

## Hardware Software platforms : Project presentation I2C receiver temperature sensor

Denis Pierre  
Liégeois Loïc  
Rasic Cyril

# 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

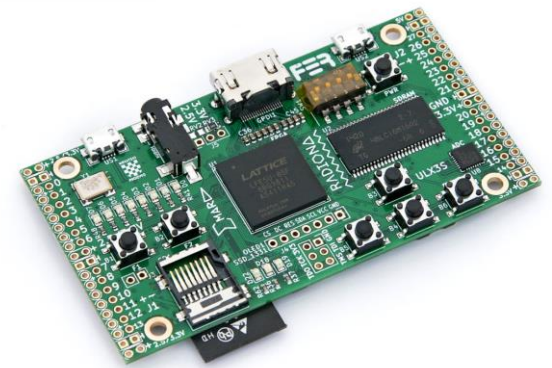
# Project presentation

- 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

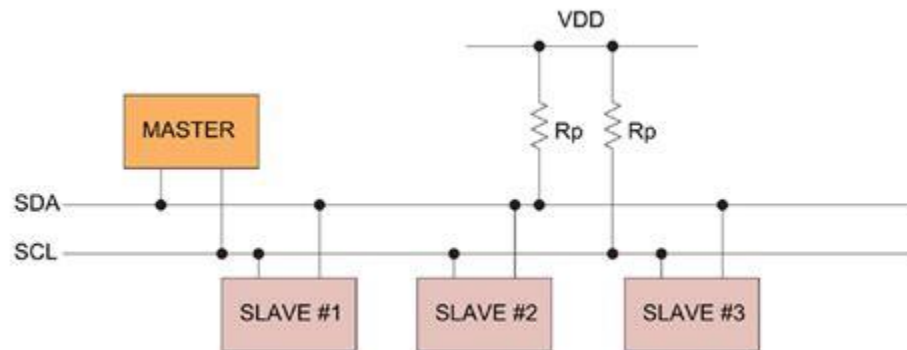
# FPGA



- 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



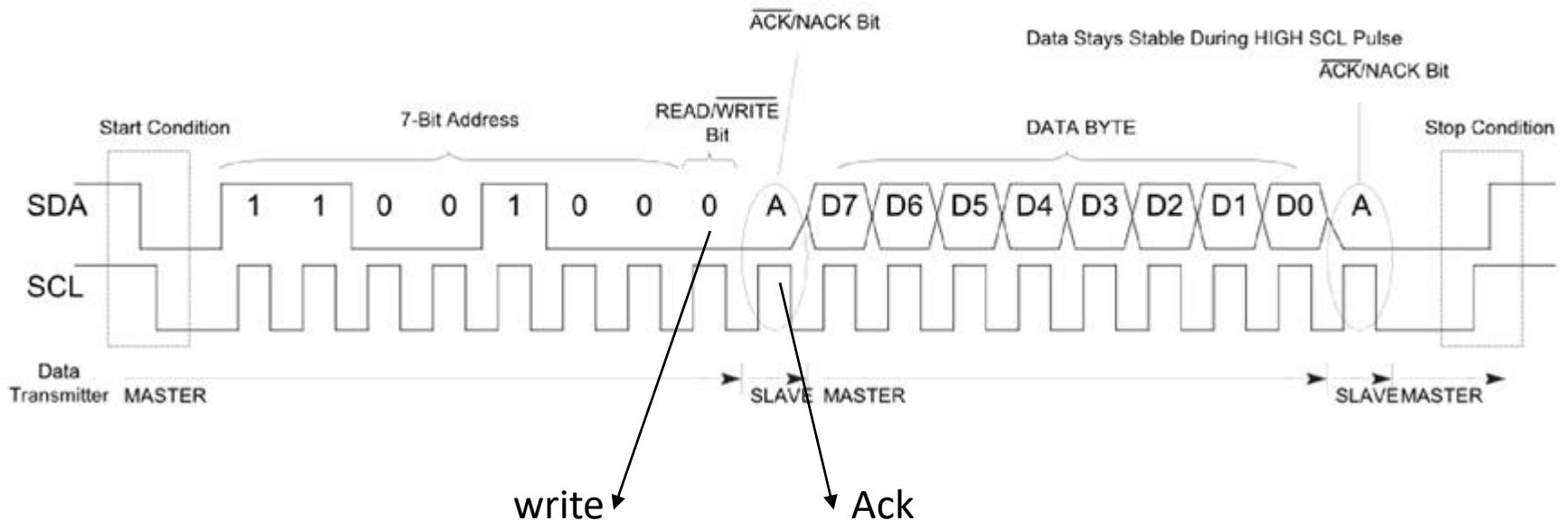
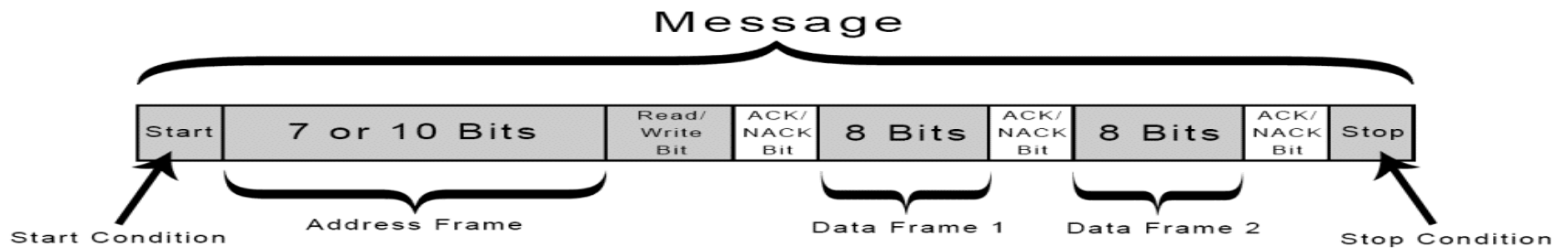
# I2C

- Frame :
  - Start condition (high->low while clk is high)
  - Address 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 :



# 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

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity driver is
5     port
6     (
7         -- Input ports
8         CLK : in  std_logic;
9         RST : in  std_logic;
10        SDAin  : in  std_logic;
11
12        -- Output ports
13        SCL : out  std_logic;
14        SDAout : out  std_logic;
15        REGIN  : out std_logic_vector(7 downto 0);
16        REGINOUT : out std_logic_vector(7 downto 0)
17    );
18 end driver;
```

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

# Definition of the architecture

```
20 architecture rtl of driver is
21
22     signal slaveAddress_write: std_logic_vector(7 downto 0) := "10010000";
23     signal slaveAddress_read: std_logic_vector(7 downto 0) := "10010001";
24     signal registerSettings : std_logic_vector(7 downto 0) := "00000011";
25
26     signal update_clk : std_logic := '0';
27     signal SCL_Clk : std_logic := '0';
28     signal SCLout : std_logic := '1';
29     signal clk_cnt : integer range 0 to 64 := 0;
30     signal clk_cnt2 : integer range 0 to 32 := 0;
31
32     signal bitcount : integer range 0 to 20 := 0;
33
34     signal data1 : std_logic_vector(7 downto 0) := "00000000";
35     signal data2 : std_logic_vector(7 downto 0) := "00000000";
36     signal DataIndicator : std_logic_vector(4 downto 0) := "00000";
37
38     signal state : std_logic_vector(7 downto 0) := x"00";
39     signal sda01 : std_logic := '1';
40     signal slaveACK : std_logic := '0';
41     signal errors : std_logic_vector(3 downto 0) := "0000";
42
43 begin
44
45     SDAout <= '1' when sda01 = '1' else '0'; -- converts sda01 from 0 or 1 to 0 or Z
46     SCL <= SCLout;
47     REGIN <= data1;
48     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

```
50 process(CLK)
51 begin
52     If rising_edge(CLK) then
53         if clk_cnt < 64 then
54             clk_cnt <= clk_cnt +1;
55         else
56             clk_cnt <= 0;
57         end if;
58
59         if clk_cnt < 32 or clk_cnt = 32 then
60             SCL_Clk <= '0';
61         elsif clk_cnt > 32 then
62             SCL_Clk <= '1';
63         end if;
64
65
66         if clk_cnt2 < 32 then
67             clk_cnt2 <= clk_cnt2 +1;
68         else
69             clk_cnt2 <= 0;
70         end if;
71
72         if clk_cnt2 < 16 or clk_cnt2 = 16 then
73             update_clk <= '0';
74         elsif clk_cnt2 > 16 then
75             update_clk <= '1';
76         end if;
77
78
79
80     end if;
81 end process;
```

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

# 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

```

1
2 PROCESS(SCL_Clk, RST, update_clk)
3 BEGIN
4     if(rising_edge(SCL_Clk)) then
5         CASE state IS
6             -----IDLE CONDITION
7             when x"00" => -- idle
8                 SCLout <= '1'; -- SCL = 1
9                 sda01 <= '1'; -- SDA = 1
10                state <= x"01";
11
12            -----START CONDITION
13            when x"01" => -- start condition
14                SCLout <= '1'; -- SCL stays 1 while
15                sda01 <= '0'; -- SDA transitions low
16                bitcount <= 7;
17                state <= x"02";
18
19            -----WRITE ADDRESS
20            when x"02" => -- sda transition state
21                SCLout <= '0'; -- when scl low
22                --sda01 <= slaveAddress_write(bitcount);
23                state <= x"03";
24
25            when x"03" => -- write address state prt2
26                SCLout <= '1';
27                if bitcount - 1 >= 0 then
28                    bitcount <= bitcount -1;
29                    state <= x"02";
30                else
31                    bitcount <= 7;
32                    state <= x"04";
33                end if;
34
35            -----SLAVE ACK
36            when x"04" => -- slave ack bit prt1
37                SCLout <= '0'; -- SCL = 1
38                sda01 <= '1'; -- SDA = 1
39                state <= x"05";
40
41            when x"05" => -- slave ack bit prt2
42                SCLout <= '1'; -- SCL = 1
43                slaveACK <= SDAin; -- 0 = ack, 1 = error
44                if SDAin = '1' then
45                    state <= x"EE";
46                    errors <= "1000";
47                else
48                    state <= x"06";
49                end if;
50
51        end case;
52    end if;
53 END

```

```

133                                     --WRITE TO REGISTER
134     when x"06" => -- sda transition state
135         SCLout <= '0'; -- when scl low
136         --sda01 <= registerSettings(bitcount);
137         state <= x"07";
138
139     when x"07" => -- write register state prt2
140         SCLout <= '1';
141         if bitcount - 1 >= 0 then
142             bitcount <= bitcount - 1;
143             state <= x"06";
144         else
145             bitcount <= 7;
146             state <= x"08";
147         end if;
148
149 -----SLAVE ACK
150     when x"08" => -- slave ack bit prt1
151         SCLout <= '0'; -- SCL = 1
152         sda01 <= '1'; -- SDA = 1
153         state <= x"09";
154
155     when x"09" => -- slave ack bit prt2
156         SCLout <= '1'; -- SCL = 1
157         slaveAck <= SDAin; -- 0 = ack, 1 = error
158         if SDAin = '1' then
159             state <= x"EE";
160             errors <= "0100";
161         else
162             state <= x"10";
163         end if;
164
165 -----STOP CONDITION
166     when x"10" => -- stop
167         SCLout <= '0'; -- SCL = 1
168         sda01 <= '0'; -- SDA = 1
169         state <= x"11";
170
171     when x"11" => --
172         SCLout <= '1'; -- SCL 1 while
173         sda01 <= '0'; -- SDA stays low
174         state <= x"12";
175
176     when x"12" => -- stop
177         SCLout <= '1'; -- SCL stays 1
178         sda01 <= '1'; -- SDA transitions to 1
179         state <= x"13";

```



# Communication 1

```

179 -----START CONDITION
180
181 when x"13" => -- idle
182   SCLout <= '1'; -- SCL = 1
183   sda01 <= '1'; -- SDA = 1
184   state <= x"14";
185
186 when x"14" => -- start condition
187   SCLout <= '1'; -- SCL stays 1 while
188   sda01 <= '0'; -- SDA transitions low
189   bitcount <= 7;
190   state <= x"15";
191 -----WRITE ADDRESS
192
193 when x"15" => -- sda transition state
194   SCLout <= '0'; -- when scl low
195   --sda01 <= slaveAddress_read(bitcount);
196   state <= x"16";
197
198 when x"16" => -- write address state prt2
199   SCLout <= '1';
200   if bitcount - 1 >= 0 then
201     bitcount <= bitcount -1;
202     state <= x"15";
203   else
204     bitcount <= 7;
205     state <= x"17";
206   end if;
207 -----SLAVE ACK
208
209 when x"17" => -- slave ack bit prt1
210   SCLout <= '0'; -- SCL = 1
211   sda01 <= '1'; -- SDA = 1
212   state <= x"18";
213
214 when x"18" => -- slave ack bit prt2
215   SCLout <= '1'; -- SCL = 1
216   slaveAck <= SDAin; -- 0 = ack, 1 = error
217   bitcount <= 7;
218   if SDAin = '1' then
219     state <= x"EE";
220     errors <= "0010";
221   else
222     state <= x"21";
223   end if;
224
225 -----read 8 MSB from converter
226
227 when x"21" => -- sda transition state
228   SCLout <= '0'; -- when scl low
229   sda01 <= '1';
230   state <= x"22";
231
232 when x"22" => -- write register state prt2
233   SCLout <= '1';
234   sda01 <= '1';
235   if bitcount - 1 >= 0 then
236     --data1(bitcount) <= SDAin;
237     bitcount <= bitcount -1;
238     state <= x"21";
239   else
240     --data1(bitcount) <= SDAin;
241     bitcount <= 7;
242     state <= x"23";
243   end if;
244 -----MASTER ACK
245
246 when x"23" => -- ack bit prt1
247   SCLout <= '0'; -- SCL = 1
248   --sda01 <= '0'; -- SDA = 1
249   state <= x"24";
250
251 when x"24" => -- ack bit prt2
252   SCLout <= '1'; -- SCL = 1
253   sda01 <= '0'; -- SDA = 0
254   bitcount <= 7;
255   state <= x"25";
256 -----read 8 LSB from converter
257
258 when x"25" => -- sda transition state
259   SCLout <= '0'; -- when scl low
260   sda01 <= '1';
261   state <= x"26";
262
263 when x"26" => -- loops back to stop condition to take
264   SCLout <= '1'; -- another 12 bits of data
265   sda01 <= '1';
266   if bitcount - 1 >= 0 then
267     --data2(bitcount) <= SDAin;
268     bitcount <= bitcount -1;
269     state <= x"25";
270   else
271     --data2(bitcount) <= SDAin;
272     bitcount <= 7;
273     state <= x"27";
274   end if;
275 -----MASTER ACK
276
277 when x"27" => -- ack bit prt1
278   SCLout <= '0'; -- SCL = 1
279   --sda01 <= '0'; -- SDA = 1
280   state <= x"28";
281
282 when x"28" => -- ack bit prt2
283   SCLout <= '1'; -- SCL = 1
284   sda01 <= '0'; -- SDA = 0
285   bitcount <= 7;
286   state <= x"29";
287 -----STOP CONDITION
288
289 when x"29" => -- stop
290   SCLout <= '0'; -- SCL = 1
291   sda01 <= '0'; -- SDA = 1
292   state <= x"30";
293
294 when x"30" => --
295   SCLout <= '1'; -- SCL 1 while
296   sda01 <= '0'; -- SDA stays low
297   state <= x"31";
298
299 when x"31" => -- stop
300   SCLout <= '1'; -- SCL stays 1
301   sda01 <= '1'; -- SDA transitions to 1
302   state <= x"00";
303
304 when others =>
305   SCLout <= '1';
306   sda01 <= '1';
307   state <= x"00";
308 END CASE;
309 end if;
310 end process;

```

# Communication 2

```
311 PROCESS(RST, update_clk)
312 BEGIN
313   if(rising_edge(update_clk)) then
314     CASE state IS
315     -----IDLE CONDITION
316     when x"00" => -- idle
317       --SCLout <= '1'; -- SCL = 1
318       --sda01 <= '1'; -- SDA = 1
319       --state <= x"01";
320     -----START CONDITION
321     when x"01" => -- start condition
322       --SCLout <= '1'; -- SCL stays 1 while
323       --sda01 <= '0'; -- SDA transitions low
324       --bitcount <= 7;
325       --state <= x"02";
326     -----WRITE ADDRESS
327     when x"02" => -- sda transition state
328       --SCLout <= '0'; -- when scl low
329       sda01 <= slaveAddress_write(bitcount);
330       --state <= x"03";
331     when x"03" => -- write address state prt2
332       --SCLout <= '1';
333       if bitcount - 1 >= 0 then
334         --bitcount <= bitcount -1;
335         --state <= x"02";
336       else
337         bitcount <= 7;
338         --state <= x"04";
339       end if;
340     -----SLAVE ACK
341     when x"04" => -- slave ack bit prt1
342       --SCLout <= '0'; -- SCL = 1
343       sda01 <= '1'; -- SDA = 1
344       --state <= x"05";
345     when x"05" => -- slave ack bit prt2
346       --SCLout <= '1'; -- SCL = 1
347       --slaveACK <= SDAin; -- 0 = ack, 1 = error
348       if SDAin = '1' then
349         --state <= x"EE";
350         --errors <= "1000";
351       else
352         --state <= x"06";
353       end if;
354     -----WRITE TO REGISTER
355     when x"06" => -- sda transition state
356       --SCLout <= '0'; -- when scl low
357       sda01 <= registerSettings(bitcount);
358       --state <= x"07";
359     when x"07" => -- write register state prt2
360       --SCLout <= '1';
361       if bitcount - 1 >= 0 then
362         --bitcount <= bitcount -1;
363         --state <= x"06";
364       else
365         bitcount <= 7;
366         --state <= x"08";
367       end if;
368     end if;
369   end if;
370 end if;
371
```

```
372 -----SLAVE ACK
373 when x"08" => -- slave ack bit prt1
374   --SCLout <= '0'; -- SCL = 1
375   --sda01 <= '1'; -- SDA = 1
376   --state <= x"09";
377
378 when x"09" => -- slave ack bit prt2
379   --SCLout <= '1'; -- SCL = 1
380   --slaveACK <= SDAin; -- 0 = ack, 1 = error
381   if SDAin = '1' then
382     --state <= x"EE";
383     --errors <= "0100";
384   else
385     --state <= x"10";
386   end if;
387 -----STOP CONDITION
388 when x"10" => -- stop
389   --SCLout <= '0'; -- SCL = 1
390   --sda01 <= '0'; -- SDA = 1
391   --state <= x"11";
392
393 when x"11" => --
394   --SCLout <= '1'; -- SCL 1 while
395   --sda01 <= '0'; -- SDA stays low
396   --state <= x"12";
397
398 when x"12" => -- stop
399   --SCLout <= '1'; -- SCL stays 1
400   --sda01 <= '1'; -- SDA transitions to 1
401   --state <= x"13";
402 -----START CONDITION
403 when x"13" => -- idle
404   --SCLout <= '1'; -- SCL = 1
405   --sda01 <= '1'; -- SDA = 1
406   --state <= x"14";
407
408 when x"14" => -- start condition
409   --SCLout <= '1'; -- SCL stays 1 while
410   --sda01 <= '0'; -- SDA transitions low
411   --bitcount <= 7;
412   --state <= x"15";
413
```

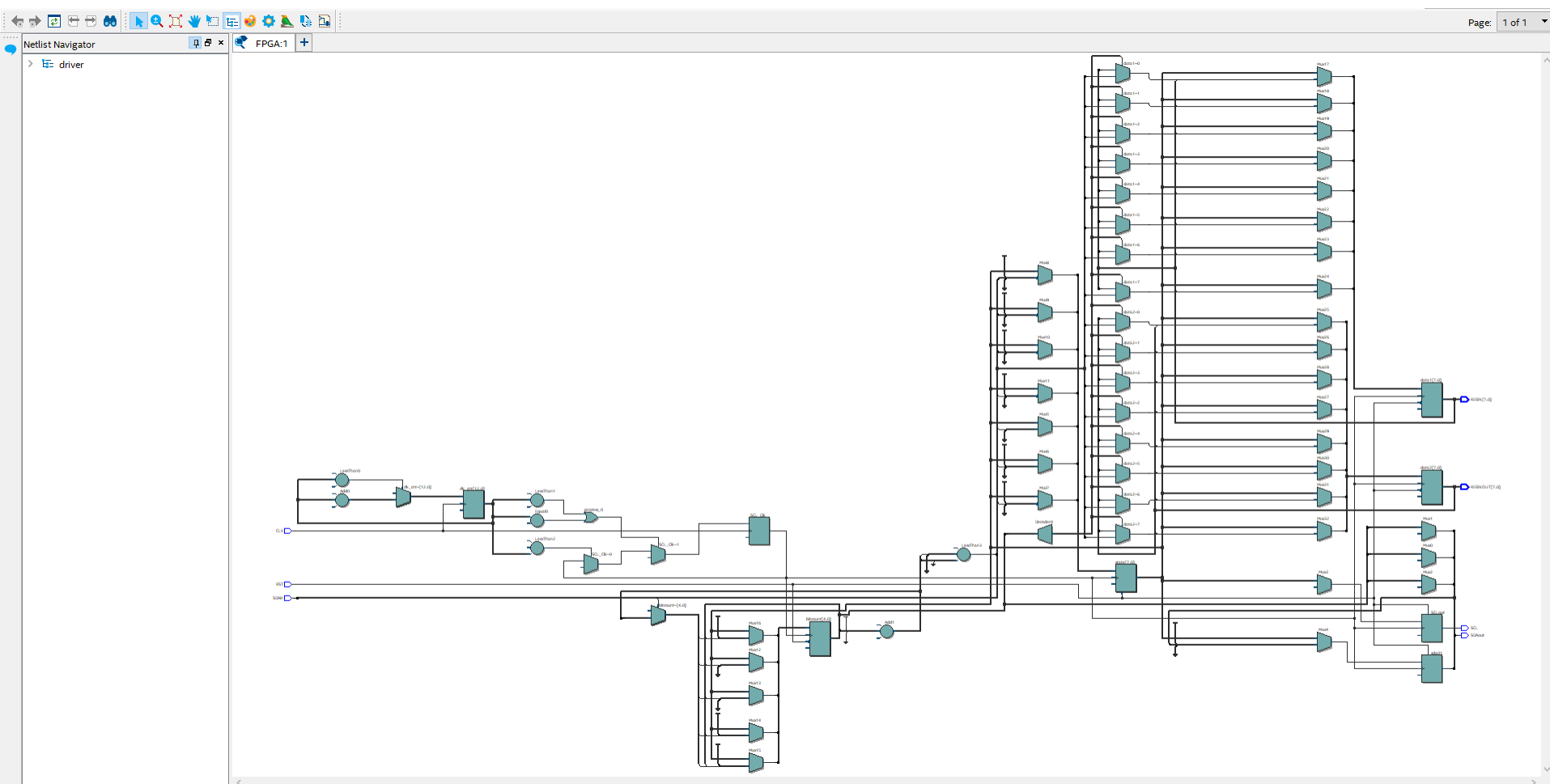
```
414 -----WRITE ADDRESS
415 when x"15" => -- sda transition state
416   --SCLout <= '0'; -- when scl low
417   sda01 <= slaveAddress_read(bitcount);
418   --state <= x"16";
419
420 when x"16" => -- write address state prt2
421   --SCLout <= '1';
422   if bitcount - 1 >= 0 then
423     --bitcount <= bitcount -1;
424     --state <= x"15";
425   else
426     --bitcount <= 7;
427     --state <= x"17";
428   end if;
429 -----SLAVE ACK
430 when x"17" => -- slave ack bit prt1
431   --SCLout <= '0'; -- SCL = 1
432   --sda01 <= '1'; -- SDA = 1
433   --state <= x"18";
434
435 when x"18" => -- slave ack bit prt2
436   --SCLout <= '1'; -- SCL = 1
437   --slaveACK <= SDAin; -- 0 = ack, 1 = error
438   --bitcount <= 7;
439   if SDAin = '1' then
440     --state <= x"EE";
441     --errors <= "0010";
442   else
443     --state <= x"21";
444   end if;
445 -----read 8 MSB from converter
446 when x"21" => -- sda transition state
447   --SCLout <= '0'; -- when scl low
448   --sda01 <= '1';
449   --state <= x"22";
450
451 when x"22" => -- write register state prt2
452   --SCLout <= '1';
453   --sda01 <= '1';
454   if bitcount - 1 >= 0 then
455     data1(bitcount) <= SDAin;
456     --bitcount <= bitcount -1;
457     --state <= x"21";
458   else
459     data1(bitcount) <= SDAin;
460     --bitcount <= 7;
461     --state <= x"23";
462   end if;
463
```

# Communication 2

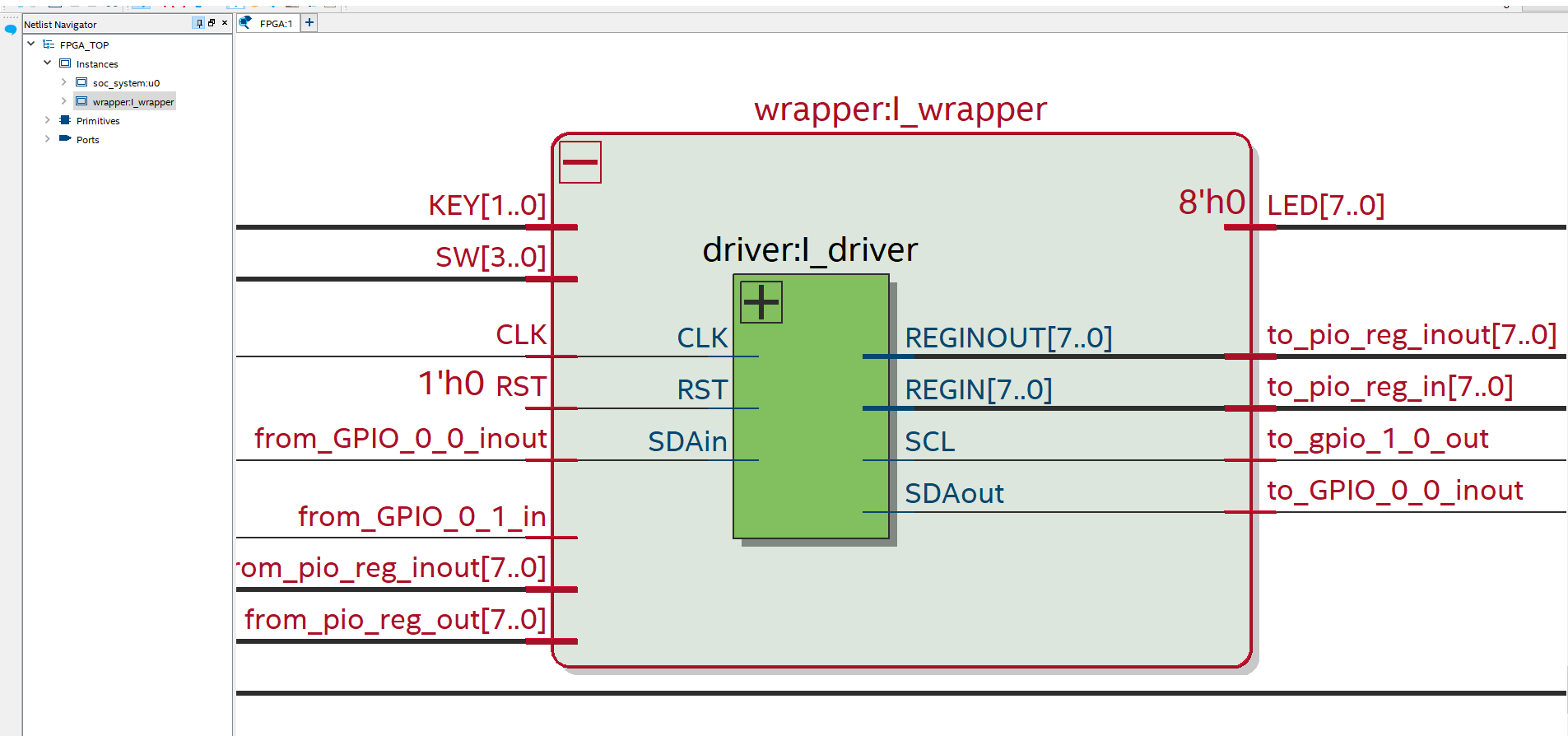
```
464 -----MASTER ACK
465 when x"23" => -- ack bit prt1
466 --SCLout <= '0'; -- SCL = 1
467 sda01 <= '0'; -- SDA = 1
468 --state <= x"24";
469
470 when x"24" => -- ack bit prt2
471 --SCLout <= '1'; -- SCL = 1
472 --sda01 <= '0'; -- SDA = 0
473 --bitcount <= 7;
474 --state <= x"25";
475 -----read 8 LSB from converter
476 when x"25" => -- sda transition state
477 --SCLout <= '0'; -- when scl low
478 --sda01 <= '1';
479 --state <= x"26";
480
481 when x"26" => -- loops back to stop condition to take
482 --SCLout <= '1'; -- another 12 bits of data
483 --sda01 <= '1';
484 if bitcount - 1 >= 0 then
485 data2(bitcount) <= SDAin;
486 --bitcount <= bitcount -1;
487 --state <= x"25";
488 else
489 data2(bitcount) <= SDAin;
490 --bitcount <= 7;
491 --state <= x"27";
492 end if;
493 -----MASTER ACK
494 when x"27" => -- ack bit prt1
495 --SCLout <= '0'; -- SCL = 1
496 sda01 <= '0'; -- SDA = 1
497 --state <= x"28";
498
499 when x"28" => -- ack bit prt2
500 --SCLout <= '1'; -- SCL = 1
501 --sda01 <= '0'; -- SDA = 0
502 --bitcount <= 7;
503 --state <= x"29";
504
```

```
507 -----STOP CONDITION
508
509 when x"29" => -- stop
510 --SCLout <= '0'; -- SCL = 1
511 --sda01 <= '0'; -- SDA = 1
512 --state <= x"30";
513
514 when x"30" => --
515 --SCLout <= '1'; -- SCL 1 while
516 --sda01 <= '0'; -- SDA stays low
517 --state <= x"31";
518
519 when x"31" => -- stop
520 --SCLout <= '1'; -- SCL stays 1
521 --sda01 <= '1'; -- SDA transitions to 1
522 --state <= x"00";
523
524
525 when others =>
526 --SCLout <= '1';
527 --sda01 <= '1';
528 --state <= x"00";
529 END CASE;
530 end if;
531
532 end process;
533
534
535
536
537
538 end rtl;
```

# Synthesis



# Synthesis



# 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
1  #include "DRIVER_read.h"
2  #include "pio_reg_in.h"
3  #include "pio_reg_inout.h"
4
5  #define RESOLUTION 0.0625
6  float DRIVER_readTemp() {
7      unsigned int firstByte = PIO_REG_IN_read();
8      unsigned int secondByte = PIO_REG_INOUT_read();
9      int intPart;
10     int decPart;
11
12     if (firstByte <= 127)
13         intPart = (int)firstByte;
14     else
15         intPart = -(int)~firstByte - 1;
16
17
18     if (secondByte <= 127)
19         decPart = (int)secondByte;
20     else
21         decPart = -(int)~secondByte - 1;
22
23
24
25     float temp = intPart;
26     temp += decPart*RESOLUTION;
27
28     return temp;
29 }
30
```



# 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

```

53 DUT: driver
54   port map (
55       RST    => sRST,
56       CLK    => sCLK,
57       SDAin  => sSDA,
58
59       SDAout  => dSDA,
60       SCL    => sSCL,
61       REGIN  => dREGIN,
62       REGINOUT => dREGINOUT
63   );
64
65   P_sCLK: process
66   begin -- 50MHz clock
67       sCLK <= not sCLK;
68       wait for PERIOD/2;
69   end process;
70
71   PP: process
72   begin -- 50MHz clock
73       sSDA <= '1';
74
75       --start
76       wait until dSDA = '0';
77
78       --address
79       wait until sSCL = '0';
80       wait until sSCL = '1';
81       aa(7) <= dSDA;
82       wait until sSCL = '1';
83       aa(6) <= dSDA;
84       wait until sSCL = '1';
85       aa(5) <= dSDA;
86       wait until sSCL = '1';
87       aa(4) <= dSDA;
88       wait until sSCL = '1';
89       aa(3) <= dSDA;
90       wait until sSCL = '1';
91       aa(2) <= dSDA;
92       wait until sSCL = '1';
93       aa(1) <= dSDA;
94       wait until sSCL = '1';
95       aa(0) <= dSDA;
96
97       --ack
98       wait until sSCL = '0';
99       sSDA <= '0';
100      wait until sSCL = '0';
101
102      sSDA <= '1';

```

```

104 --register
105 wait until sSCL = '1';
106 aa(7) <= dSDA;
107 wait until sSCL = '1';
108 aa(6) <= dSDA;
109 wait until sSCL = '1';
110 aa(5) <= dSDA;
111 wait until sSCL = '1';
112 aa(4) <= dSDA;
113 wait until sSCL = '1';
114 aa(3) <= dSDA;
115 wait until sSCL = '1';
116 aa(2) <= dSDA;
117 wait until sSCL = '1';
118 aa(1) <= dSDA;
119 wait until sSCL = '1';
120 aa(0) <= dSDA;
121
122 --ack
123 wait until sSCL = '0';
124 sSDA <= '0';
125 wait until sSCL = '0';
126 sSDA <= '1';
127
128 --stop
129 wait until sSCL = '1';
130 wait until dSDA = '1';
131
132 --start
133 wait until dSDA = '0';
134
135 --address
136 wait until sSCL = '0';
137 wait until sSCL = '1';
138 aa(7) <= dSDA;
139 wait until sSCL = '1';
140 aa(6) <= dSDA;
141 wait until sSCL = '1';
142 aa(5) <= dSDA;
143 wait until sSCL = '1';
144 aa(4) <= dSDA;
145 wait until sSCL = '1';
146 aa(3) <= dSDA;
147 wait until sSCL = '1';
148 aa(2) <= dSDA;
149 wait until sSCL = '1';
150 aa(1) <= dSDA;
151 wait until sSCL = '1';
152 aa(0) <= dSDA;
153

```

```

204 --ack
205 wait until sSCL = '0';
206 sSDA <= '0';
207
208 --write
209 wait until sSCL = '0';
210 sSDA <= temp(11);
211 wait until sSCL = '0';
212 sSDA <= temp(10);
213 wait until sSCL = '0';
214 sSDA <= temp(9);
215 wait until sSCL = '0';
216 sSDA <= temp(8);
217 wait until sSCL = '0';
218 sSDA <= temp(7);
219 wait until sSCL = '0';
220 sSDA <= temp(6);
221 wait until sSCL = '0';
222 sSDA <= temp(5);
223 wait until sSCL = '0';
224 sSDA <= temp(4);
225
226 wait until sSCL = '1';
227 wait until sSCL = '0';
228 sSDA <= '0';
229
230 --write 2
231 wait until sSCL = '0';
232 sSDA <= temp(3);
233 wait until sSCL = '0';
234 sSDA <= temp(2);
235 wait until sSCL = '0';
236 sSDA <= temp(1);
237 wait until sSCL = '0';
238 sSDA <= temp(0);
239
240 wait until sSCL = '0';
241 sSDA <= '0';
242 wait until sSCL = '0';
243 sSDA <= '0';
244 wait until sSCL = '0';
245 sSDA <= '0';
246 wait until sSCL = '0';
247 sSDA <= '0';
248
249 wait until sSCL = '0';
250 sSDA <= '1';

```

```

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

```

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

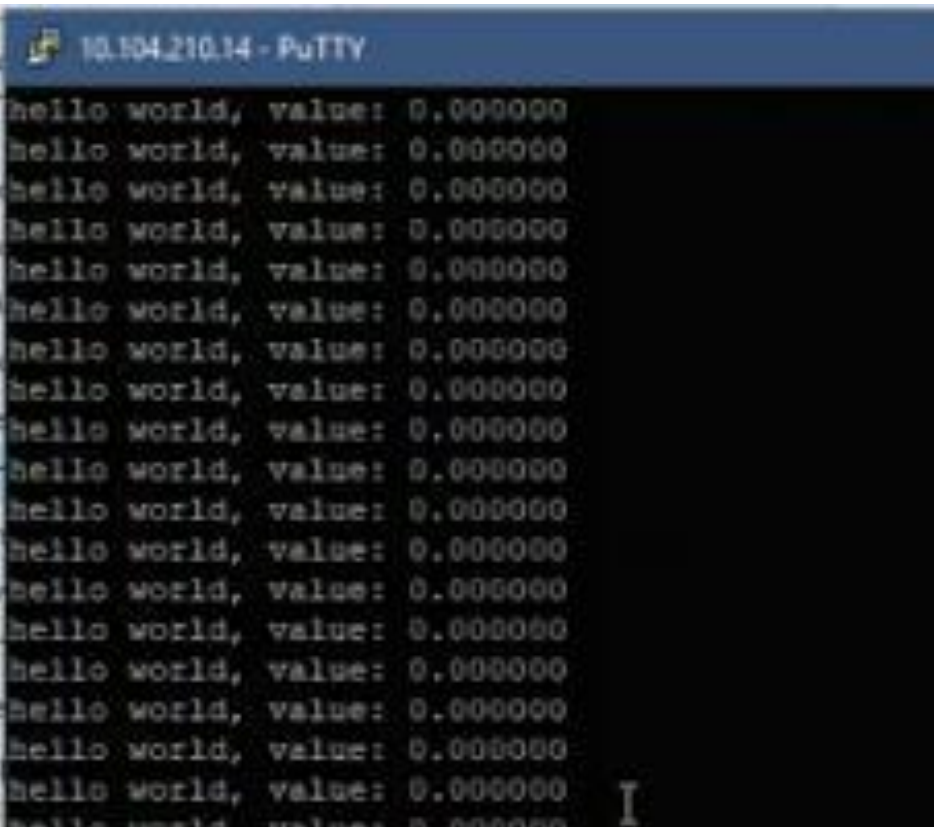
The screenshot shows the Xilinx Vivado IDE interface. The top menu bar includes 'Simulate', 'Add', 'Structure', 'Tools', 'Layout', 'Bookmarks', 'Window', and 'Help'. Below the menu is a toolbar with various simulation and layout icons. The main window displays a waveform for 'Wave - Default'. The left pane shows a list of signals: /driver\_tb/sCLK, /driver\_tb/sRST, /driver\_tb/sSDA, /driver\_tb/sSCL, /driver\_tb/dREGIN, /driver\_tb/dREGIN..., /driver\_tb/slaveAdd..., /driver\_tb/slaveAdd..., /driver\_tb/aa, /driver\_tb/temp, and /driver\_tb/DUT/state. The right pane shows the waveform for these signals. The bottom status bar indicates 'Now' at 11314740 ns and 'Cursor 1' at 0.00 ns.

Université de Mons I2C receiver temperature sensor - Rasic Cyril 28

# Results (SCL and SDA)



# Results displayed by software



A screenshot of a PuTTY terminal window titled "10.104.210.14 - PuTTY". The terminal displays a continuous stream of the text "hello world, value: 0.000000" on multiple lines. A cursor is visible at the end of the last line.

Result=0 because no sensor connected