Projet d'électronique III

Alexandre Carlessi Thomas Denoréaz Johan Berdat

13 juin 2013

Table des matières

1 Introduction

L'objectif du projet est de développer un ordinateur de bord pour une voiture. Cela permet d'avoir une vision d'ensemble de la conception d'un système électronique.

L'automobile possède un certains nombre de capteurs (tachymètre, niveau d'essence...) qui relaient leur informations analogique jusqu'à l'unité de traitement. Le signal est échantillonné et stocké en mémoire, pour être analysé par l'unité de contrôle. Cette dernière prend les décisions qui s'imposent (avertissement et régulations) et envoie les informations à l'interface utilisateur.

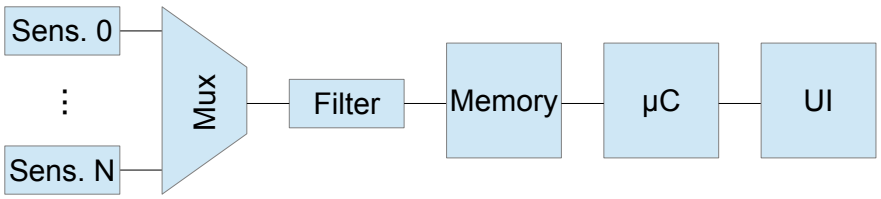

Fig. 1 – Schema bloc du système

2 Composants analogiques et échantillonage

On peut citer beaucoup grandeurs à mesurer dans une voiture.
- Tachymètres, pour mesurer la vitesse des roues et du moteur.
- Thermomètres, pour mesurer la température extérieure et du moteur.
- Baromètres, pour la pression de l'huile et autres fluides.

Des amplificateurs sont utilisés sur les capteurs pour permettre au signal analogique de parcourir les grandes distances qui les séparent du système central.

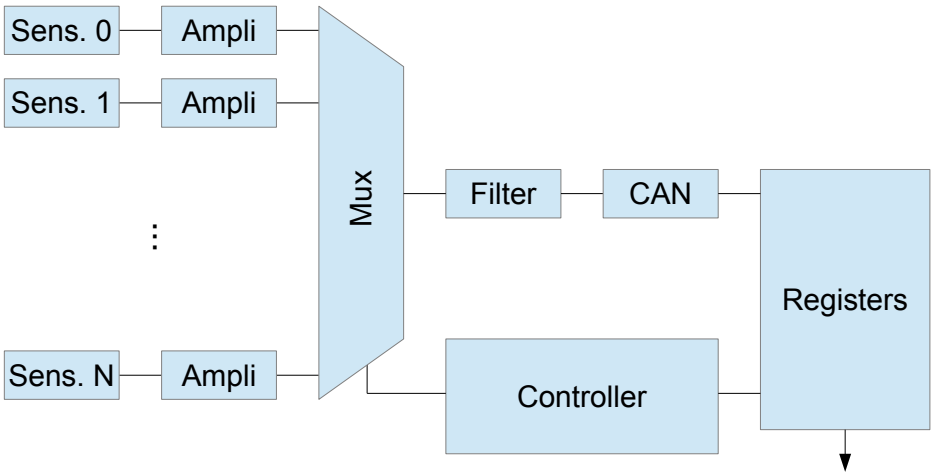

FIG. 2 – Schéma bloc du système de mesure

Pour économiser le nombre de composants utilisés, on n'utilise qu'un seul convertisseur analogique-numérique, après un multiplexeur. Un contrôleur s'occupe d'échantillonner successivement les différents capteurs, pour mettre à jour les registres qui leur correspondent.

Ces informations ne requièrent pas une précision élevée. Nous avons donc choisi une fréquence d'échantillonage pour chaque capteur de $10Hz$. Ainsi, si nous avons 5 capteurs, le contrôleur aura une fréquence de $50Hz$.

Après le multiplexeur, un filtre actif s'occupe de nettoyer le signal. On a opté pour un simple filtre passe-bas, afin de filtrer les fréquences supérieures à $10Hz$. Il serait bien sûr possible de cascader ce filtre pour améliorer la qualité (et éventuellement considérer d'autres types de filtres, tel le Sallen-Key de Butterworth d'ordre N).

L'étape suivante consiste à convertir le signal analogique en signal numérique. Pour cela, on a à disposition plusieurs sortes de convertisseurs. Nous avons choisi la famille de convertisseurs A/N à approximations successives, étant adaptés pour un usage général. En effet, ils représentent un bon compromis précision, vitesse, prix.

Le seul désavantage de ce genre de convertisseur est le nombre limité de bits disponibles (8 à 14 bits). Pour corriger ce problème, il est possible de monter ces CAN en cascade, et ainsi augmenter la résolution. Nous estimons que 16 bits sont suffisants pour les grandeurs que nous utilisons.

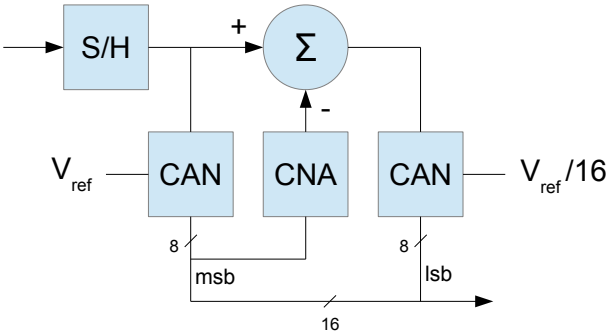

FIG. 3 – Convertisseur analogique-digital 16-bits

Enfin, ce signal numérique est mémorisé dans des registres, qui pourront être lus par l'unité principale.

3 Unités de contrôle et de traitement

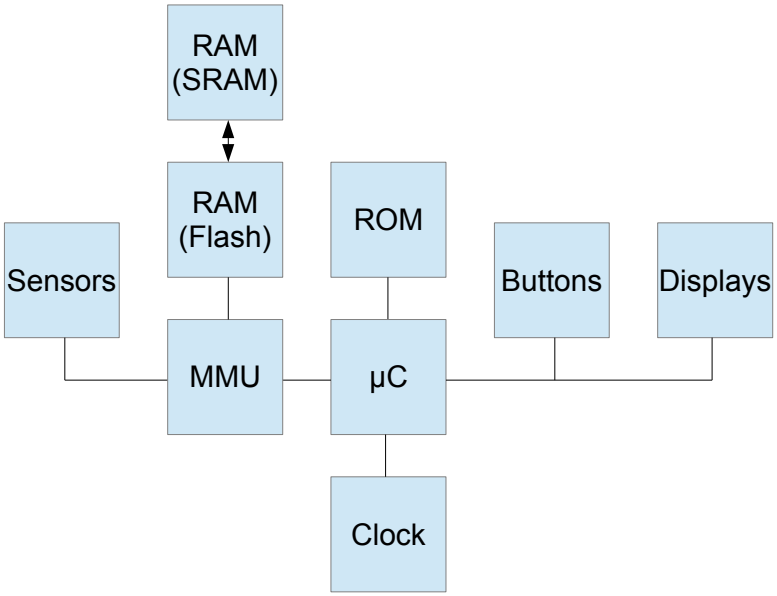

Fig. 4 – Schéma bloc du système

L'unité de contrôle (UC) est reliée à tous les autres composants, et se charge de coordonner les tâches.

Du côté de la mémoire, les différents registres des capteurs sont accessibles en lecture seule. Une mémoire SRAM nous paraît être un bon choix, rapide et robuste. Au vu de la taille minime nécessaire, son coût est acceptable.

À cela s'ajoute une autre mémoire vive qui contient les données nécessaires au système. De nouveau, nous utilisons une SRAM pour conserver les valeurs (compteur kilométrique, par exemple) lorsque le système est sous tension. Et lorsque le système est privé d'alimentation, une copie de la mémoire est sauvegardée sur une mémoire non-volatile, de type Flash. Cette décision est motivée par le nombre limité d'écriture sur les supports Flash. En effet, avec une fréquence d'écriture de $10Hz$, la mémoire Flash atteindrait rapidement ses limites de réécriture.

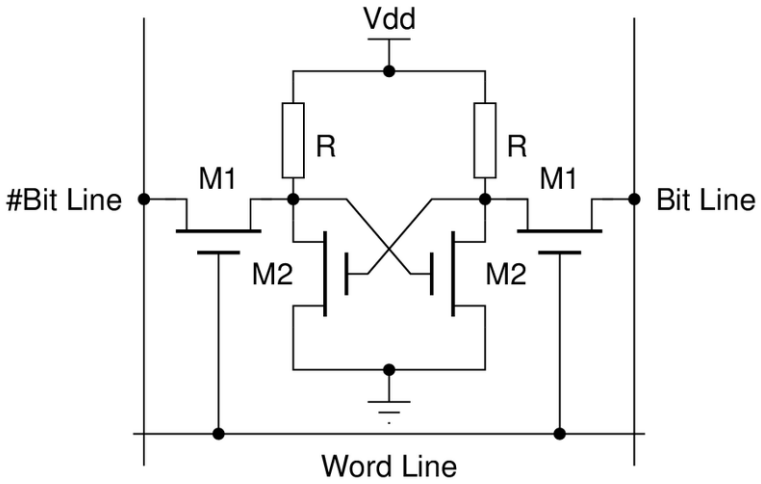

Fig. 5 – SRAM (source Wikipédia)

3.1 Clock

La gestion de l'heure a été faite à l'aide du circuit décrit par le code VHDL suivant, celui-ci retourne un *timestamp* de l'heure actuelle, ce module peut être configuré depuis l'ECU.

```
1  -- ------------------------------------------------------------------------
2  -- ---- clock.vhd
3  -- ------------------------------------------------------------------------
4  -- ---- Project   : BA6 - Elec III
5  -- ---- Authors   :
6  -- ----    (183785) Thomas Denoréaz
7  -- ----    (204393) Johan Berdat
8  -- ----    (194875) Alexandre Carlessi
9  -- ----
10 -- ---- Versions :
11 -- ----    - 2013.03.19 - Initial version
12 -- ------------------------------------------------------------------------
13
14 library ieee;
15 use ieee.std_logic_1164.all;
16 use ieee.numeric_std.all;
17
18 use work.utils.all;
19
20 entity ecu_clock is
21   port(
22     clk        : in  ubit;
23     rst        : in  ubit;
24
25     cs         : in  ubit;
26     load       : in  ubit;
27     value      : in  udword;
28
29     timestamp  : out udword
30   );
31 end entity ecu_clock;
32
33 architecture RTL of ecu_clock is
34   signal timestamp_intern : udword;
35 begin
36   timestamp <= timestamp_intern;
37
38   process(clk, rst) is
39   begin
40     if rst = '1' then
41       timestamp_intern <= (others => '0');
42
43     elsif rising_edge(clk) then
44       timestamp_intern <= timestamp_intern + 1;
45       if is_set(cs) and is_set(load) then
46         timestamp_intern <= value;
47       end if;
48     end if;
49   end process;
50
51 end architecture RTL;
```

Listing 1 – Module de contrôle de l'horloge

3.2 ECU (Electronic Control Unit)

L'idée est de contrôler tous les modules ainsi que les capteurs dans l'unité de contrôle électronique.

Voici le code VHDL permettant de décrire ce circuit, et ainsi de le simuler :

```
1  -- ------------------------------------------------------------------------
2  -- ---- ecu.vhd
3  -- ------------------------------------------------------------------------
4  -- ---- Project   : BA6 - Elec III
5  -- ---- Authors   :
6  -- ----    (183785) Thomas Denoréaz
7  -- ----    (204393) Johan Berdat
8  -- ----    (194875) Alexandre Carlessi
9  -- ----
10 -- ---- Versions :
11 -- ----    - 2013.03.19 - Initial version
12 -- ----    - 2013.06.01 - Measure system
13 -- ------------------------------------------------------------------------
14
15 library ieee;
16 use ieee.std_logic_1164.all;
17 use ieee.numeric_std.all;
18
19 use work.utils.all;
20
21 -- Electronic Control Unit
22 entity ecu is
23   port(
24     clk         : in  ubit;
25     rst         : in  ubit;
26
27     -- MMU
28     mmu_rw      : out ubit;
29     mmu_addr    : out ubits(3 downto 0);
30     mmu_in      : in  udword;
31     mmu_out     : out udword;
32
33     -- HCI
34     op          : in  ubits(3 downto 0);
35     opx         : in  ubits(3 downto 0);
36
37     velocity    : out udword;
38     engine      : out udword;
39     distance    : out udword;
40     timestamp   : out udword
41   );
42 end entity ecu;
43
44 architecture RTL of ecu is
45   constant CONST_WAIT_TICK : uword := X"0000";
46
47   constant ECU_CLOCK   : ubits(3 downto 0) := X"0";
48   constant ECU_CHRONO  : ubits(3 downto 0) := X"1";
49
50   constant ADDR_VELOCITY   : ubits(3 downto 0) := X"0";
51   constant ADDR_ENGINE     : ubits(3 downto 0) := X"1";
52   constant ADDR_TIMESTAMP  : ubits(3 downto 0) := X"8";
53   constant ADDR_DISTANCE   : ubits(3 downto 0) := X"9";
```

```
54
55    signal clk_ms : ubit;
56
57    signal clock_cs        : ubit;
58    signal clock_timestamp : udword;
59
60    signal chrono_cs        : ubit;
61    signal chrono_timestamp : udword;
62
63    type state_t is (INIT, LOAD, TIMER, FETCH, COMPUTE);
64    signal state : state_t;
65
66    signal counter : uword;
67
68    type sensor_t is (S_FIRST, S_VELOCITY, S_ENGINE, S_LAST);
69    signal sensor : sensor_t;
70
71    signal int_velocity : udword;
72    signal int_engine   : udword;
73
74    type measure_t is (M_FIRST, M_TIMESTAMP, M_DISTANCE, M_LAST);
75    signal measure : measure_t;
76
77    signal delta_t        : udword;
78    signal mem_timestamp : udword;
79    signal mem_distance   : udword;
80
81  begin
82    clk_ms <= clk;
83
84    -- Clock
85    ecu_clock_0 : entity work.ecu_clock(RTL)
86      port map(
87        clk       => clk_ms,
88        rst       => rst,
89        cs        => clock_cs,
90        load      => opx(0),
91        value     => mmu_in,
92        timestamp => clock_timestamp
93      );
94
95    -- Chrono
96    ecu_chrono_0 : entity work.ecu_chrono(RTL)
97      port map(
98        clk        => clk_ms,
99        rst        => rst,
100       cs         => chrono_cs,
101       start_stop => opx(0),
102       clear      => opx(1),
103       timestamp  => chrono_timestamp
104     );
105
106   UI : process(clk, rst) is
107   begin
108     if rst = '1' then
109       clock_cs  <= '0';
110       chrono_cs <= '0';
111       timestamp <= (others => '0');
```

```
112
113       elsif rising_edge(clk) then
114         clock_cs  <= '0';
115         chrono_cs <= '0';
116
117         case op is
118           when ECU_CLOCK =>
119             clock_cs  <= '1';
120             timestamp <= clock_timestamp;
121
122           when ECU_CHRONO =>
123             chrono_cs <= '1';
124             timestamp <= chrono_timestamp;
125
126           when others => null;
127         end case;
128
129       end if;
130     end process UI;
131
132     velocity <= int_velocity;
133     engine   <= int_engine;
134     distance <= mem_distance;
135
136     controller : process(clk, rst) is
137       variable mult : uddword;
138     begin
139       if rst = '1' then
140         mmu_rw   <= '0';
141         mmu_addr <= (others => '0');
142         mmu_out  <= (others => '0');
143
144         counter <= (others => '0');
145
146         int_velocity <= (others => '0');
147         int_engine   <= (others => '0');
148
149         delta_t       <= (others => '0');
150         mem_timestamp <= (others => '0');
151         mem_distance  <= (others => '0');
152
153         state   <= INIT;
154         sensor  <= S_FIRST;
155         measure <= M_FIRST;
156
157       elsif rising_edge(clk) then
158         mmu_addr <= (others => '0');
159         mmu_rw   <= '0';
160
161         counter <= counter + 1;
162
163         case state is
164           when INIT =>
165             counter <= (others => '0');
166             sensor  <= S_FIRST;
167             measure <= M_FIRST;
168             state   <= LOAD;
169
```

```
170             when LOAD =>
171               case measure is
172                 when M_FIRST =>
173                   measure  <= M_TIMESTAMP;
174                   mmu_addr <= ADDR_TIMESTAMP;
175
176                 when M_TIMESTAMP =>
177                   mem_timestamp <= mmu_in;
178
179                   measure  <= M_DISTANCE;
180                   mmu_addr <= ADDR_DISTANCE;
181
182                 when M_DISTANCE =>
183                   mem_distance <= mmu_in;
184
185                   measure <= M_LAST;
186
187                 when M_LAST =>
188                   measure <= M_FIRST;
189                   state   <= TIMER;
190
191               end case;
192
193             when TIMER =>
194               if counter > CONST_WAIT_TICK then -- wait a moment
195                 counter <= (others => '0');
196                 sensor  <= S_FIRST;
197                 measure <= M_FIRST;
198                 state   <= FETCH;
199               end if;
200
201             when FETCH =>
202               case sensor is
203                 when S_FIRST =>
204                   sensor   <= S_VELOCITY;
205                   mmu_addr <= ADDR_VELOCITY;
206                 when S_VELOCITY =>
207                   int_velocity <= mmu_in;
208
209                   sensor   <= S_ENGINE;
210                   mmu_addr <= ADDR_ENGINE;
211
212                 when S_ENGINE =>
213                   int_engine <= mmu_in;
214
215                   sensor <= S_LAST;
216
217                 when S_LAST =>
218                   sensor <= S_FIRST;
219                   state  <= COMPUTE;
220               end case;
221
222             when COMPUTE =>
223               case measure is
224                 when M_FIRST =>
225                   measure <= M_TIMESTAMP;
226
227                 when M_TIMESTAMP =>
```

```
228                 delta_t  <= clock_timestamp - mem_timestamp;
229                 mmu_addr <= ADDR_TIMESTAMP;
230                 mmu_rw   <= '1';
231                 mmu_out  <= clock_timestamp;
232
233                 measure <= M_DISTANCE;
234
235               when M_DISTANCE =>
236                 mult          := delta_t * int_velocity;
237                 mem_distance <= mem_distance + mult(37 downto 6);
238
239                 mmu_addr <= ADDR_DISTANCE;
240                 mmu_rw   <= '1';
241                 mmu_out  <= mem_distance + mult(37 downto 6);
242                 measure  <= M_LAST;
243
244               when M_LAST =>
245                 measure <= M_FIRST;
246                 state   <= TIMER;
247
248             end case;
249         end case;
250       end if;
251   end process controller;
252 end architecture RTL;
```

Listing 2 – Module de gestion du système (UC)

4 Simulation - interface utilisateur

L'interface utilisateur a été implémenté en utilisant en tâche de fond *ModelSim*, qui permet de simuler du code VHDL. L'idée est d'utiliser directement le vrai code VHDL de notre interface en simulant les entrées/sorties du système de mesure.

Le module d'affichage a été développé en Java, en utilisant le framework *Spring*[1] pour simplifier le travail. Le module de simulation est une coopération entre Java et VHDL. En VHDL la simulation est effectuée, alors qu'en Java le traitement des données est fait.

Pour l'utiliser, il faut appuyer sur l'accélérateur et ainsi voir la voiture avancer, et pour l'arrêter appuyer sur le frein.

1. `http://www.springsource.org/`

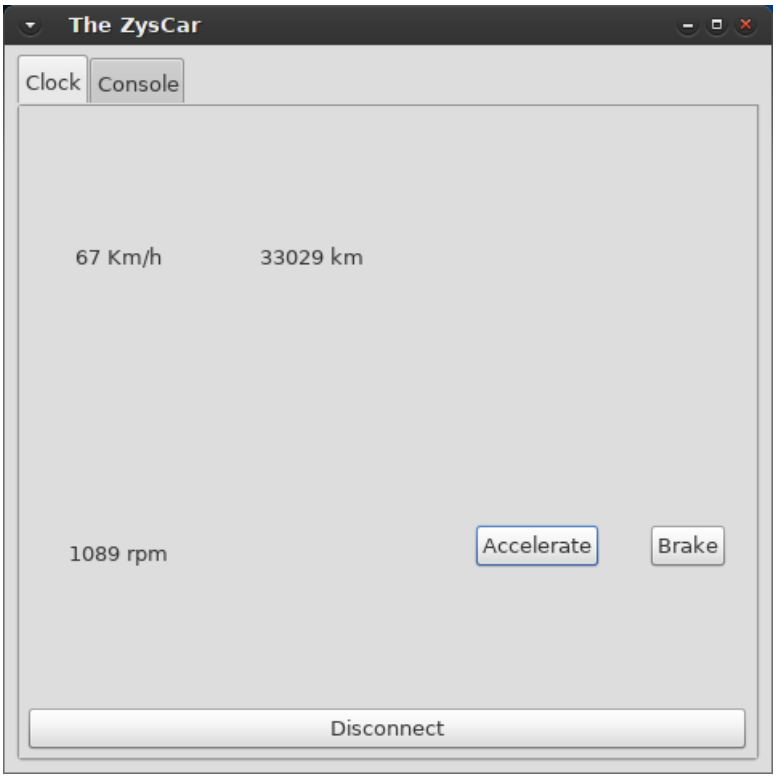

FIG. 6 – The ZysCar

Voici un exemple de listing permettant de contrôler le simulateur.

```
1  -- ------------------------------------------------------------------------
2  -- ---- clock_simulator.vhd
3  -- ------------------------------------------------------------------------
4  -- ---- Project   : LabSimulator - Package Library
5  -- ---- Authors   :
6  -- ----    (183785) Thomas Denoréaz
7  -- ----    (204393) Johan Berdat
8  -- ----    (194875) Alexandre Carlessi
9  -- ----
10 -- ---- Versions :
11 -- ----    - 2013.03.19 - Initial version
12 -- ------------------------------------------------------------------------
13
14 library ieee;
15 use ieee.std_logic_1164.all;
16 use ieee.std_logic_textio.all;
17 use ieee.numeric_std.all;
18 use std.textio.all;
19
20 use work.txt_util.all;
21 use work.vunit.all;
22 use work.utils.all;
23
24 entity ecu_simulator is
25 end;
26
27 architecture simulator of ecu_simulator is
28
29   -- ------ Simulated circuit ------------------------------------------
```

```
30
31    signal clk        : ubit                 := '0';
32    signal rst        : ubit                 := '1';
33    signal mmu_rw     : ubit                 := '0';
34    signal mmu_addr   : ubits(3 downto 0)    := (others => '0');
35    signal mmu_in     : udword               := (others => '0');
36    signal mmu_out    : udword               := (others => '0');
37    signal op         : ubits(3 downto 0)    := (others => '0');
38    signal opx        : ubits(3 downto 0)    := (others => '0');
39    signal velocity   : udword               := (others => '0');
40    signal engine     : udword               := (others => '0');
41    signal distance   : udword               := (others => '0');
42    signal timestamp  : udword               := (others => '0');
43
44    signal sensor_velocity   : udword := (others => '0');
45    signal sensor_engine     : udword := (others => '0');
46    signal sensor_timestamp  : udword := (others => '0');
47
48    -- ----- Simulated value ------------------------------------------------
49    signal sim_accelerate : ubit := '0';
50    signal sim_brake      : ubit := '0';
51
52  begin
53
54    -- ------------------------------------------------------------------------
55    -- ----- Simulated circuit ------------------------------------------------
56    -- ------------------------------------------------------------------------
57
58    -- Electronic Control Unit
59
60    ecu_inst : entity work.ecu
61      port map(clk       => clk,
62               rst       => rst,
63               mmu_rw    => mmu_rw,
64               mmu_addr  => mmu_addr,
65               mmu_in    => mmu_in,
66               mmu_out   => mmu_out,
67               op        => op,
68               opx       => opx,
69               velocity  => velocity,
70               engine    => engine,
71               distance  => distance,
72               timestamp => timestamp);
73
74    -- ------------------------------------------------------------------------
75    -- ----- Memory implementation --------------------------------------------
76    -- ------------------------------------------------------------------------
77    mmu_read : process(mmu_addr, sensor_velocity, sensor_engine,
      ➥sensor_timestamp) is
78    begin
79      case mmu_addr is
80        when X"0"    => mmu_in <= sensor_velocity;
81        when X"1"    => mmu_in <= sensor_engine;
82        when X"8"    => mmu_in <= sensor_timestamp;
83        when others => mmu_in <= (others => '0');
84      end case;
85    end process mmu_read;
86
```

```
87    mmu_write : process(clk, rst) is
88    begin
89      if rst = '1' then
90        sensor_velocity  <= (others => '0');
91        sensor_engine    <= (others => '0');
92        sensor_timestamp <= (others => '0');
93
94      elsif rising_edge(clk) then
95        if sim_brake = '1' then
96          if sensor_velocity > 10 then
97            sensor_velocity <= sensor_velocity - 5;
98          else
99            sensor_velocity <= (others => '0');
100         end if;
101       elsif sim_accelerate = '1' then
102         sensor_velocity <= sensor_velocity + 1;
103         sensor_engine <= sensor_engine + ('0' & sensor_velocity(31 downto
              ➥1));
104       end if;
105
106       if mmu_rw = '1' then
107         case mmu_addr is
108           when X"8"   => sensor_timestamp <= mmu_out;
109           when others => null;
110         end case;
111       end if;
112     end if;
113   end process mmu_write;
114
115   -- ------------------------------------------------------------------
116   -- ------ Simulator implementation ----------------------------------
117   -- ------------------------------------------------------------------
118
119   rst <= '0' after 1 ns;
120
121   -- Print all updated signals
122   process is
123   begin
124     vunit_update(str(timestamp) & "␣" & str(mmu_rw) & "␣" & str(mmu_addr) &
          ➥ "␣" & str(mmu_out) & "␣" & str(velocity) & "␣" & str(engine) & "␣
          ➥" & str(distance));
125     wait on timestamp, mmu_rw, mmu_addr, mmu_out, velocity, engine,
          ➥distance;
126   end process;
127 end simulator;
```

Listing 3 – Testbench de contrôle du Simuateur en VHDL

5 Conclusion

Il est intéressant de voir que même à l'aide d'un simulateur, un tel projet peut être mené à bien. Il ne reste plus qu'à créer des prototypes de tous les circuits afin de tester et vérifier nos hypothèses, si celles-ci se montrent erroné, il sera toujours possible de modifier certains des modules pour recadrer le projet dans la bonne direction.