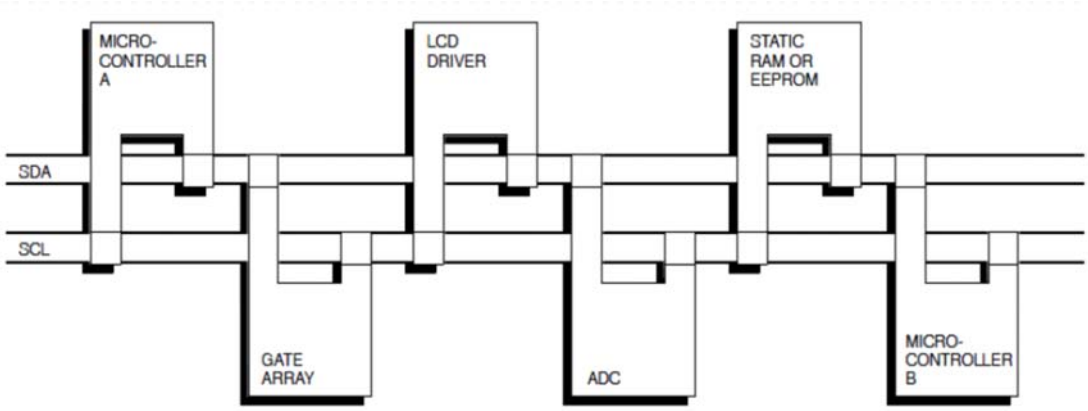
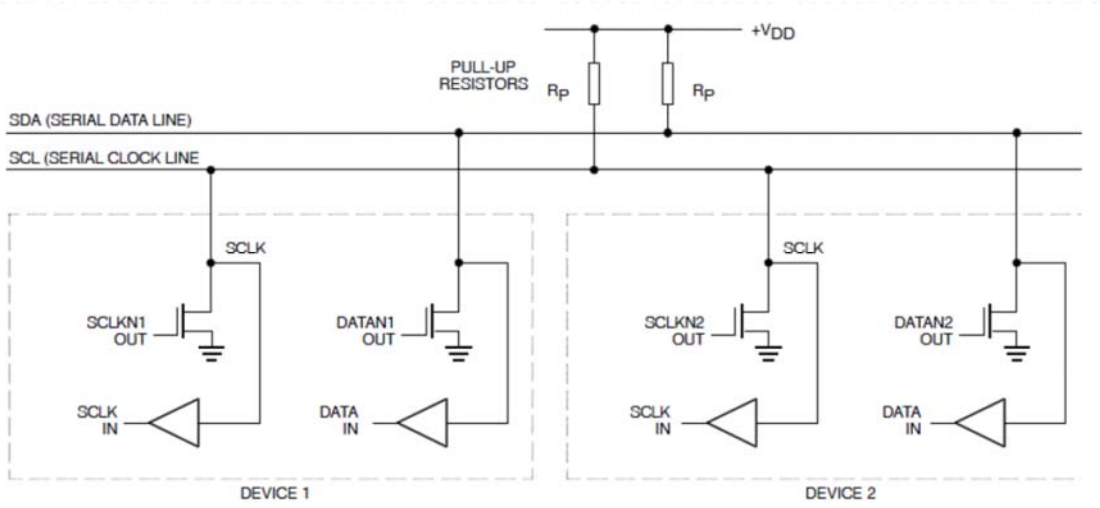
PILOTE I2C

Fabrice Aubépart – fabrice.aubepart@univ-amu.fr

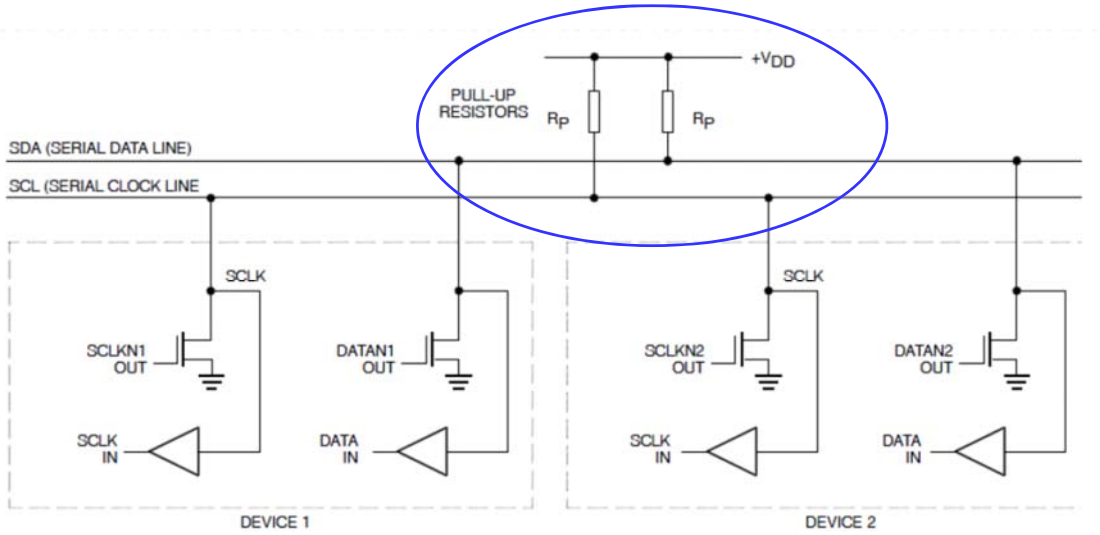
1. Caractéristiques I2C

- Bus de communication série synchrone,
- Bus bidirectionnel,
- Bus avec un protocole de reconnaissance.



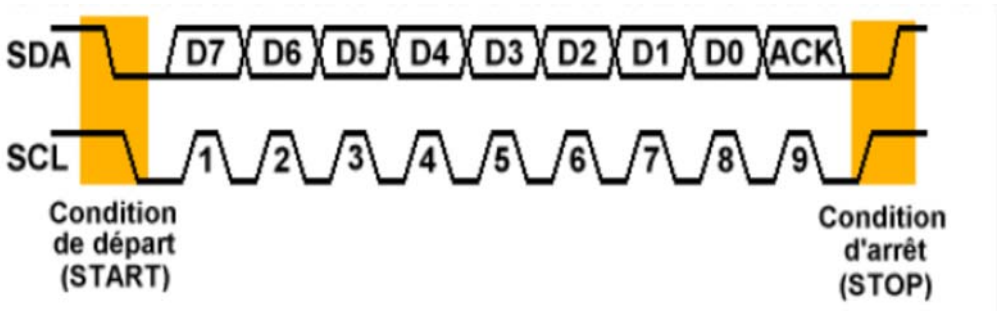


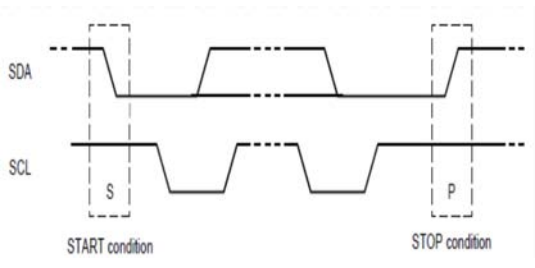
- Le bus est constitué de deux fils (plus la masse) :
  - **SCL : serial clock qui est l'horloge de cadencement des communications**
  - **SDA : serial data permet les échanges bidirectionnels entre le maitre et un esclave.**

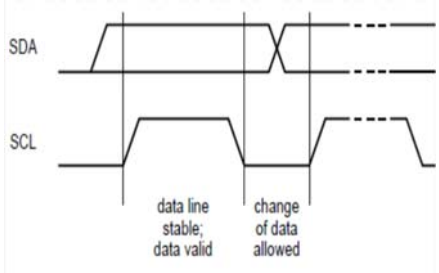


- Résistances de tirage (Pull-up resistors) réalisent un 'ET' logique cablé.
  - Intérêt : bus bidirectionnel
  - L'état de repos est donc le '1' logique (niveau V_{DD}).

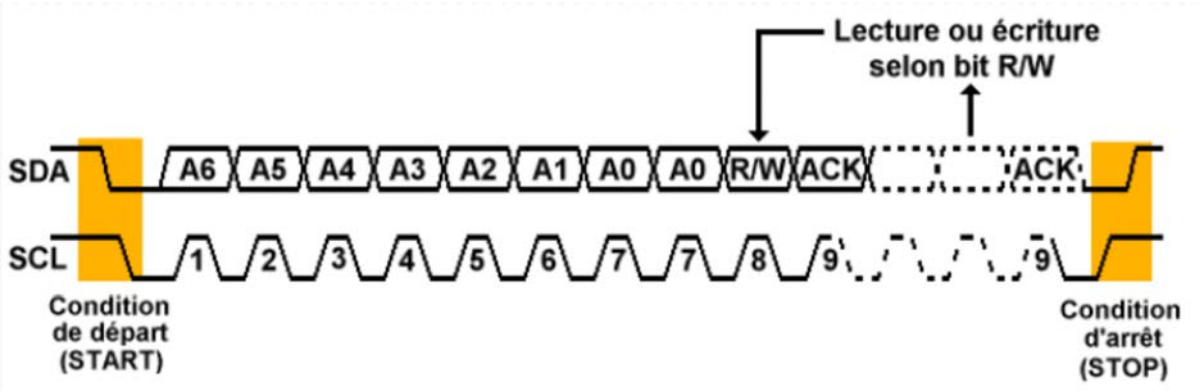
- Transmission d'une information :



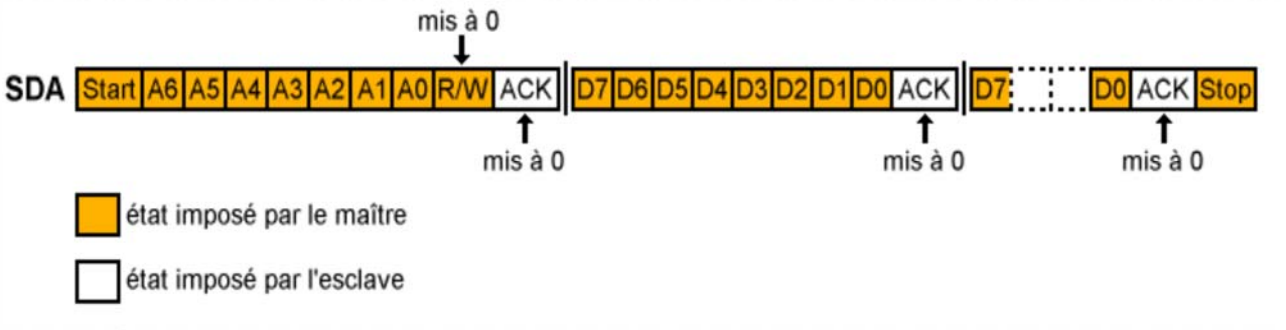




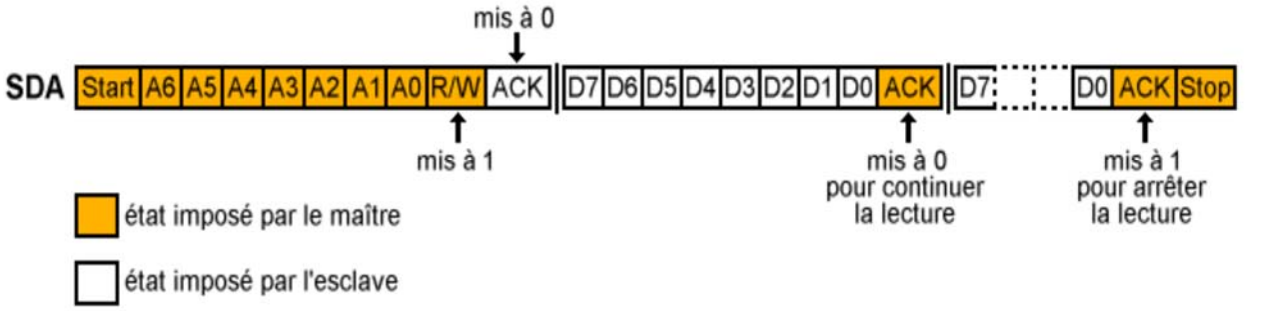
Transmission d'une adresse :



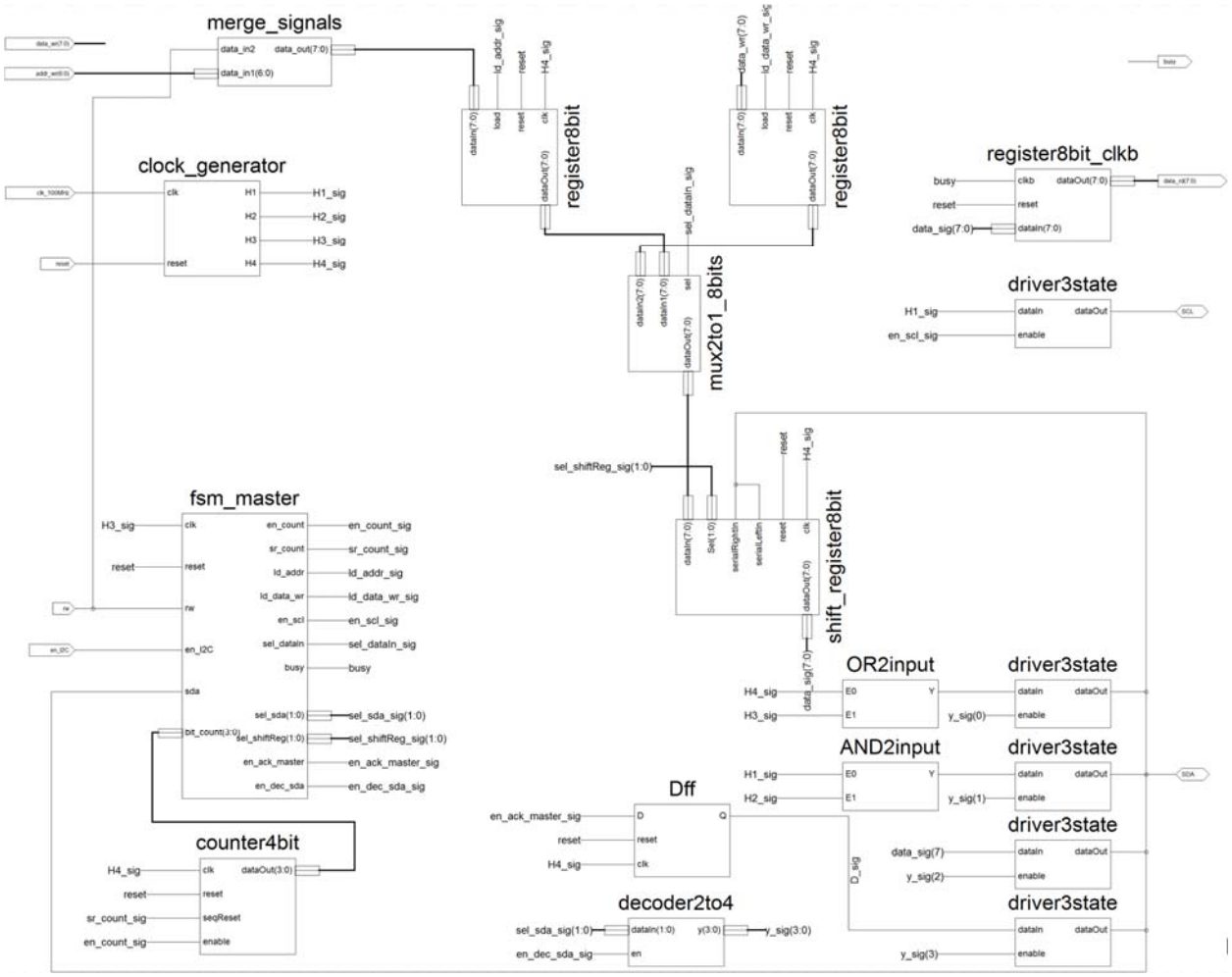
Ecriture d'une donnée (Master > Slave) :



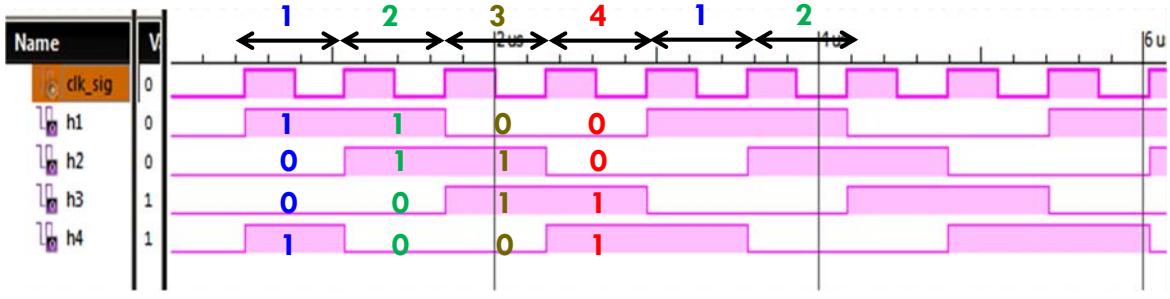
- Lecture d'une donnée (Master < Slave) :



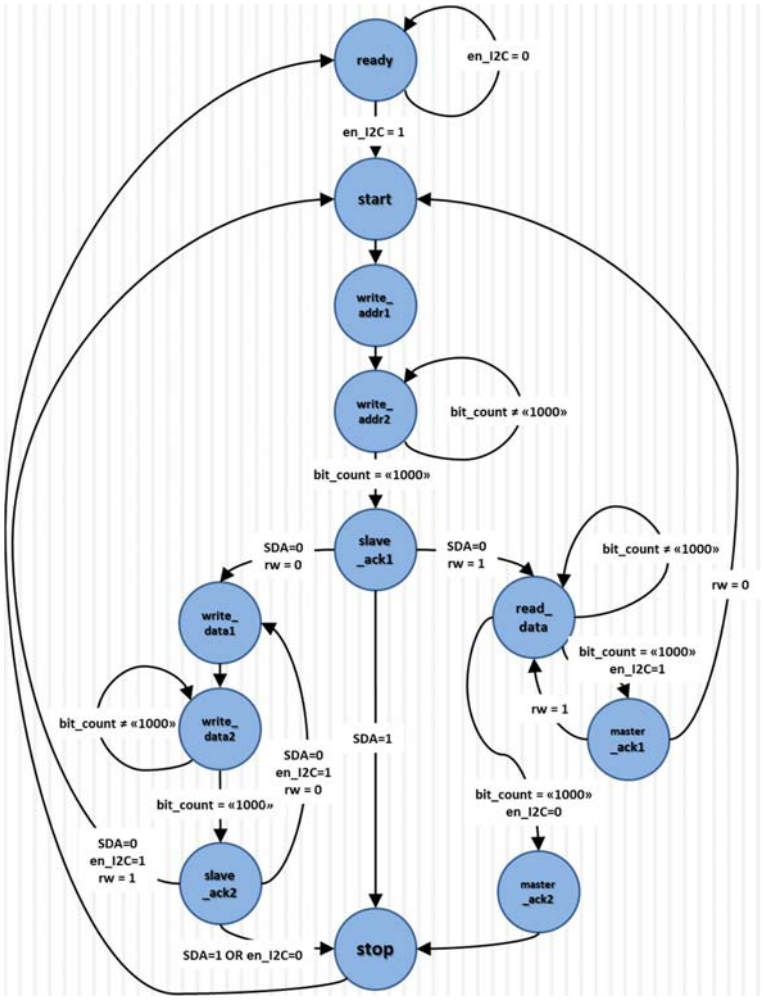
2. Architecture Master_I2C



- Fonction « clock_generator » :



3. Simulation de fsm_master



- On crée un testbench en VHDL : « test_fsm_master »
- Tous les signaux d'entrées sont initialisés

```
--Inputs
signal clk : std_logic := '0';
signal reset : std_logic := '1';
signal rw : std_logic := '0';
signal en_I2C : std_logic := '0';
signal sda : std_logic := '0';
signal bit_count : std_logic_vector(3 downto 0) := (others => '0');

--Outputs
signal en_count : std_logic;
signal sr_count : std_logic;
signal sel_sda : std_logic_vector(1 downto 0);
signal ld_addr : std_logic;
signal ld_data_wr : std_logic;
signal sel_shiftReg : std_logic_vector(1 downto 0);
signal en_scl : std_logic;
signal sel_dataIn : std_logic;
signal busy : std_logic;
signal en_ack_master : std_logic;
signal en_dec_sda : std_logic;
```


- On déclare la période du signal d'horloge (dans la zone de déclaration)
  - La période correspond à une fréquence de 400kHz

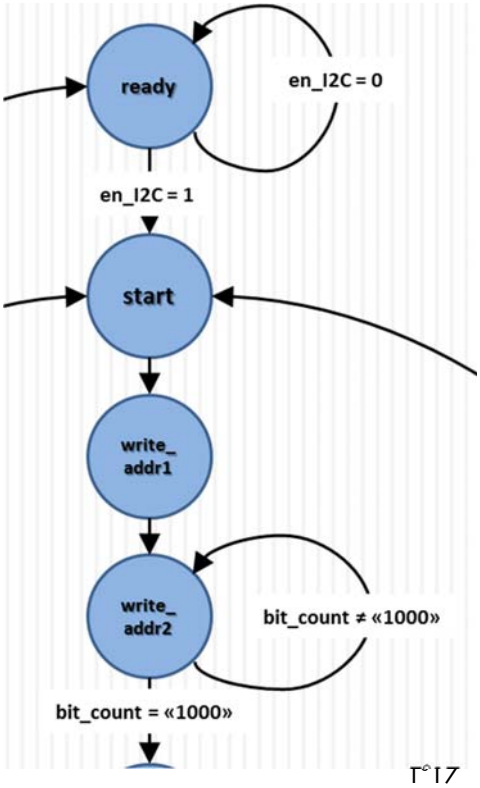
```
-- Clock period definitions
constant clk_period : time := 2.5 us;
```

- On crée le signal d'horloge avec un processus (dans l'architecture)
  - La période correspond à une fréquence de 400kHz

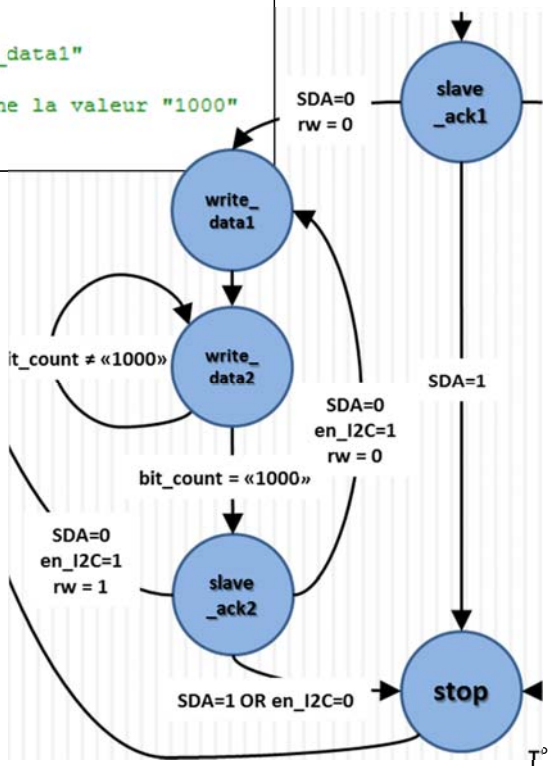
```
-- Clock process definitions
clk_process :process
begin
   clk <= '0';
   wait for clk_period/2;
   clk <= '1';
   wait for clk_period/2;
end process;
```

Création des stimuli par un processus :

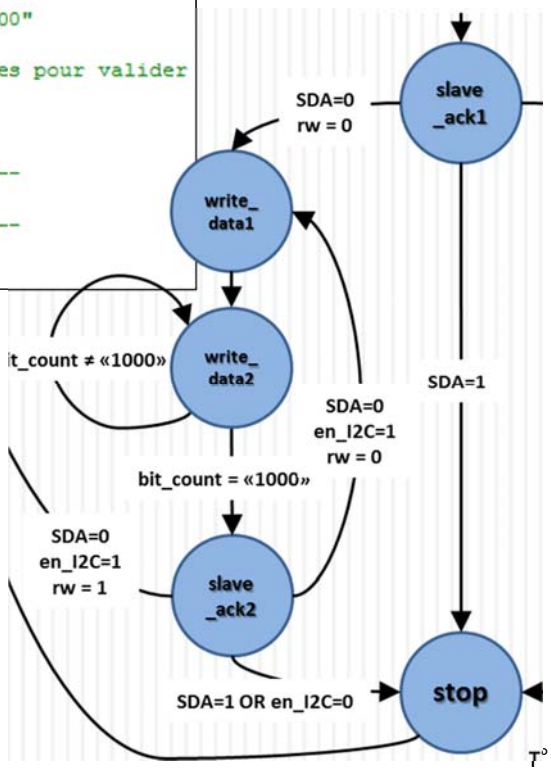
```
124     -- Stimulus process
125     stim_proc: process
126     begin
127        -- Mise à '1' du reset à t=0s
128        -- jusqu'au premier front descendant de clk
129        reset <= '1';
130        wait until (falling_edge(clk));
131        reset <= '0';
132        --
133        -- Attente de 2x2.5us = 5 us
134        wait for clk_period*2;
135        -- Mise à '1' du signal de validation
136        en_I2C <= '1' ;
137        --
138        -- La FSM se met en marche :
139        -- "ready" => "start" => "write_addr1" => "write_addr2"
140        -- elle va reseter et enclencher le compteur
141        --on attend alors que le cpt donne la valeur "1000"
142        wait until (bit_count = "1000");
143        -- "write_addr2" => "slave_ack1"
```




```
144        -- On change les états de certaines entrées pour valider
145        -- les états "write_data1" à "slave_ack2" : écriture d'une donnée
146        SDA <= '0';
147        en_I2C <= '1';
148        rw <= '0';
149        --"slave_ack1" => "write_data1" => "write_data2"
150        --on attend de nouveau que le cpt donne la valeur "1000"
151        wait until (bit_count = "1000");
152        -- "write_data2" => "slave_ack2"
153        -- teset du retour de "slave_ack2" => "write_data1"
154        -- "write_data1" => "write_data2"
155        -- on attend une seconde fois que le cpt donne la valeur "1000"
156        wait until (bit_count = "1000");
157        -- "write_data2" => "slave_ack2"
```

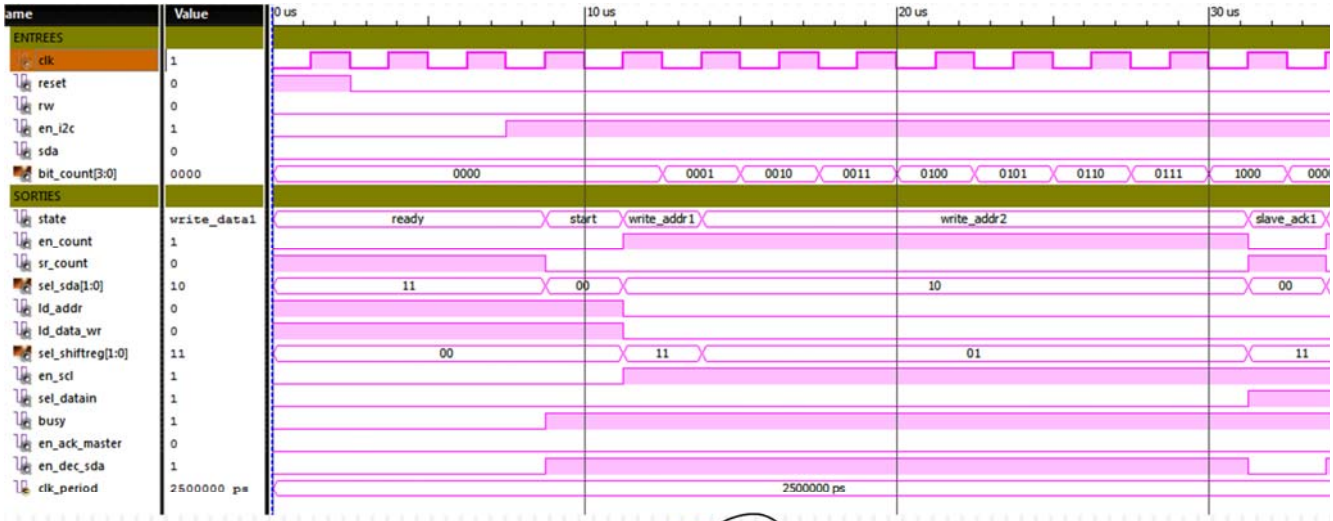


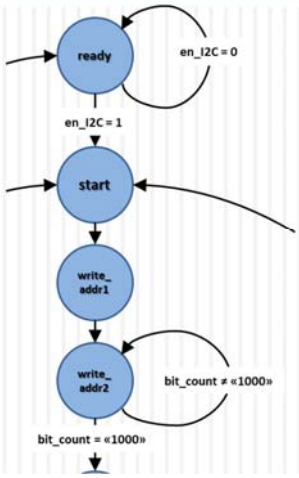
```
158         -- On change les états de certaines entrées pour valider
159         -- le retour "slave_ack2" => "start"
160         en_I2C <= '1';
161         SDA <= '0';
162         rw <= '1';
163         -- "start" => "write_addr1" => "write_addr2"
164         -- FSM resete et enclenche le compteur
165         -- attende que le cpt donne la valeur "1000"
166         wait until (bit_count = "1000");
167         -- On change les états de certaines entrées pour valider
168         -- "slave_ack1" => "stop"
169         SDA <= '1';
170         -- "stop" => "ready" => etc...
171         ----------------------------------------
172         -- A VOUS ECRIRE LA SUITE !
173         ----------------------------------------
174         wait;
175     end process;
```



- Processus pour la modélisation du counter4bit dans le contexte de la simulation

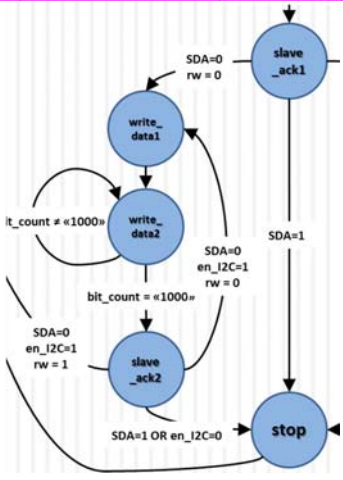
```
177
178     -- Modele de simulation de counter4bit
179     -- Attention : processus pas synthetisable !!!!!
180     counter4bit_process :process
181     begin
182        wait until (sr_count = '1');
183        wait until (falling_edge(clk));
184        bit_count <= "0000";
185        wait until (en_count = '1');
186        for i in 1 to 8 loop
187           wait until (falling_edge(clk));
188           bit_count <= bit_count + 1;
189           report "compteur incrémenté";
190        end loop;
191     end process;
192
```


Value
ENTREES
clk
reset
rw
en_i2c
sda
bit_count[3:0]
SORTIES
state
en_count
sr_count
sel_sda[1:0]
ld_addr
ld_data_wr
sel_shiftreg[1:0]
en_scl
sel_datain
busy
en_ack_master
en_dec_sda
clk_period
write_data1
2500000 ps
0 us
10 us
20 us
30 us
ready
start
write_addr1
write_addr2
slave_ack1

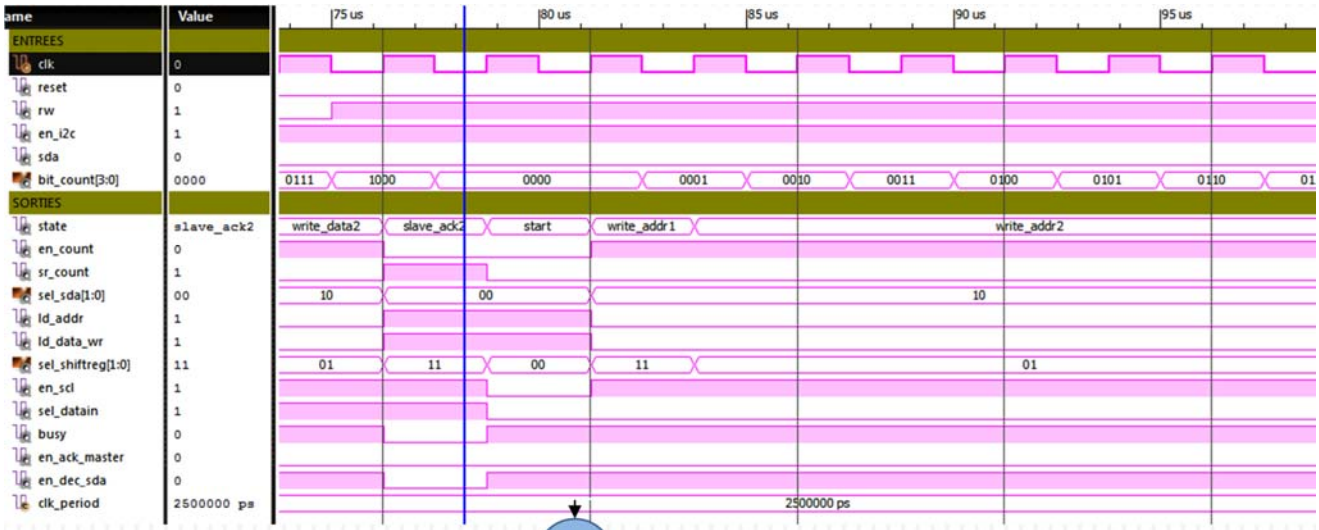

ready
en_I2C = 0
en_I2C = 1
start
write_ addr1
write_ addr2
bit_count ≠ «1000»
bit_count = «1000»

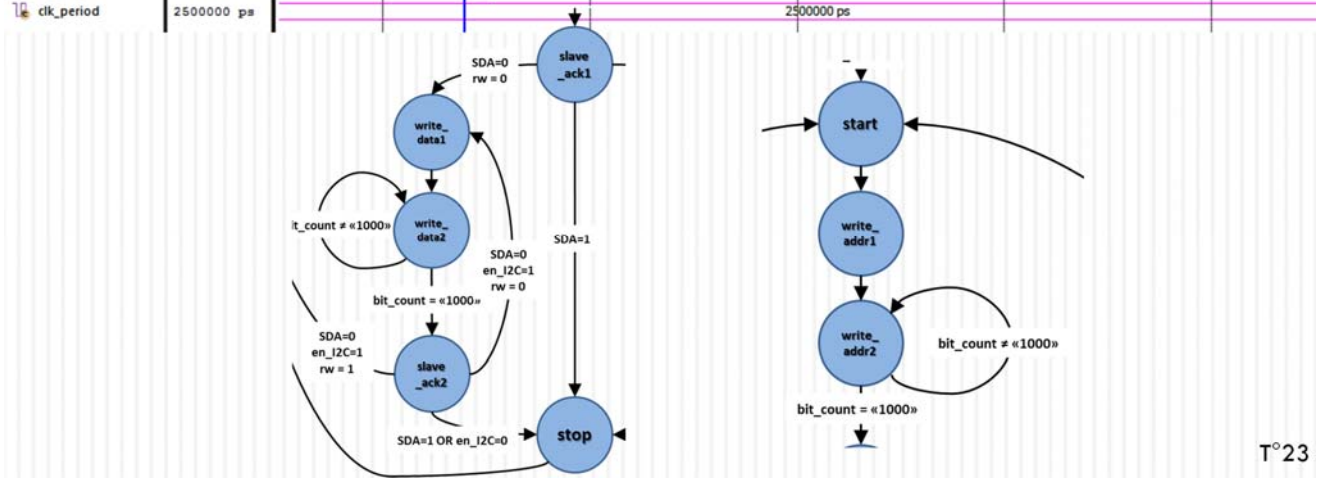
T°21


Value
ENTREES
clk
reset
rw
en_i2c
sda
bit_count[3:0]
SORTIES
state
en_count
sr_count
sel_sda[1:0]
ld_addr
ld_data_wr
sel_shiftreg[1:0]
en_scl
sel_datain
busy
en_ack_master
en_dec_sda
clk_period
0001
write_data1
2500000 ps
35 us
40 us
45 us
50 us
55 us
slave_ack1
write_data2
slave_ack2

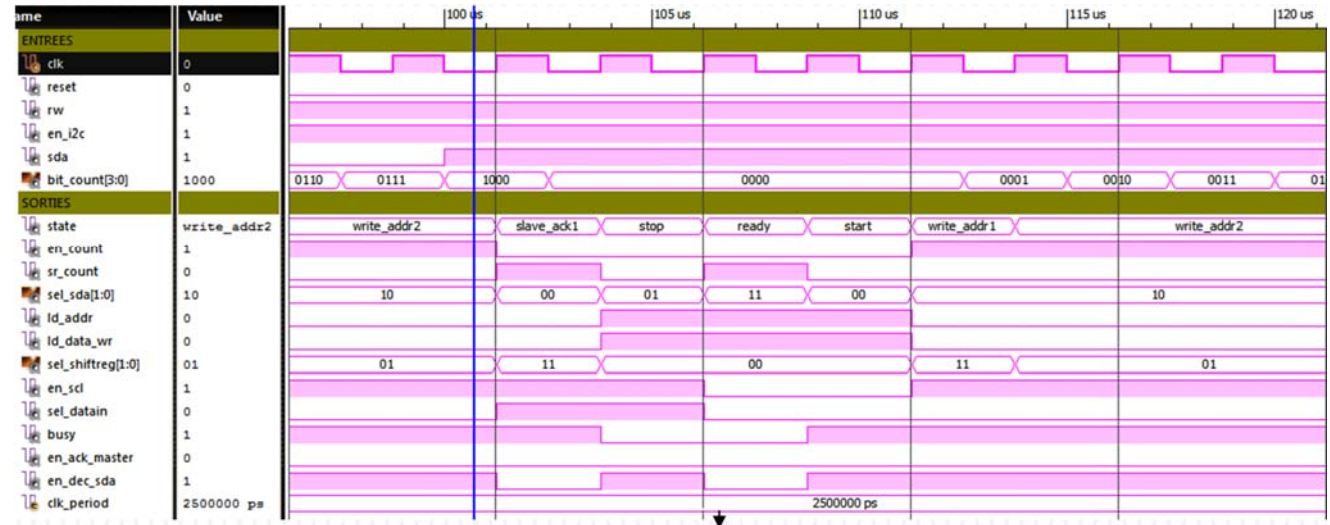

SDA=0
rw = 0
slave _ack1
write_ data1
write_ data2
bit_count ≠ «1000»
SDA=1
SDA=0
en_I2C=1
rw = 0
bit_count = «1000»
SDA=0
en_I2C=1
rw = 1
slave _ack2
SDA=1 OR en_I2C=0
stop

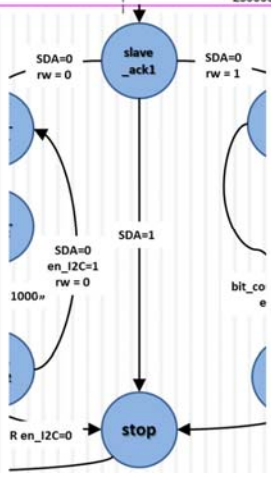
T°22


Value
ENTREES
clk
reset
rw
en_i2c
sda
bit_count[3:0]
SORTIES
state
en_count
sr_count
sel_sda[1:0]
ld_addr
ld_data_wr
sel_shiftreg[1:0]
en_scl
sel_datain
busy
en_ack_master
en_dec_sda
clk_period
75 us
80 us
85 us
90 us
95 us
write_data2
slave_ack2
start
write_addr1
write_addr2
2500000 ps


slave _ack1
SDA=0 rw = 0
write_ data1
write_ data2
SDA=1
SDA=0 en_I2C=1 rw = 0
bit_count = «1000»
SDA=0 en_I2C=1 rw = 1
slave _ack2
SDA=1 OR en_I2C=0
stop
start
write_ addr1
write_ addr2
bit_count ≠ «1000»

T°23

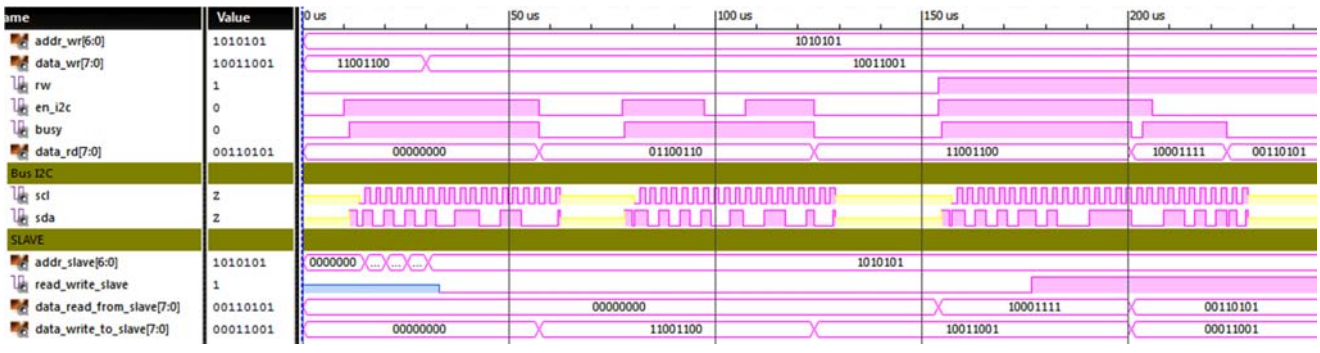

Value
ENTREES
clk
reset
rw
en_i2c
sda
bit_count[3:0]
SORTIES
state
en_count
sr_count
sel_sda[1:0]
ld_addr
ld_data_wr
sel_shiftreg[1:0]
en_scl
sel_datain
busy
en_ack_master
en_dec_sda
clk_period
100 us
105 us
110 us
115 us
120 us
write_addr2
slave_ack1
stop
ready
start
write_addr1
2500000 ps


slave _ack1
SDA=0 rw = 0
SDA=0 rw = 1
SDA=1
SDA=0 en_I2C=1 rw = 0
stop

T°24

4. Simulation du Master_I2C

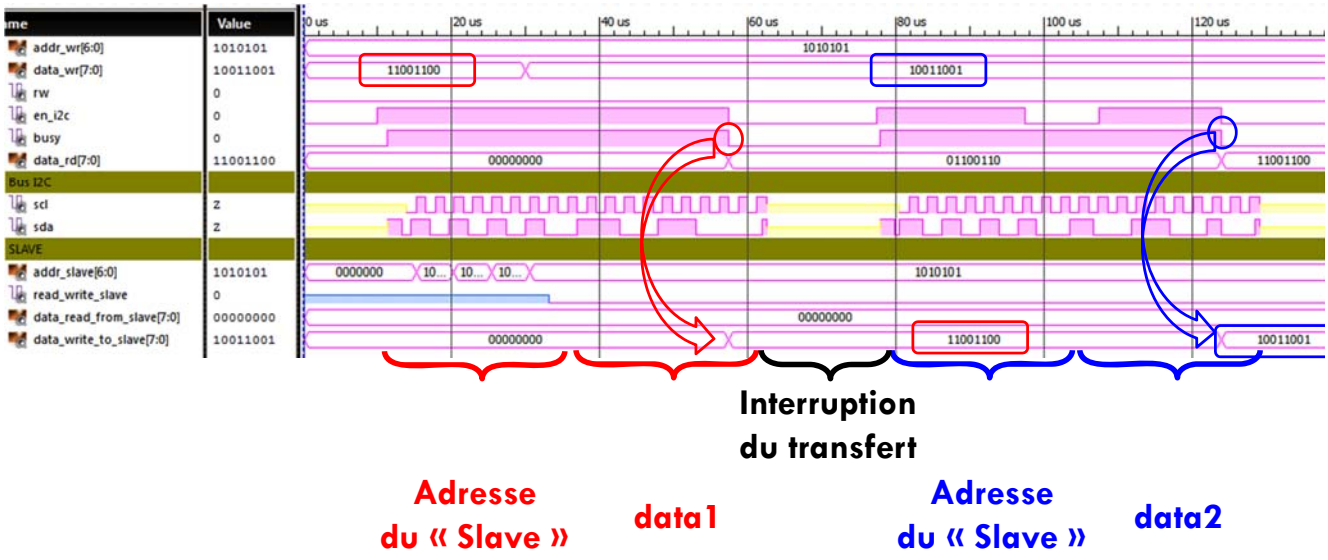
- Simulation complète entre « Master » et « Slave »
  - Ecriture de deux mots au « Slave »
  - Lecture de deux mots provenant du « Slave »

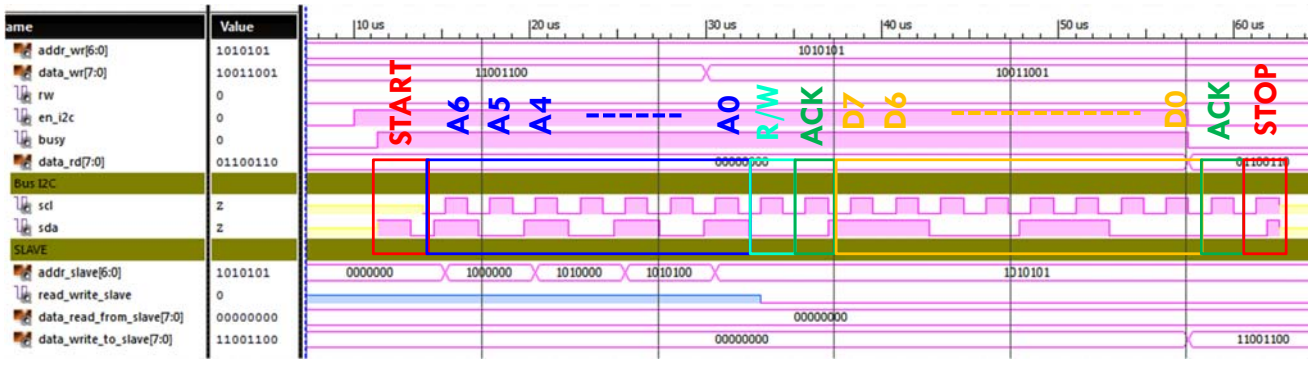


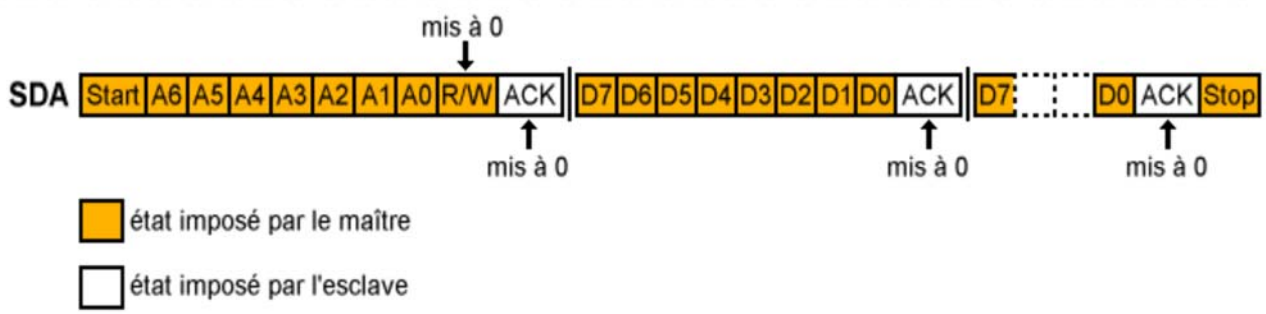
Phase d'écriture vers le « Slave »

Phase de lecture depuis le « Slave »

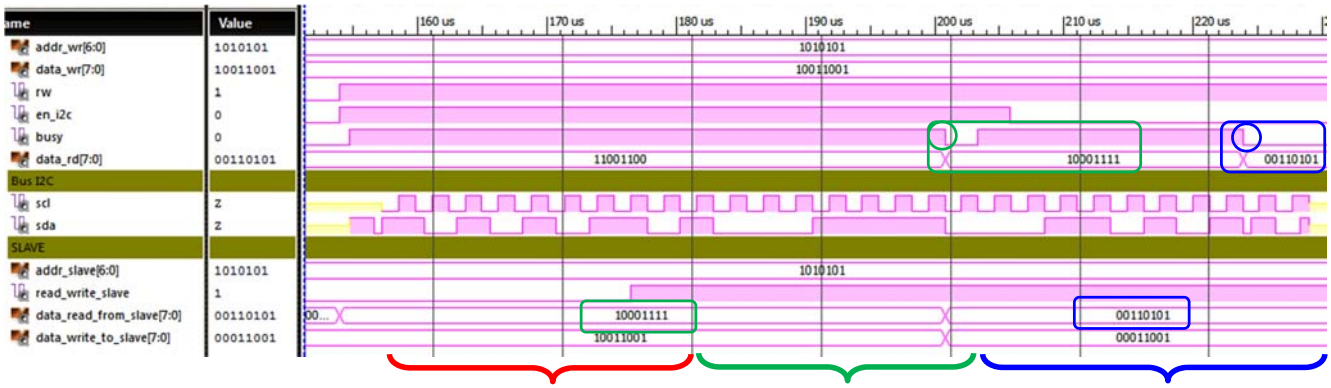
Phase d'écriture vers le « Slave »

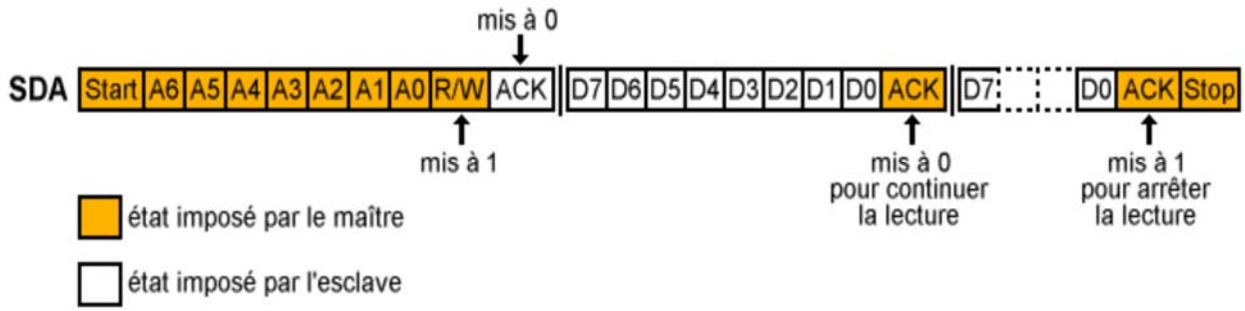




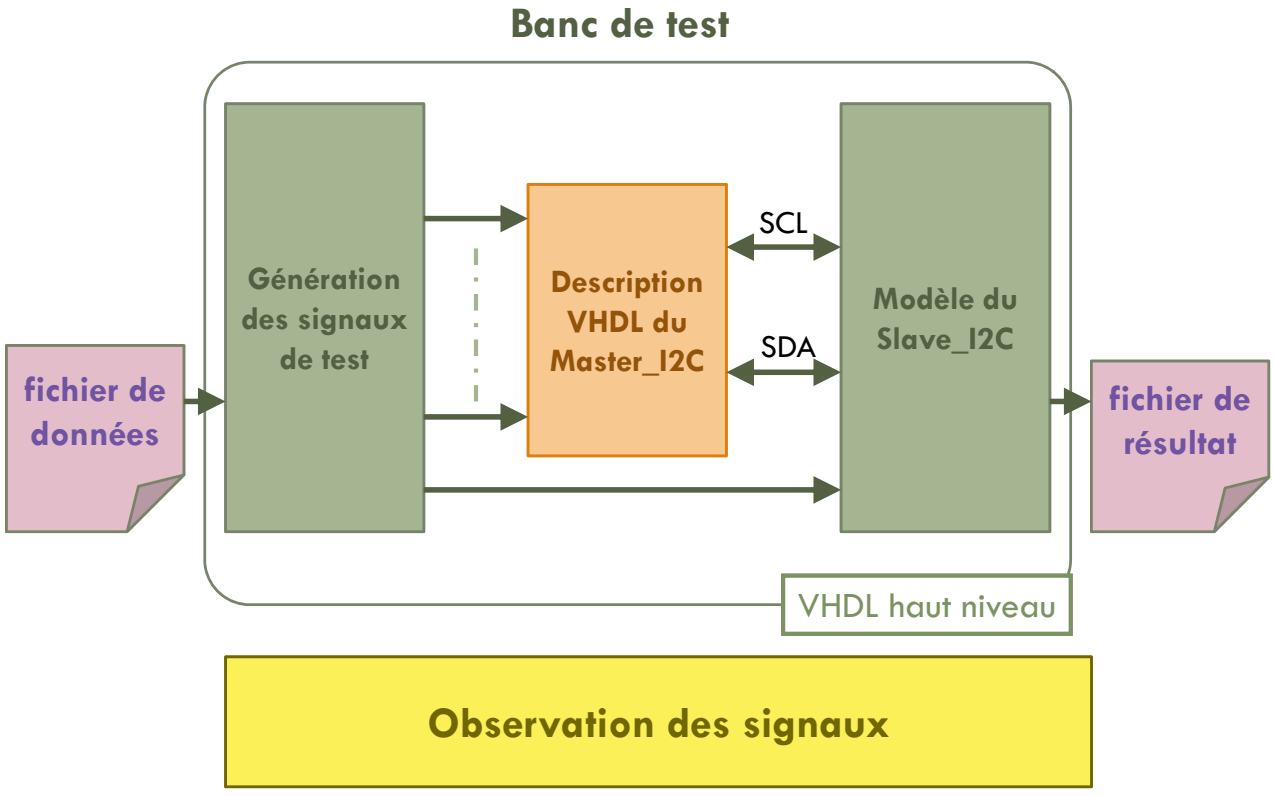


Phase de lecture depuis le « Slave »





Comment faire de telles simulations ?



```
  1  --------------------------------------------------------------------------------
  2  -- Company:    Université Aix-Marseille, IUT Aix-Marseille, Dept Geii
  3  -- Engineer:   Fabrice Aubépart
  4  --
  5  -- Create Date:   10:59:27 01/30/2014
  6  -- Design Name:   masterI2C
  7  -- Module Name:   A:/projets/projets_iut/tpI2C/projetI2C/test_fsm_master.vhd
  8  -- Project Name:  projetI2C
  9  -- Target Device:  Spartan6
 10  -- Tool versions:  ISE 14.3
 11  -- Description:   Testbench pour valider la machine à états finis du master_I2C
 12  --                Voir résultats de simulation dans document "projet MasterI2C.ppt"
 13  --
 14  -- VHDL Test Bench Created by ISE for module: fsm_master
 15  --
 16  -- Dependencies:
 17  --
 18  -- Revision:
 19  -- Revision 0.01 - File Created
 20  -- Additional Comments:
 21  --
 22  -- Notes:
 23  -- This testbench has been automatically generated using types std_logic and
 24  -- std_logic_vector for the ports of the unit under test.  Xilinx recommends
 25  -- that these types always be used for the top-level I/O of a design in order
 26  -- to guarantee that the testbench will bind correctly to the post-implementation
 27  -- simulation model.
 28  --------------------------------------------------------------------------------
 29  LIBRARY ieee;
 30  USE ieee.std_logic_1164.ALL;
 31  use IEEE.STD_LOGIC_ARITH.ALL;
 32  use IEEE.STD_LOGIC_UNSIGNED.ALL;
 33
 34  ENTITY test_fsm_master IS
 35  END test_fsm_master;
 36
 37  ARCHITECTURE behavior OF test_fsm_master IS
 38
 39      -- Component Declaration for the Unit Under Test (UUT)
 40
 41      COMPONENT fsm_master
 42      PORT(
 43           clk : IN  std_logic;
 44           reset : IN  std_logic;
 45           rw : IN  std_logic;
 46           en_I2C : IN  std_logic;
 47           sda : IN  std_logic;
 48           bit_count : IN  std_logic_vector(3 downto 0);
 49           en_count : OUT  std_logic;
 50           sr_count : OUT  std_logic;
 51           sel_sda : OUT  std_logic_vector(1 downto 0);
 52           ld_addr : OUT  std_logic;
 53           ld_data_wr : OUT  std_logic;
 54           sel_shiftReg : OUT  std_logic_vector(1 downto 0);
 55           en_scl : OUT  std_logic;
 56           sel_dataIn : OUT  std_logic;
 57           busy : OUT  std_logic;
```

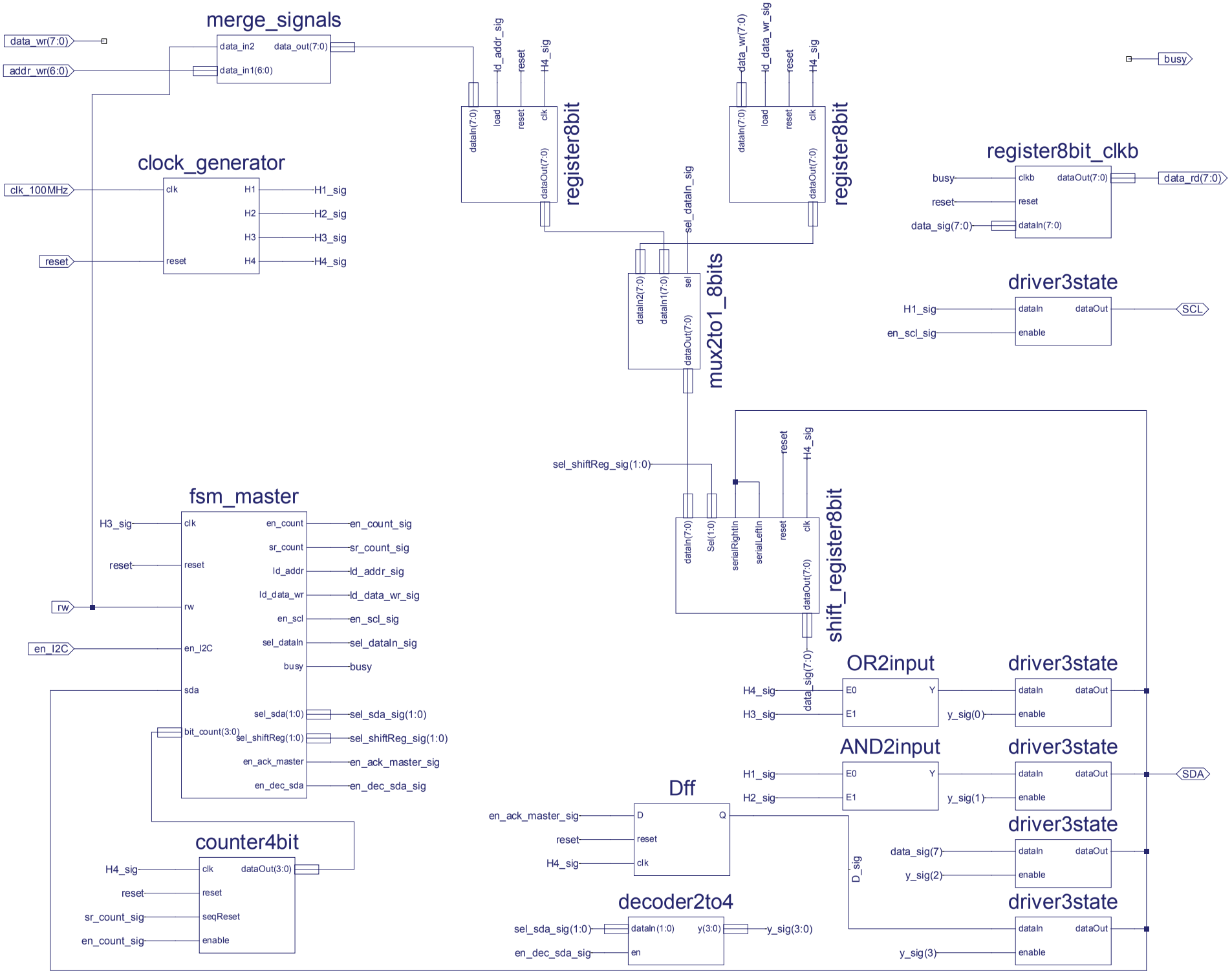


```
 58           en_ack_master : OUT  std_logic;
 59           en_dec_sda : OUT  std_logic
 60          );
 61      END COMPONENT;
 62
 63
 64      --Inputs
 65      signal clk : std_logic := '0';
 66      signal reset : std_logic := '1';
 67      signal rw : std_logic := '0';
 68      signal en_I2C : std_logic := '0';
 69      signal sda : std_logic := '0';
 70      signal bit_count : std_logic_vector(3 downto 0) := (others => '0');
 71
 72      --Outputs
 73      signal en_count : std_logic;
 74      signal sr_count : std_logic;
 75      signal sel_sda : std_logic_vector(1 downto 0);
 76      signal ld_addr : std_logic;
 77      signal ld_data_wr : std_logic;
 78      signal sel_shiftReg : std_logic_vector(1 downto 0);
 79      signal en_scl : std_logic;
 80      signal sel_dataIn : std_logic;
 81      signal busy : std_logic;
 82      signal en_ack_master : std_logic;
 83      signal en_dec_sda : std_logic;
 84
 85      -- Clock period definitions
 86      constant clk_period : time := 2.5 us;
 87
 88  BEGIN
 89
 90      -- Instantiate the Unit Under Test (UUT)
 91      uut: fsm_master PORT MAP (
 92             clk => clk,
 93             reset => reset,
 94             rw => rw,
 95             en_I2C => en_I2C,
 96             sda => sda,
 97             bit_count => bit_count,
 98             en_count => en_count,
 99             sr_count => sr_count,
100             sel_sda => sel_sda,
101             ld_addr => ld_addr,
102             ld_data_wr => ld_data_wr,
103             sel_shiftReg => sel_shiftReg,
104             en_scl => en_scl,
105             sel_dataIn => sel_dataIn,
106             busy => busy,
107             en_ack_master => en_ack_master,
108             en_dec_sda => en_dec_sda
109           );
110
111      -- Clock process definitions
112      clk_process :process
113      begin
114         clk <= '0';
```

```
115         wait for clk_period/2;
116         clk <= '1';
117         wait for clk_period/2;
118      end process;
119
120
121      -- Stimulus process
122      stim_proc: process
123      begin
124         -- Mise à '1' du reset à t=0s
125         -- jusqu'au premier front descendant de clk
126         reset <= '1';
127         wait until (falling_edge(clk));
128         reset <= '0';
129         --
130         -- Attente de 2x2.5us = 5 us
131         wait for clk_period*2;
132         -- Mise à '1' du signal de validation
133         en_I2C <= '1' ;
134         --
135         -- La FSM se met en marche :
136         -- "ready" => "start" => "write_addr1" => "write_addr2"
137         -- elle va reseter et enclencher le compteur
138         --on attend alors que le cpt donne la valeur "1000"
139         wait until (bit_count = "1000");
140         -- "write_addr2" => "slave_ack1"
141         -- On change les états de certaines entrées pour valider
142         -- les états "write_data1" à "slave_ack2" : écriture d'une donnée
143         SDA <= '0';
144         en_I2C <= '1';
145         rw <= '0';
146         --"slave_ack1" => "write_data1" => "write_data2"
147         --on attend de nouveau que le cpt donne la valeur "1000"
148         wait until (bit_count = "1000");
149         -- "write_data2" => "slave_ack2"
150         -- teset du retour de "slave_ack2" => "write_data1"
151         -- "write_data1" => "write_data2"
152         -- on attend une seconde fois que le cpt donne la valeur "1000"
153         wait until (bit_count = "1000");
154         -- "write_data2" => "slave_ack2"
155         -- On change les états de certaines entrées pour valider
156         -- le retour "slave_ack2" => "start"
157         en_I2C <= '1';
158         SDA <= '0';
159         rw <= '1';
160         -- "start" => "write_addr1" => "write_addr2"
161         -- FSM resete et enclenche le compteur
162         -- attende que le cpt donne la valeur "1000"
163         wait until (bit_count = "1000");
164         -- On change les états de certaines entrées pour valider
165         -- "slave_ack1" => "stop"
166         SDA <= '1';
167         -- "stop" => "ready" => etc...
168         ------------------------------------------
169         -- A VOUS ECRIRE LA SUITE !
170         ------------------------------------------
171         wait;
```



```
172      end process;
173
174
175      -- Modele de simulation de counter4bit
176      -- Attention : processus pas synthetisable !!!!!
177      counter4bit_process :process
178      begin
179         wait until (sr_count = '1');
180         wait until (falling_edge(clk));
181         bit_count <= "0000";
182         wait until (en_count = '1');
183         for i in 1 to 8 loop
184            wait until (falling_edge(clk));
185            bit_count <= bit_count + 1;
186            report "compteur incrémenté";
187         end loop;
188      end process;
189
190
191   END;
192
```


merge_signals
data_wr(7:0)
addr_wr(6:0)
data_in2
data_in1(6:0)
data_out(7:0)
ld_addr_sig
reset
H4_sig
dataIn(7:0)
load
clk
dataOut(7:0)
register8bit
ld_data_wr_sig
busy
clock_generator
clk_100MHz
H1
H2
H3
H4
H1_sig
H2_sig
H3_sig
H4_sig
register8bit_clkb
clkb
data_rd(7:0)
data_sig(7:0)
sel_dataIn_sig
mux2to1_8bits
dataIn2(7:0)
dataIn1(7:0)
sel
driver3state
dataIn
dataOut
enable
SCL
en_scl_sig
fsm_master
H3_sig
en_count
sr_count
ld_addr
ld_data_wr
en_scl
sel_dataIn
en_count_sig
sr_count_sig
ld_addr_sig
ld_data_wr_sig
en_scl_sig
sel_dataIn_sig
rw
en_I2C
sda
busy
sel_sda(1:0)
bit_count(3:0)
sel_shiftReg(1:0)
en_ack_master
en_dec_sda
sel_sda_sig(1:0)
sel_shiftReg_sig(1:0)
en_ack_master_sig
en_dec_sda_sig
sel_shiftReg_sig(1:0)
Sel(1:0)
serialRightIn
serialLeftIn
shift_register8bit
data_sig(7:0)
OR2input
AND2input
E0
E1
Y
H4_sig
H3_sig
y_sig(0)
H1_sig
H2_sig
y_sig(1)
SDA
Dff
D
Q
en_ack_master_sig
D_sig
data_sig(7)
y_sig(2)
counter4bit
dataOut(3:0)
seqReset
decoder2to4
dataIn(1:0)
y(3:0)
sel_sda_sig(1:0)
en_dec_sda_sig
y_sig(3:0)
y_sig(3)

```
-- Vhdl test bench created from schematic
A:\projets\projets_iut\tpI2C\projetI2C\masterI2C.sch - Wed Jan 22 09:42:04 2014
--
-- Author : Fabrice AUBEPART
-- Université Aix-Marseille, IUT Aix-Marseille, Dept Geii
--
-- Version : 4.0
-- Date : 28 Janvier 2014

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
LIBRARY UNISIM;
USE UNISIM.Vcomponents.ALL;
ENTITY masterI2C_masterI2C_sch_tb IS
END masterI2C_masterI2C_sch_tb;
ARCHITECTURE behavioral OF masterI2C_masterI2C_sch_tb IS

   COMPONENT masterI2C
   PORT( rw :  IN STD_LOGIC;
         en_I2C  :  IN STD_LOGIC;
         SDA  :  INOUT STD_LOGIC;
         clk_100MHz :  IN STD_LOGIC;
         reset  :  IN STD_LOGIC;
         data_rd :  OUT  STD_LOGIC_VECTOR (7 DOWNTO 0);
         data_wr :  IN STD_LOGIC_VECTOR (7 DOWNTO 0);
         SCL  :  INOUT STD_LOGIC;
         busy :  OUT  STD_LOGIC;
         addr_wr :  IN STD_LOGIC_VECTOR (6 DOWNTO 0));
   END COMPONENT;

   SIGNAL rw  :  STD_LOGIC;
   SIGNAL en_I2C  :  STD_LOGIC;
   SIGNAL SDA  :  STD_LOGIC:= 'Z';
   SIGNAL clk_100MHz :  STD_LOGIC := '0';
   SIGNAL reset  :  STD_LOGIC;
   SIGNAL data_rd :  STD_LOGIC_VECTOR (7 DOWNTO 0);
   SIGNAL data_wr :  STD_LOGIC_VECTOR (7 DOWNTO 0);
   SIGNAL SCL  :  STD_LOGIC;
   SIGNAL busy :  STD_LOGIC;
   SIGNAL addr_wr :  STD_LOGIC_VECTOR (6 DOWNTO 0);

   -- Signaux pour testbench
   signal addr_slave : std_logic_vector(6 downto 0):="0000000";
   signal read_write_slave : std_logic;
   signal data_read_from_slave : std_logic_vector(7 downto 0):="00000000";
   signal data_write_to_slave : std_logic_vector(7 downto 0):="00000000";
   signal data_write : std_logic_vector(7 downto 0) := (others => '0');

BEGIN

   UUT: masterI2C PORT MAP(
      rw => rw,
      en_I2C => en_I2C,
      SDA => SDA,
      clk_100MHz => clk_100MHz,
      reset => reset,
```



```
      data_rd => data_rd,
      data_wr => data_wr,
      SCL => SCL,
      busy => busy,
      addr_wr => addr_wr
   );

-------------------------------------------------------
-- Horloge et reset
-------------------------------------------------------
-- Horloge H du master est de 100MHz (10 ns): horloge de la carte Nexys3
-- Horloge SCL de la ligne I2C est fixé ici à 400kHz (2.5 us)
clk_100MHz <= not clk_100MHz after 5 ns;
reset <= '1', '0' after 152 ns;

-- Processus pour signaux de démarrage
process
begin
   -- ecriture de données
   addr_wr <= "1010101";
   data_wr <= "11001100";
   rw <= '0';
   en_I2C <= '0';
   wait for 10 us;
   en_I2C <= '1';
   wait for 20 us;
   data_wr <= "10011001";
   wait until (busy = '0');
   en_I2C <= '0';
   wait for 20 us;
   en_I2C <= '1';
   wait for 20 us;
   en_I2C <= '0';
   wait for 10 us;
   en_I2C <= '1';
   wait until (busy = '0');
   en_I2C <= '0';
   wait for 30 us;
-- lecture de données
   en_I2C <= '1';
   rw <= '1';
   --addr_wr <= "1001001";
   data_read_from_slave <= "10001111";
   wait for 10 us;
   en_I2C <= '1';
   wait until (busy = '0');
   data_read_from_slave <= "00110101";
   wait for 5 us;
   en_I2C <= '0';
   wait;
end process;

-- Mise à jour du signal écrit au slave uniquement sur front descendant du busy
data_write_to_slave <= data_write when (falling_edge(busy));

-- Processus de gestion des signaux en écriture vers slave
-- ou en lecture depuis slave
```

```
114   process
115   variable i : natural;
116   begin
117
118      -- Attente du START
119      wait until (falling_edge(SDA) and SCL = 'Z');
120      report "start ok";
121      -- Boucle pour récupérer les 7 premiers bits de l'adresse
122      for i in 6 downto 0 loop
123         wait until (rising_edge(SCL));
124         -- Signal affectée utilisée dans ce type de processus !
125         -- affectation d'une valeur à un signal ne prend effet que lorsque
126         -- le processus se remet en attente.
127         addr_slave(i) <= SDA;
128         report "Adresse modifie";
129      end loop;
130      wait until (rising_edge(SCL));
131      read_write_slave <= SDA;
132      -- le prochain front descendant de SCL sera utilisée pour mettre SDA = '0' (ACQ)
133      -- si addr_sig = "1010101", sinon SDA = 'Z' (slave non concerné !)
134      wait until (falling_edge(SCL));
135      if (addr_slave /= "1010101") then
136         SDA <= 'Z';
137         report "Slave non concerné";
138      else
139         -- Mise à '0' de ACK
140         SDA <= '0';
141
142         -- Mode écriture dans SLAVE
143         if (read_write_slave = '0') then
144            -- Au prochain front descendant de SCL on remet SDA = 'Z'
145            wait until (falling_edge(SCL));
146            SDA <= 'Z';
147            report "debut écriture dans le SLAVE des données";
148            etiq1 : loop
149               for i in 7 downto 0 loop
150                  wait until (rising_edge(SCL));
151                  data_write(i) <= SDA;
152                  -- test si STOP (ou STOP) pas envoyé (peut-être envoyé n'importe quand
      d'après la norme)
153                  -- le temps d'attente ici est fixé < à 1/2 période de SCL, mais
      suffisant grand
154                  -- Pour que SDA soit passer à ... '1' dans le cas du STOP / ou '0'
      dans cas du START
155                  wait for 1 us;
156                  exit etiq1 when (data_write(i) /= SDA);
157               end loop;
158               -- ACQ est is à '0'
159               wait until (falling_edge(SCL));
160               SDA <= '0';
161               -- ligne SDA libéré pour envoi prochaine donnée
162               wait until (falling_edge(SCL));
163               SDA <= 'Z';
164               report "fin écriture data depuis SLAVE";
165            end loop etiq1;
166
167         --Mode lecture depuis SLAVE
```

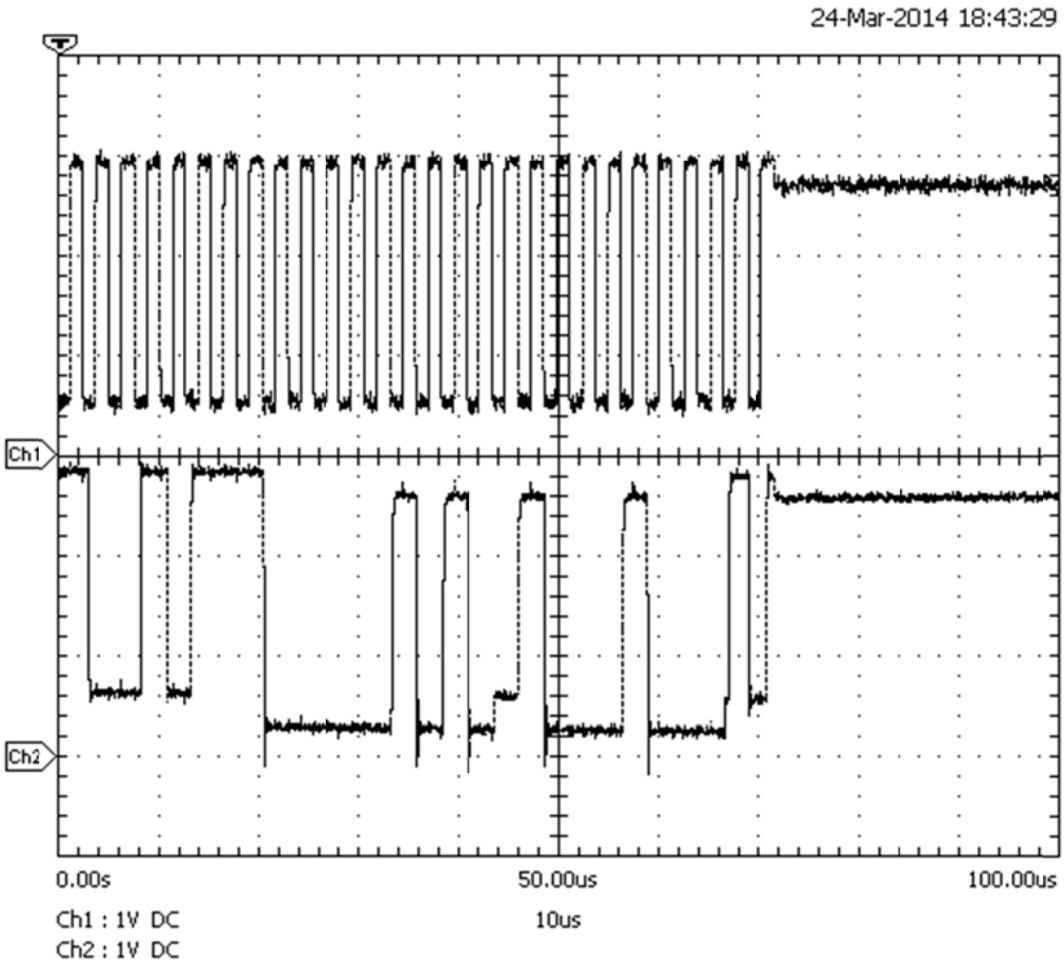


```
168         else
169            report "debut lecture depuis le SLAVE des données";
170            etiq2 : loop
171               for i in 7 downto 0 loop
172                  wait until (falling_edge(SCL));
173                  SDA <= data_read_from_slave(i);
174                  -- test si STOP (ou STOP) pas envoyé (peut-être envoyé n'importe quand
      d'après la norme)
175                  -- le temps d'attente ici est fixé < à 1/2 période de SCL, mais
      suffisant grand
176                  -- Pour que SDA soit passer à ... '1' dans le cas du STOP / ou '0'
      dans cas du START
177                  wait for 1 us;
178                  exit etiq2 when (data_read_from_slave(i) /= SDA);
179               end loop;
180               -- ACQ est mis à 'Z' par le SLAVE dans l'attente d'un ACQ de la part du
      MASTER
181               wait until (falling_edge(SCL));
182               SDA <= 'Z';
183               -- ligne SDA testé pour savoir si ACQ = '0' par le MAster
184               -- si ACQ = '1' on attend un stop
185               -- sinon on reboucle pour envoi d'une nouvelle donnée
186               wait until (rising_edge(SCL));
187               exit etiq2 when (SDA = '1');
188               report "fin lecture data depuis SLAVE";
189            end loop etiq2;
190               report "sortie du mode lecture depuis slave";
191            -- ligne SDA libéré pour envoi prochaine donnée
192               wait until (falling_edge(SCL));
193               SDA <= 'Z';
194
195         end if;
196      end if;
197   end process;
198
199   END;
200
```

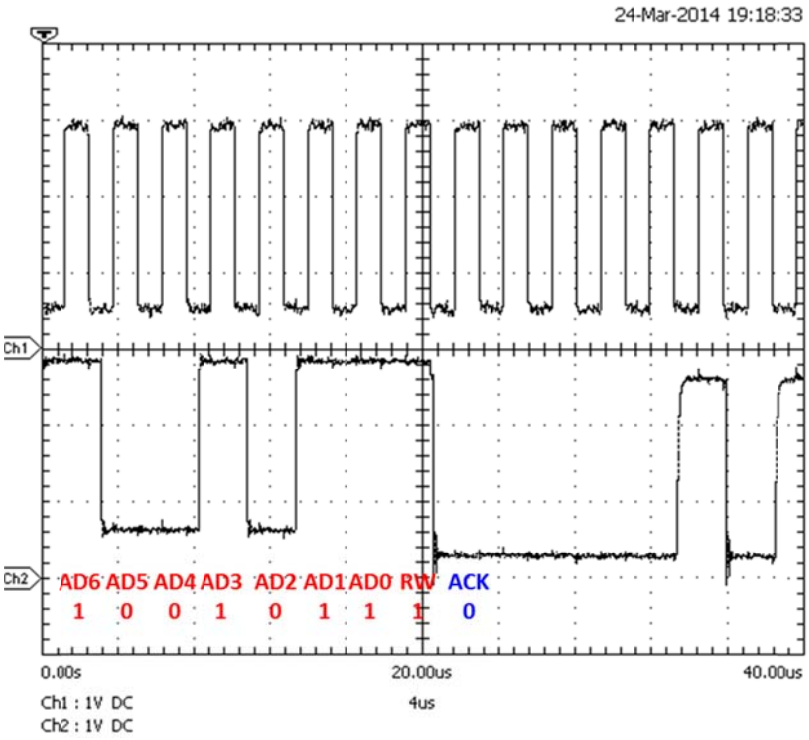
Résultats capteur de température (module Pmod TMP2)

Observation de la TRAME I2C envoyée en mode lecture sur capteur de température et réception de 2 octets (information 16 bits correspondant à la température).

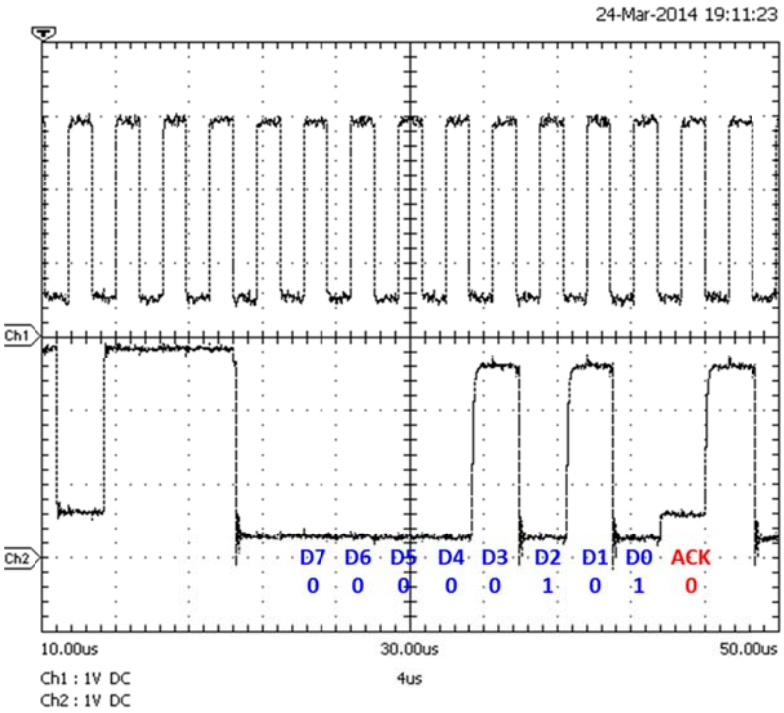
Remarque : La liaison a été améliorée en mettant des résistances externes de tirage vers Vdd (3,3 V) de 1 kΩ (les résistances de Pull-Up internes au FPGA sont insuffisantes : valeurs trop grandes ?)



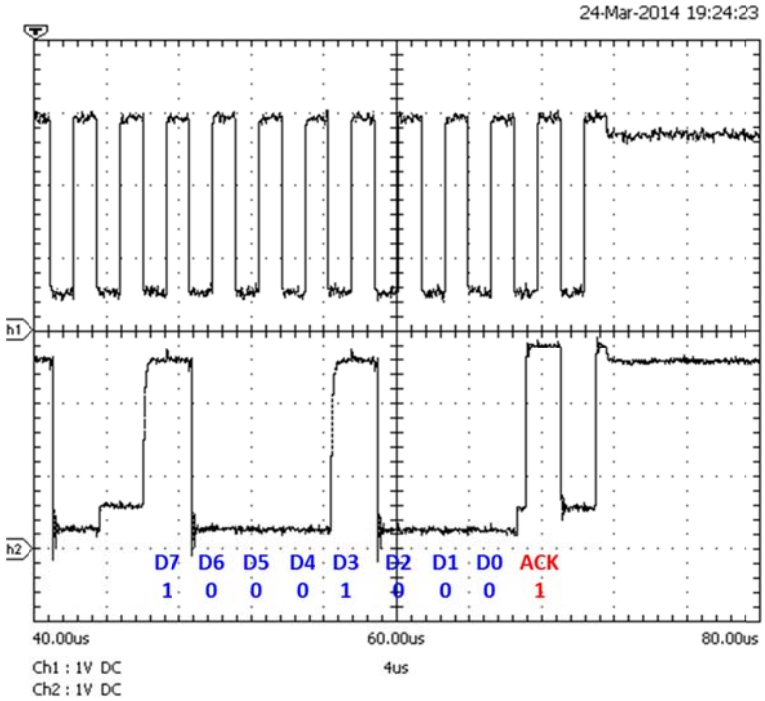
Envoi de l'adresse au capteur + RW, suivi de la réponse du module par ACK à '0'



Envoi par le module du premier octet (8 bits de poids forts) de la température qui est sur 16 bits au total, suivi de la réponse du master I2C par ACK à '0'

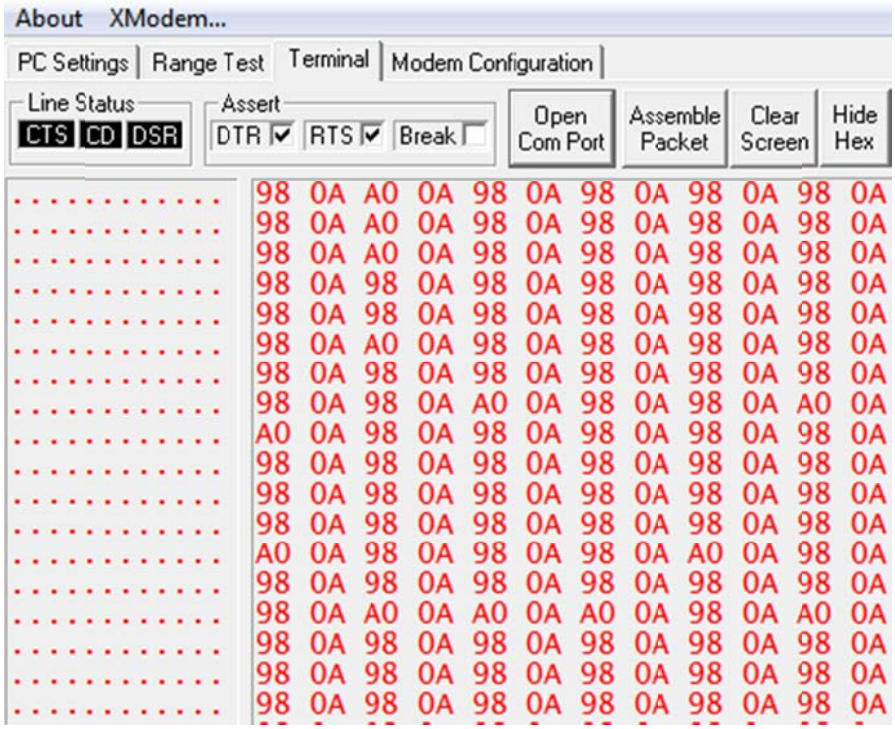


Envoi par le module du second octet (8 bits de poids faibles) de la température qui est sur 16 bits au total, suivi de la réponse du master I2C par ACK à '1', qui va forcer ensuite à une stop :

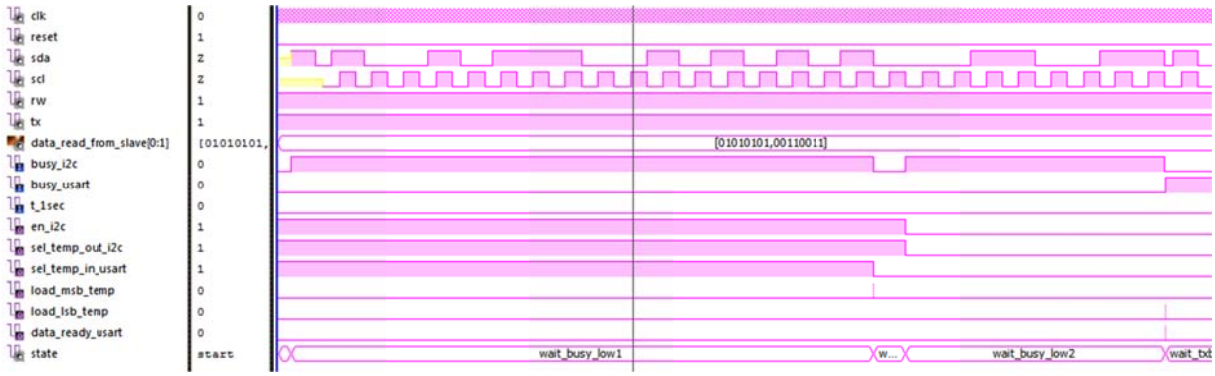


La Température lue est donc en binaire : 0000 0101 1000 1000 soit en hexadécimal : 05 88

Observation sur un X-CTU permettant d'observer les valeurs en hexadécimal. *Attention, ne pas observer les caractères alphanumériques qui ne correspondent à rien dans ce cas.*



Résultats de simulation : lecture de deux octets sur SDA (Adresse du module + octet MSB puis octet LSB de la température). *Pour plus d'informations voir travail antérieur sur simulations Master_I2C.*



Résultats de simulation : envoi des deux octets sur la ligne Tx (octet LSB puis octet MSB de la température)