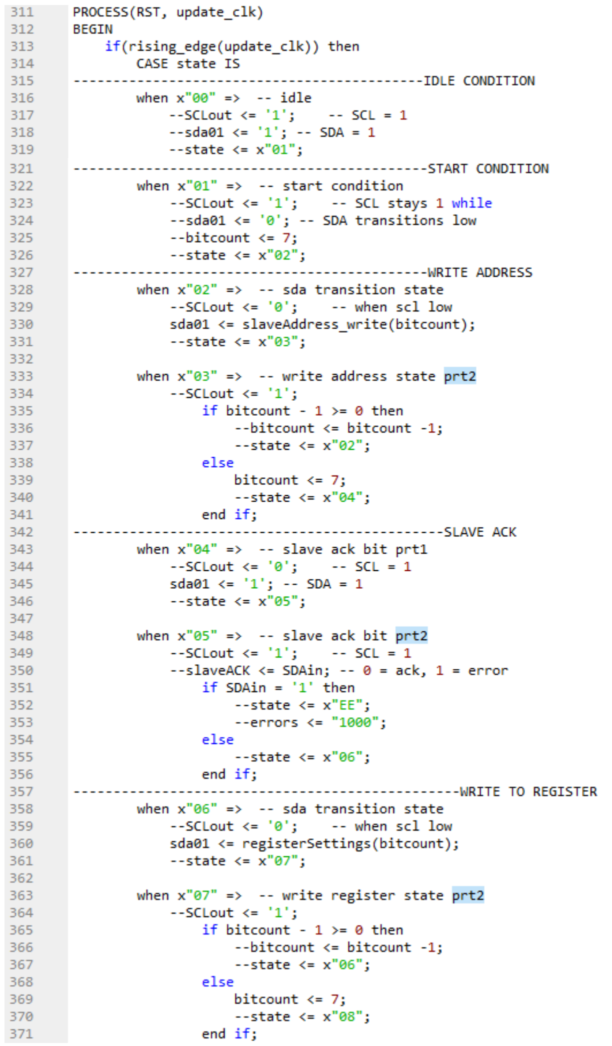


```vhdl
--------------------------------------------------read 8 MSB from converte
        when x"21" =>  -- sda transition state
            SCLout <= '0';  -- when scl low
            sda01 <= '1';
            state <= x"22";

        when x"22" =>  -- write register state prt2
            SCLout <= '1';
            sda01 <= '1';
                if bitcount - 1 >= 0 then
                    --data1(bitcount) <= SDAin;
                    bitcount <= bitcount -1;
                    state <= x"21";
                else
                    --data1(bitcount) <= SDAin;
                    bitcount <= 7;
                    state <= x"23";
                end if;
----------------------------------------------MASTER ACK
        when x"23" =>  -- ack bit prt1
            SCLout <= '0';  -- SCL = 1
            --sda01 <= '0'; -- SDA = 1
            state <= x"24";

        when x"24" =>  -- ack bit prt2
            SCLout <= '1';  -- SCL = 1
            sda01 <= '0'; -- SDA = 0
            bitcount <= 7;
            state <= x"25";
--------------------------------------------------read 8 LSB from converte
        when x"25" =>  -- sda transition state
            SCLout <= '0';  -- when scl low
            sda01 <= '1';
            state <= x"26";

        when x"26" =>  -- loops back to stop condition to take
            SCLout <= '1';  -- another 12 bits of data
            sda01 <= '1';
                if bitcount - 1 >= 0 then
                    --data2(bitcount) <= SDAin;
                    bitcount <= bitcount -1;
                    state <= x"25";
                else
                    --data2(bitcount) <= SDAin;
                    bitcount <= 7;
                    state <= x"27";
                end if;
```

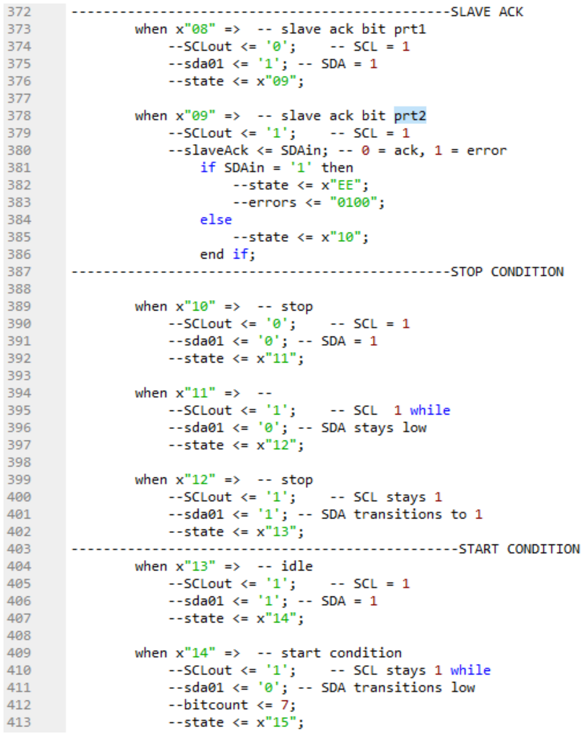


```vhdl
---------------------------------------------MASTER ACK
        when x"27" =>  -- ack bit prt1
            SCLout <= '0';  -- SCL = 1
            --sda01 <= '0'; -- SDA = 1
            state <= x"28";

        when x"28" =>  -- ack bit prt2
            SCLout <= '1';  -- SCL = 1
            sda01 <= '0'; -- SDA = 0
            bitcount <= 7;
            state <= x"29";


---------------------------------------------STOP CONDITION

        when x"29" =>  -- stop
            SCLout <= '0';  -- SCL = 1
            sda01 <= '0'; -- SDA = 1
            state <= x"30";

        when x"30" =>  --
            SCLout <= '1';  -- SCL  1 while
            sda01 <= '0'; -- SDA stays low
            state <= x"31";

        when x"31" =>  -- stop
            SCLout <= '1';  -- SCL stays 1
            sda01 <= '1'; -- SDA transitions to 1
            state <= x"00";



        when others =>
            SCLout <= '1';
            sda01 <= '1';
            state <= x"00";
        END CASE;
    end if;

end process;
```

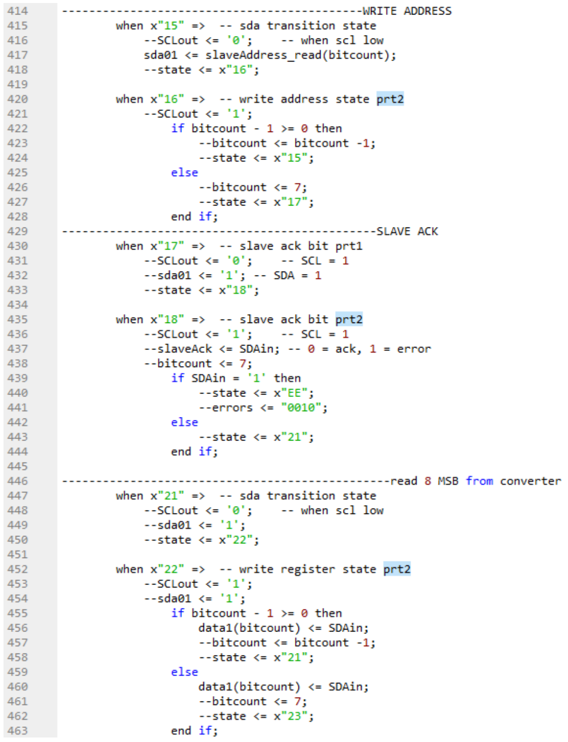
Communication 2



```
PROCESS(RST, update_clk)
BEGIN
    if(rising_edge(update_clk)) then
        CASE state IS
--------------------------------------------IDLE CONDITION
        when x"00" =>  -- idle
            --SCLout <= '1';     -- SCL = 1
            --sda01 <= '1'; -- SDA = 1
            --state <= x"01";

--------------------------------------------START CONDITION
        when x"01" =>  -- start condition
            --SCLout <= '1';     -- SCL stays 1 while
            --sda01 <= '0'; -- SDA transitions low
            --bitcount <= 7;
            --state <= x"02";
--------------------------------------------WRITE ADDRESS
        when x"02" =>  -- sda transition state
            --SCLout <= '0';     -- when scl low
            sda01 <= slaveAddress_write(bitcount);
            --state <= x"03";

        when x"03" =>  -- write address state prt2
            --SCLout <= '1';
                if bitcount - 1 >= 0 then
                    --bitcount <= bitcount -1;
                    --state <= x"02";
                else
                    bitcount <= 7;
                    --state <= x"04";
                end if;
----------------------------------------------SLAVE ACK
        when x"04" =>  -- slave ack bit prt1
            --SCLout <= '0';     -- SCL = 1
            sda01 <= '1'; -- SDA = 1
            --state <= x"05";

        when x"05" =>  -- slave ack bit prt2
            --SCLout <= '1';     -- SCL = 1
            --slaveACK <= SDAin; -- 0 = ack, 1 = error
                if SDAin = '1' then
                    --state <= x"EE";
                    --errors <= "1000";
                else
                    --state <= x"06";
                end if;
----------------------------------------------WRITE TO REGISTER
        when x"06" =>  -- sda transition state
            --SCLout <= '0';     -- when scl low
            sda01 <= registerSettings(bitcount);
            --state <= x"07";

        when x"07" =>  -- write register state prt2
            --SCLout <= '1';
                if bitcount - 1 >= 0 then
                    --bitcount <= bitcount -1;
                    --state <= x"06";
                else
                    bitcount <= 7;
                    --state <= x"08";
                end if;
```

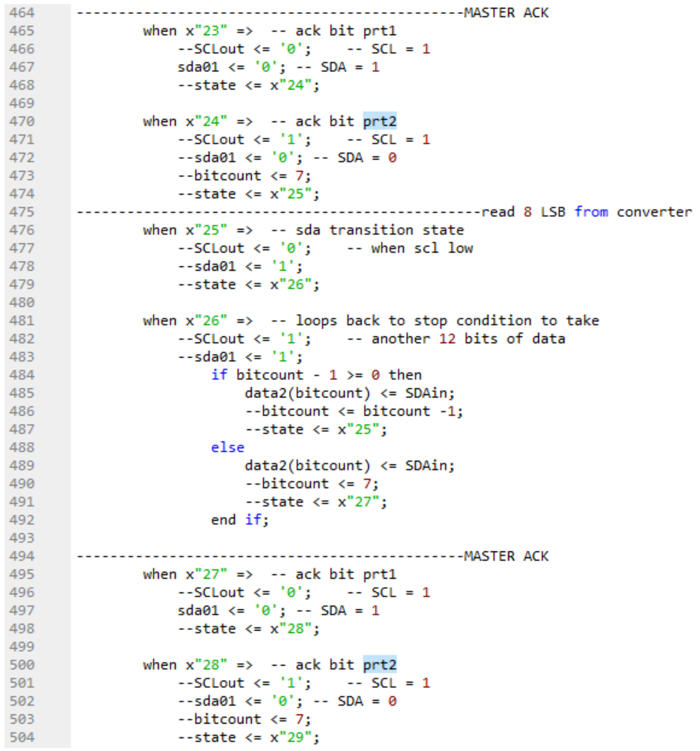


```
----------------------------------------------SLAVE ACK
        when x"08" =>  -- slave ack bit prt1
            --SCLout <= '0';     -- SCL = 1
            --sda01 <= '1'; -- SDA = 1
            --state <= x"09";

        when x"09" =>  -- slave ack bit prt2
            --SCLout <= '1';     -- SCL = 1
            --slaveAck <= SDAin; -- 0 = ack, 1 = error
                if SDAin = '1' then
                    --state <= x"EE";
                    --errors <= "0100";
                else
                    --state <= x"10";
                end if;
----------------------------------------------STOP CONDITION

        when x"10" =>  -- stop
            --SCLout <= '0';     -- SCL = 1
            --sda01 <= '0'; -- SDA = 1
            --state <= x"11";

        when x"11" =>  --
            --SCLout <= '1';     -- SCL  1 while
            --sda01 <= '0'; -- SDA stays low
            --state <= x"12";

        when x"12" =>  -- stop
            --SCLout <= '1';     -- SCL stays 1
            --sda01 <= '1'; -- SDA transitions to 1
            --state <= x"13";
-----------------------------------------------START CONDITION
        when x"13" =>  -- idle
            --SCLout <= '1';     -- SCL = 1
            --sda01 <= '1'; -- SDA = 1
            --state <= x"14";

        when x"14" =>  -- start condition
            --SCLout <= '1';     -- SCL stays 1 while
            --sda01 <= '0'; -- SDA transitions low
            --bitcount <= 7;
            --state <= x"15";
```

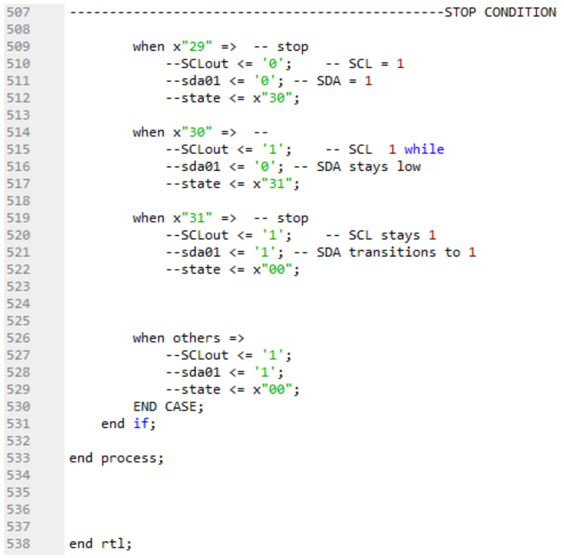


```
--------------------------------------------WRITE ADDRESS
        when x"15" =>  -- sda transition state
            --SCLout <= '0';     -- when scl low
            sda01 <= slaveAddress_read(bitcount);
            --state <= x"16";

        when x"16" =>  -- write address state prt2
            --SCLout <= '1';
                if bitcount - 1 >= 0 then
                    --bitcount <= bitcount -1;
                    --state <= x"15";
                else
                    --bitcount <= 7;
                    --state <= x"17";
                end if;
----------------------------------------------SLAVE ACK
        when x"17" =>  -- slave ack bit prt1
            --SCLout <= '0';     -- SCL = 1
            --sda01 <= '1'; -- SDA = 1
            --state <= x"18";

        when x"18" =>  -- slave ack bit prt2
            --SCLout <= '1';     -- SCL = 1
            --slaveAck <= SDAin; -- 0 = ack, 1 = error
            --bitcount <= 7;
                if SDAin = '1' then
                    --state <= x"EE";
                    --errors <= "0010";
                else
                    --state <= x"21";
                end if;

------------------------------------------------read 8 MSB from converter
        when x"21" =>  -- sda transition state
            --SCLout <= '0';     -- when scl low
            --sda01 <= '1';
            --state <= x"22";

        when x"22" =>  -- write register state prt2
            --SCLout <= '1';
            --sda01 <= '1';
                if bitcount - 1 >= 0 then
                    data1(bitcount) <= SDAin;
                    --bitcount <= bitcount -1;
                    --state <= x"21";
                else
                    data1(bitcount) <= SDAin;
                    --bitcount <= 7;
                    --state <= x"23";
                end if;
```

Communication 2

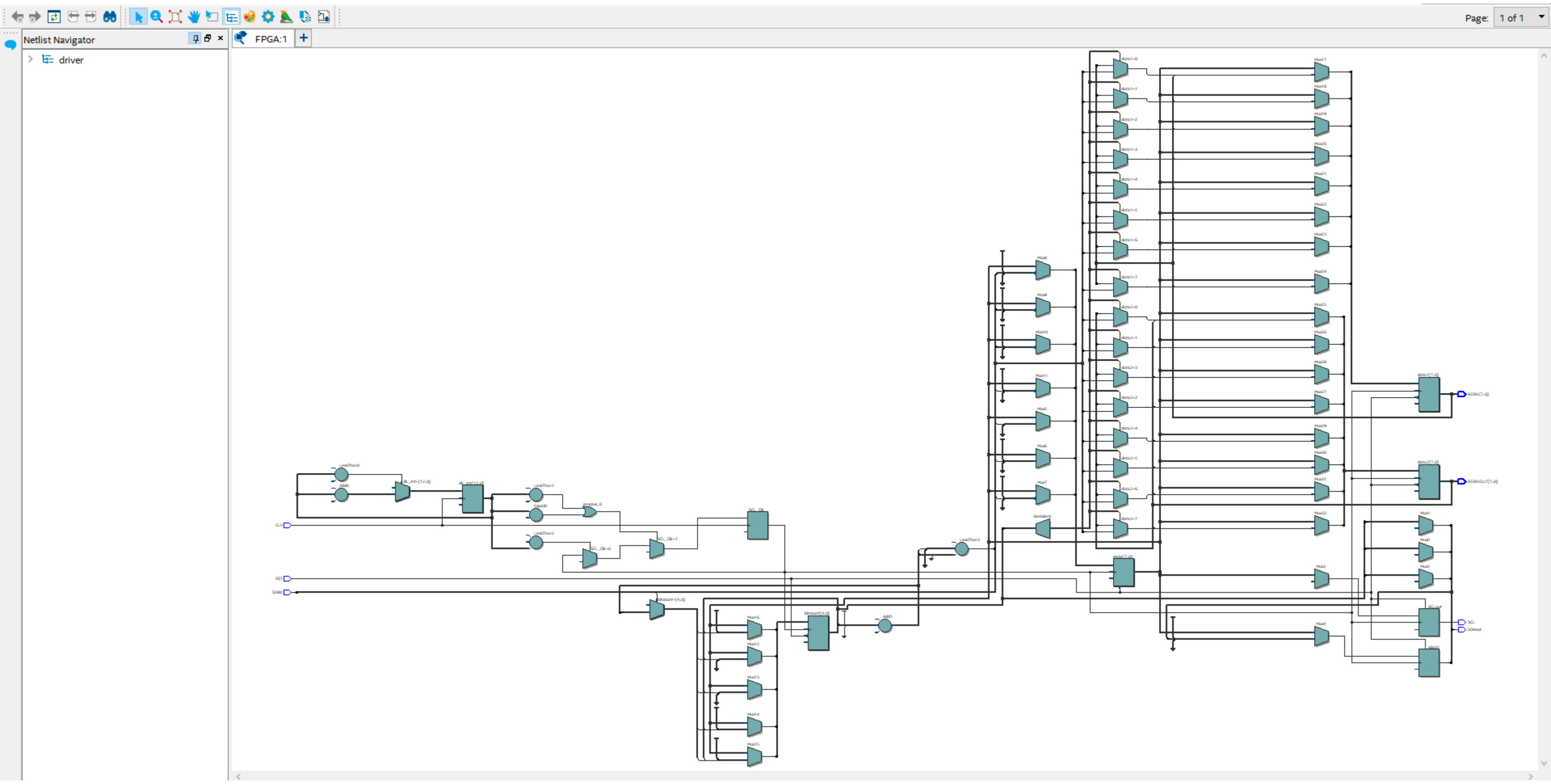


```vhdl
---------------------------------------------MASTER ACK
        when x"23" =>  -- ack bit prt1
            --SCLout <= '0';     -- SCL = 1
            sda01 <= '0'; -- SDA = 1
            --state <= x"24";

        when x"24" =>  -- ack bit prt2
            --SCLout <= '1';     -- SCL = 1
            --sda01 <= '0'; -- SDA = 0
            --bitcount <= 7;
            --state <= x"25";
-----------------------------------------------read 8 LSB from converter
        when x"25" =>  -- sda transition state
            --SCLout <= '0';     -- when scl low
            --sda01 <= '1';
            --state <= x"26";

        when x"26" =>  -- loops back to stop condition to take
            --SCLout <= '1';     -- another 12 bits of data
            --sda01 <= '1';
                if bitcount - 1 >= 0 then
                    data2(bitcount) <= SDAin;
                    --bitcount <= bitcount -1;
                    --state <= x"25";
                else
                    data2(bitcount) <= SDAin;
                    --bitcount <= 7;
                    --state <= x"27";
                end if;

---------------------------------------------MASTER ACK
        when x"27" =>  -- ack bit prt1
            --SCLout <= '0';     -- SCL = 1
            sda01 <= '0'; -- SDA = 1
            --state <= x"28";

        when x"28" =>  -- ack bit prt2
            --SCLout <= '1';     -- SCL = 1
            --sda01 <= '0'; -- SDA = 0
            --bitcount <= 7;
            --state <= x"29";
```



```vhdl
-----------------------------------------------STOP CONDITION


        when x"29" =>  -- stop
            --SCLout <= '0';     -- SCL = 1
            --sda01 <= '0'; -- SDA = 1
            --state <= x"30";

        when x"30" =>  --
            --SCLout <= '1';     -- SCL  1 while
            --sda01 <= '0'; -- SDA stays low
            --state <= x"31";

        when x"31" =>  -- stop
            --SCLout <= '1';     -- SCL stays 1
            --sda01 <= '1'; -- SDA transitions to 1
            --state <= x"00";



        when others =>
            --SCLout <= '1';
            --sda01 <= '1';
            --state <= x"00";
        END CASE;
    end if;

end process;



end rtl;
```

Synthesis



Synthesis

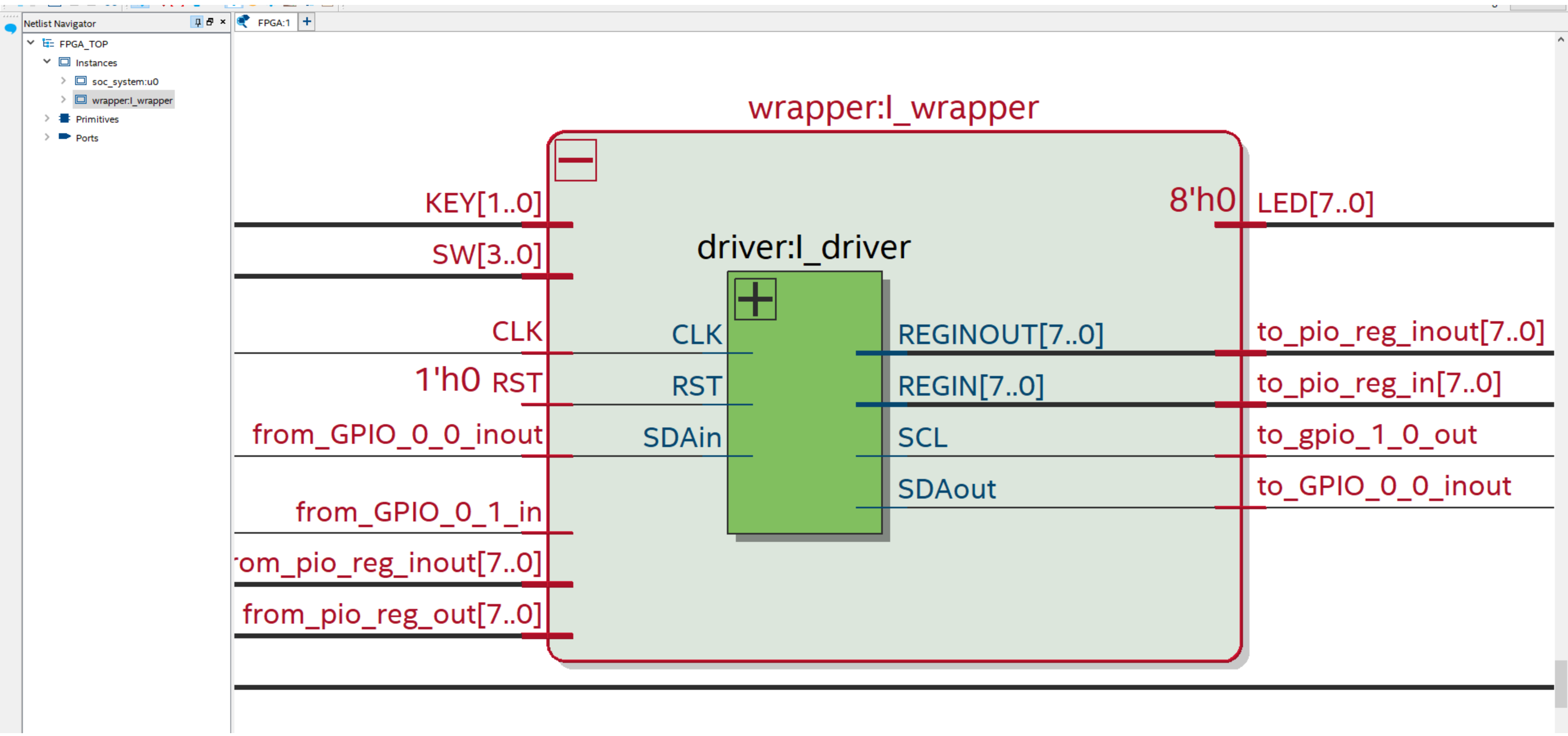

Netlist Navigator
FPGA_TOP
Instances
soc_system:u0
wrapper:l_wrapper
Primitives
Ports
FPGA:1
wrapper:l_wrapper
KEY[1..0]
SW[3..0]
CLK
1'h0 RST
from_GPIO_0_0_inout
from_GPIO_0_1_in
rom_pio_reg_inout[7..0]
from_pio_reg_out[7..0]
driver:l_driver
CLK
RST
SDAin
REGINOUT[7..0]
REGIN[7..0]
SCL
SDAout
8'h0
LED[7..0]
to_pio_reg_inout[7..0]
to_pio_reg_in[7..0]
to_gpio_1_0_out
to_GPIO_0_0_inout

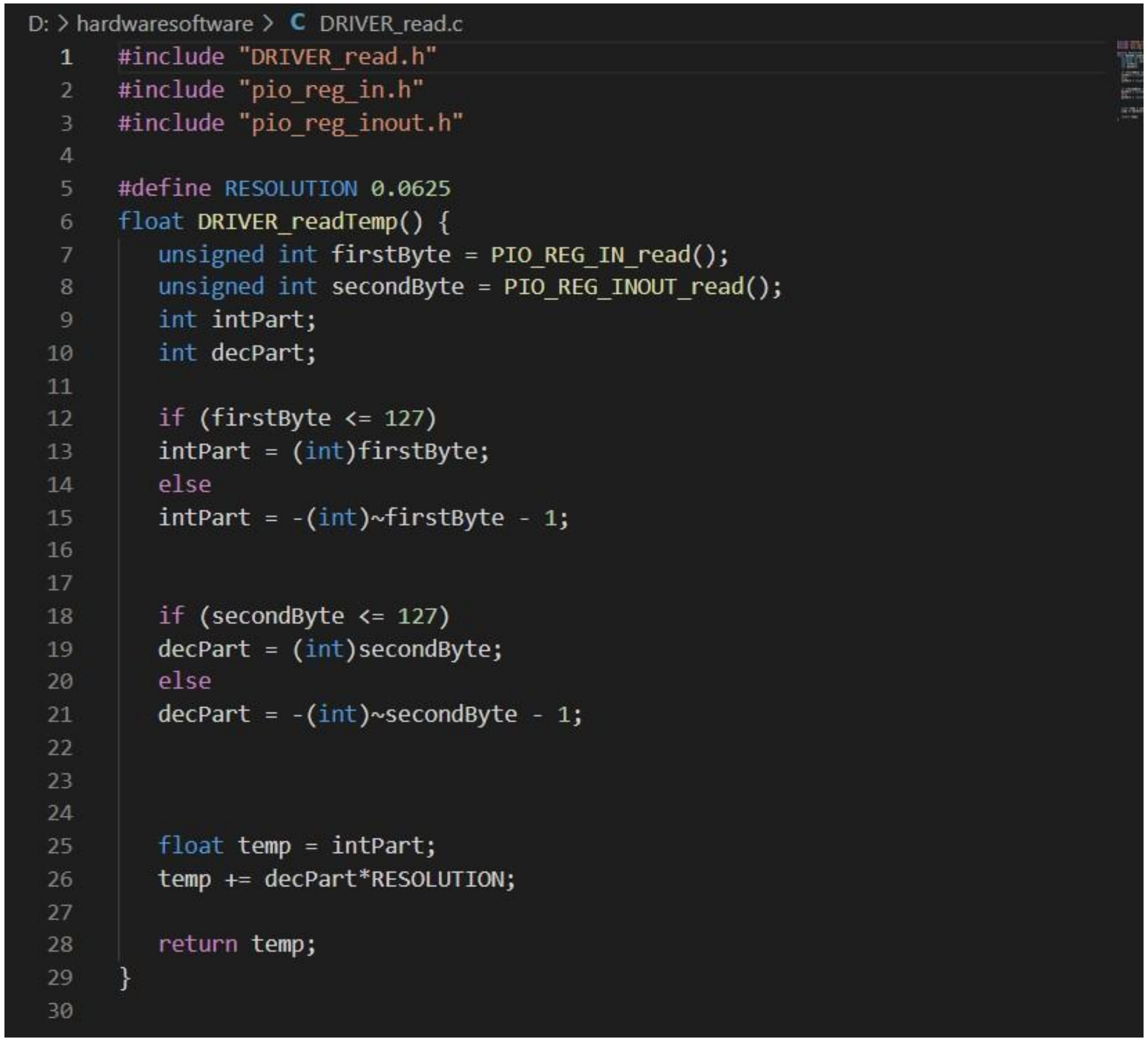
Table of contents

- Project presentation
- Reminders (FPGA, I2C drivers)
- Hardware
- Software
- Tests and results

Software

- Fetch the bits in the registers (IP adress in putty)
- Convert the unsigned int bits
  - The bits are coded in 2's complement (-127 to 127) but stored in a variable of type int unsigned (0 to 255). Due to that, we have to convert the integer and the decimal part in a int type

Software



D: > hardwaresoftware > C DRIVER_read.c

```c
#include "DRIVER_read.h"
#include "pio_reg_in.h"
#include "pio_reg_inout.h"

#define RESOLUTION 0.0625
float DRIVER_readTemp() {
    unsigned int firstByte = PIO_REG_IN_read();
    unsigned int secondByte = PIO_REG_INOUT_read();
    int intPart;
    int decPart;

    if (firstByte <= 127)
    intPart = (int)firstByte;
    else
    intPart = -(int)~firstByte - 1;


    if (secondByte <= 127)
    decPart = (int)secondByte;
    else
    decPart = -(int)~secondByte - 1;



    float temp = intPart;
    temp += decPart*RESOLUTION;

    return temp;
}

```

Table of contents

- Project presentation
- Reminders (FPGA, I2C drivers)
- Hardware
- Software
- Tests and results

testbench

- Creation of a testbench to check if working properly
  - Simulation of a communication on the bus
  - Check if the driver is reacting as it should

Testbench

```
DUT: driver
    port map (
        RST      => sRST,
        CLK      => sCLK,
        SDAin    => sSDA,

        SDAout  => dSDA,
        SCL      => sSCL,
        REGIN => dREGIN,
        REGINOUT  => dREGINOUT
    );

P_sCLK: process
begin -- 50MHz clock
    sCLK <= not sCLK;
    wait for PERIOD/2;
end process;

PP: process
begin -- 50MHz clock
    sSDA <= '1';

    --start
    wait until dSDA = '0';

    --address
    wait until sSCL = '0';
    wait until sSCL = '1';
    aa(7) <= dSDA;
    wait until sSCL = '1';
    aa(6) <= dSDA;
    wait until sSCL = '1';
    aa(5) <= dSDA;
    wait until sSCL = '1';
    aa(4) <= dSDA;
    wait until sSCL = '1';
    aa(3) <= dSDA;
    wait until sSCL = '1';
    aa(2) <= dSDA;
    wait until sSCL = '1';
    aa(1) <= dSDA;
    wait until sSCL = '1';
    aa(0) <= dSDA;

    --ack
    wait until sSCL = '0';
    sSDA <= '0';
    wait until sSCL = '0';

    sSDA <= '1';
```

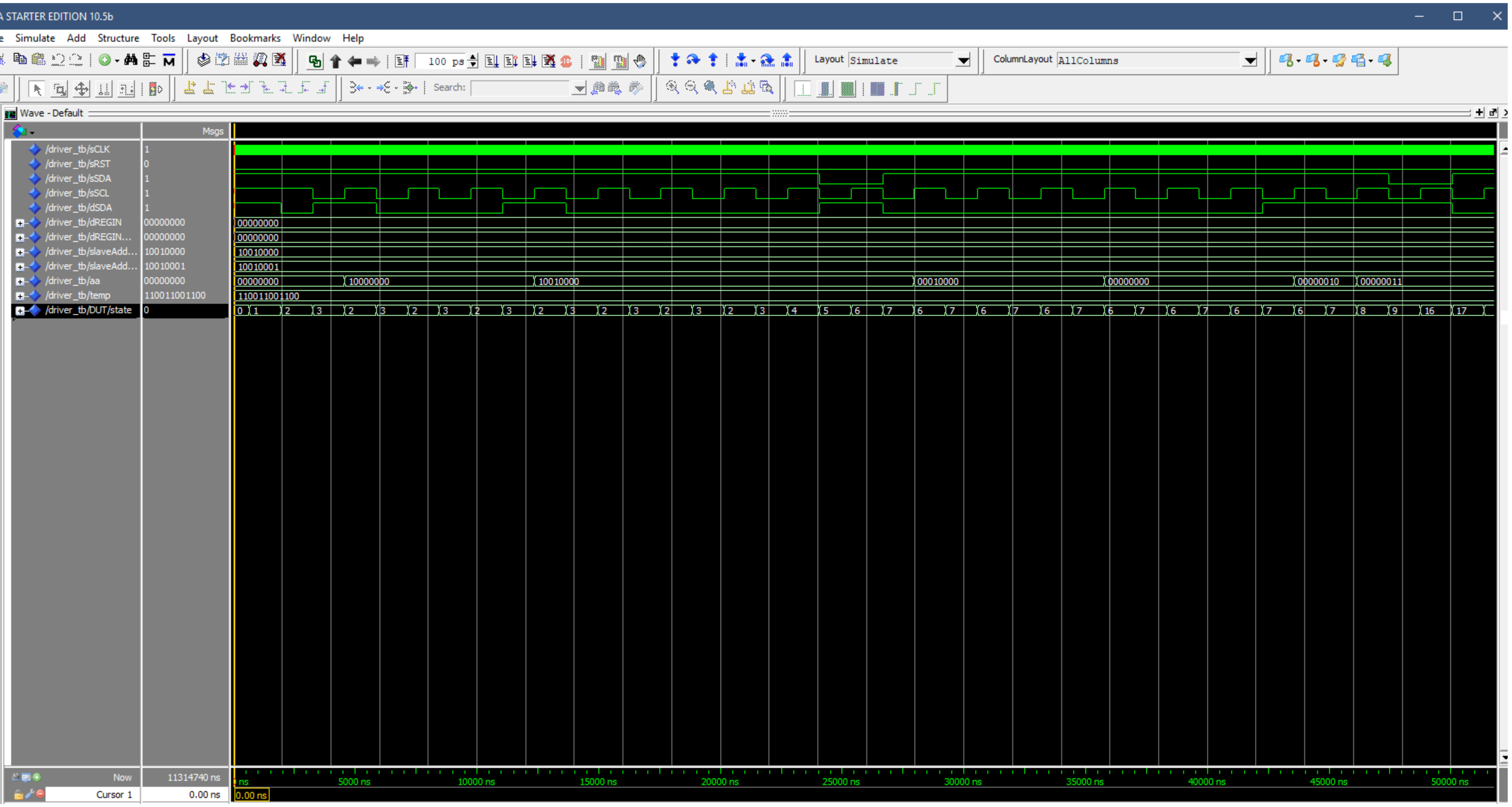
```
    --register
    wait until sSCL = '1';
    aa(7) <= dSDA;
    wait until sSCL = '1';
    aa(6) <= dSDA;
    wait until sSCL = '1';
    aa(5) <= dSDA;
    wait until sSCL = '1';
    aa(4) <= dSDA;
    wait until sSCL = '1';
    aa(3) <= dSDA;
    wait until sSCL = '1';
    aa(2) <= dSDA;
    wait until sSCL = '1';
    aa(1) <= dSDA;
    wait until sSCL = '1';
    aa(0) <= dSDA;

    --ack
    wait until sSCL = '0';
    sSDA <= '0';
    wait until sSCL = '0';
    sSDA <= '1';

    --stop
    wait until sSCL = '1';
    wait until dSDA = '1';


    --start
    wait until dSDA = '0';

    --address
    wait until sSCL = '0';
    wait until sSCL = '1';
    aa(7) <= dSDA;
    wait until sSCL = '1';
    aa(6) <= dSDA;
    wait until sSCL = '1';
    aa(5) <= dSDA;
    wait until sSCL = '1';
    aa(4) <= dSDA;
    wait until sSCL = '1';
    aa(3) <= dSDA;
    wait until sSCL = '1';
    aa(2) <= dSDA;
    wait until sSCL = '1';
    aa(1) <= dSDA;
    wait until sSCL = '1';
    aa(0) <= dSDA;
```

```
    --ack
    wait until sSCL = '0';
    sSDA <= '0';

    --write
    wait until sSCL = '0';
    sSDA <= temp(11);
    wait until sSCL = '0';
    sSDA <= temp(10);
    wait until sSCL = '0';
    sSDA <= temp(9);
    wait until sSCL = '0';
    sSDA <= temp(8);
    wait until sSCL = '0';
    sSDA <= temp(7);
    wait until sSCL = '0';
    sSDA <= temp(6);
    wait until sSCL = '0';
    sSDA <= temp(5);
    wait until sSCL = '0';
    sSDA <= temp(4);

    wait until sSCL = '1';
    wait until sSCL = '0';
    sSDA <= '0';

    --write 2
    wait until sSCL = '0';
    sSDA <= temp(3);
    wait until sSCL = '0';
    sSDA <= temp(2);
    wait until sSCL = '0';
    sSDA <= temp(1);
    wait until sSCL = '0';
    sSDA <= temp(0);

    wait until sSCL = '0';
    sSDA <= '0';
    wait until sSCL = '0';
    sSDA <= '0';
    wait until sSCL = '0';
    sSDA <= '0';
    wait until sSCL = '0';
    sSDA <= '0';


    wait until sSCL = '0';
    sSDA <= '1';
```

```
    --stop
    wait until sSCL = '0';
    wait until dSDA = '1';
```

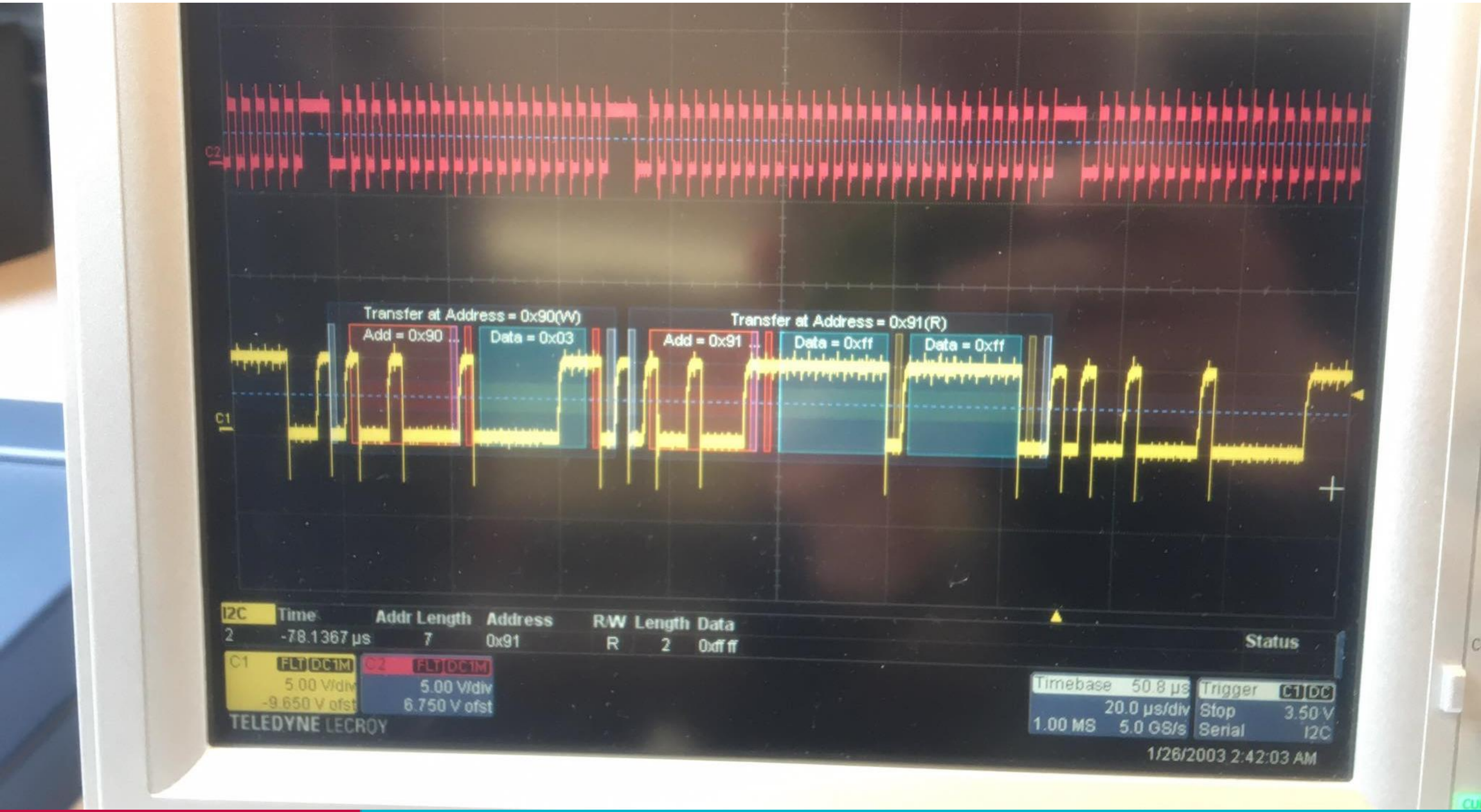
We see all the steps described in the hardware part (2 com)

Results


```
Wave - Default
                              Msgs
/driver_tb/sCLK               1
/driver_tb/sRST               0
/driver_tb/sSDA               1
/driver_tb/sSCL               1
/driver_tb/dSDA               1
/driver_tb/dREGIN             00000000       00000000
/driver_tb/dREGIN...          00000000       00000000
/driver_tb/slaveAdd...        10010000       10010000
/driver_tb/slaveAdd...        10010001       10010001
/driver_tb/aa                 00000000       00000000 | 10000000 | 10010000 | 00010000 | 00000000 | 00000010 | 00000011
/driver_tb/temp               110011001100   110011001100
/driver_tb/DUT/state          0              0 | 1 | 2 | 3 | 2 | 3 | 2 | 3 | 2 | 3 | 2 | 3 | 2 | 3 | 2 | 3 | 2 | 3 | 4 | 5 | 6 | 7 | 6 | 7 | 6 | 7 | 6 | 7 | 6 | 7 | 6 | 7 | 6 | 7 | 6 | 7 | 8 | 9 | 16 | 17

Now       11314740 ns
Cursor 1  0.00 ns
```

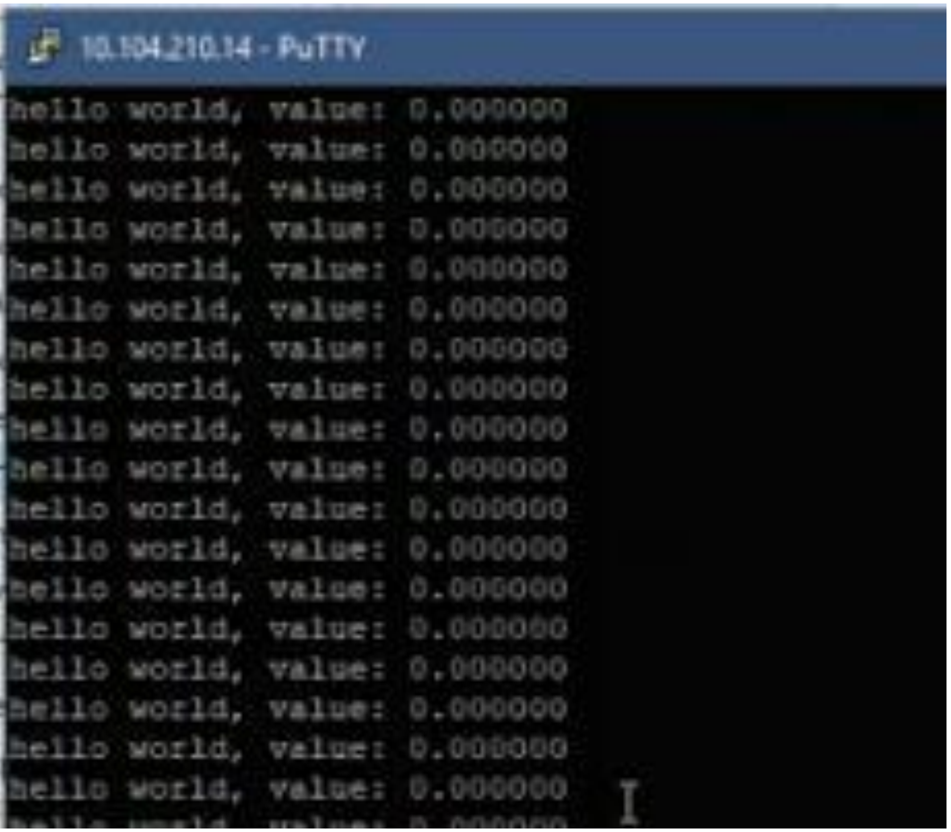
We see all the differents states changing over time

Results (SCL and SDA)



I2C	Time	Addr Length	Address	R/W	Length	Data	Status
2	-78.1367 µs	7	0x91	R	2	0xff ff	

Results displayed by software



```
hello world, value: 0.000000
hello world, value: 0.000000
hello world, value: 0.000000
hello world, value: 0.000000
hello world, value: 0.000000
hello world, value: 0.000000
hello world, value: 0.000000
hello world, value: 0.000000
hello world, value: 0.000000
hello world, value: 0.000000
hello world, value: 0.000000
hello world, value: 0.000000
hello world, value: 0.000000
hello world, value: 0.000000
hello world, value: 0.000000
hello world, value: 0.000000
hello world, value: 0.000000
hello world, value: 0.000000
```

Result=0 because no sensor connected