TD1 : VHDL, tables de vérité, diagramme d'évolution

Nous allons présenter dans ce chapitre les rudiments de programmation VHDL. Les descriptions abordées sont souvent qualifiée de RT-level (Niveau Transfert de registres) car elles concernent en général des circuits utilisés autour des registres (addition,....)

Programme VHDL simple

Un programme VHDL est composé d'au moins un couple entité/architecture. Une autre façon de présenter les choses est de dire que le programme minimum contient une entité et une architecture. Le rôle de l'entité est de déclarer quelles sont les entrées et comment elles s'appellent et la même chose pour les sorties. Par exemple si l'on désire décrire une fonction ET :

```
1         ENTITY ET IS
2         PORT(e0,e1 : IN BIT; --2 entrees appelees e0 et e1
3           s : OUT BIT);  -- 1 sortie appelée s
4         END ET;
5
6         ARCHITECTURE aET OF ET IS
7         BEGIN
8           s <= e0 and e1; -- equation de sortie
9         END aET;
```

Programme 1: Un programme simple pour un et logique

Les lignes 1 à 4 déclarent l'entité et les lignes 6 à 9 décrivent l'architecture. Tout programme VHDL contient au moins un couple entité architecture. On verra par la suite qu'il peut en contenir éventuellement plusieurs.

Les styles de programmation VHDL pour le combinatoire

Le style with select when

Nous présentons la technique des BIT_VECTOR ainsi que l'ensemble des styles de programmation. Imaginons que l'on ait la table de vérité (4 entrées 2 sorties) et l'entité correspondante :

a3	a2	a1	a0	s1	s0
0	1	0	1	1	1
0	1	1	0	0	1
1	1	0	1	1	0

```
ENTITY demo IS PORT(
  a : in BIT_VECTOR(3 DOWNTO 0);-- 4 entrées
  s : out BIT_VECTOR(1 DOWNTO 0));
  -- 2 sorties
END demo;
```

(ce qui n'est pas mentionné correspond à 00 en sortie pour les 13 lignes manquantes)

Une table de vérité comporte deux parties : partie gauche appelée partie SI qui doit donner l'ensemble des conditions possibles sur les entrées et une partie droite appelée partie ALORS donnant les valeurs des sorties.

Le plus simple quand on a un cahier des charges sous forme d'une table de vérité

est d'utiliser le constructeur VHDL "with select when". Il vous faut à tout prix apprendre à passer de l'un à l'autre sans trop vous poser de questions, c'est à dire comprendre le mécanisme de transformation. On remarquera, par exemple, que la partie SI se trouve à gauche dans la table de vérité, mais à droite dans le programme VHDL. Le programme VHDL en style "with select when" s'écrit :

```
1         ARCHITECTURE mydemo OF demo IS
2         BEGIN
3           WITH a SELECT            -- style with select when
4             s <= "11" WHEN "0101", -- premiere ligne
5                  "01" WHEN "0110", -- deuxieme ligne
6                  "10" WHEN "1101", -- troisieme ligne
7                  "00" WHEN OTHERS;
8         END mydemo;
```

Programme 2: Le style with select when

Ce style est adapté aux tables de vérité. En effet tous deux nécessitent des conditions mutuellement exclusives dans la partie SI (pour la table de vérité) et dans la partie WHEN du « WITH SELECT WHEN ». Tous les deux nécessitent une description exclusive de toutes les possibilités, visible avec le « WHEN OTHERS » en ligne 7 du programme 2. Il est possible de réunir deux sorties identiques en séparant les conditions correspondantes par "|" : WHEN "0010" | "1010", .

Exercice 1

Write a VHDL program for a one bit adder using "with select when" style. After, use two equations to solve this VHDL exercise.

Le style when else (pour information)

On écrirait le même programme en style "when else" :

```
1         ARCHITECTURE mydemo OF demo IS
2         BEGIN
3             -- style when else
4             s <= "11" WHEN a="0101" ELSE -- premiere ligne
5                  "01" WHEN a="0110" ELSE -- deuxieme ligne
6                  "10" WHEN a="1101" ELSE -- troisieme ligne
7                  "00";
8         END mydemo;
```

Programme 3: Combinatoire en style when else

Remarque : la structure "when else" ne nécessite pas de conditions mutuellement exclusives. Elle engendre alors une architecture avec priorité. Par exemple dans

```
1         j<= w when a='1' else
2             x when b='1' else
3             0;
```

les conditions ne sont pas mutuellement exclusives. Qu'adviendra-t-il en effet si a='1' et b='1' arrivent simultanément ? (Réponse : j <= w : c'est le premier qui gagne). On ne pourrait pas utiliser dans ce cas directement une structure "with select when" qui nécessite des conditions absolument exclusives.

Le style case when peut être aussi utilisé (en combinatoire comme en séquentiel, il nécessite un process dans les deux cas) :

Les styles case et if

```
1         ARCHITECTURE mydemo OF demo IS
2         BEGIN
3           PROCESS(a) BEGIN -- absolument necessaire
4             CASE a is  --style case when
5               WHEN "0101" => s <="11"; -- premiere ligne
6               WHEN "0110" => s <="01"; -- deuxieme ligne
7               WHEN "1101" => s <="10"; -- troisieme ligne
8               WHEN OTHERS => s <="00";
9             END CASE;
10          END PROCESS;
11        END mydemo;
```

Programme 4: Combinatoire avec style case when

Un autre style nécessite un process : if then else :

```
1         ARCHITECTURE mydemo OF demo IS
2         BEGIN
3           PROCESS(a) BEGIN
4               IF a="0101" THEN s <="11"; -- premiere ligne
5                 ELSIF a="0110" THEN s <="01"; -- deuxieme ligne
6                 ELSIF a= "1101" THEN s <="10"; -- troisieme ligne
7               ELSE s <="00";
8             END IF;
9           END PROCESS;
10        END mydemo;
```

Programme 5: Combinatoire avec style if then else

Conseil : utiliser le style with select when pour programmer du combinatoire si vous ne voulez pas établir d'équations. Si vous disposez des équations utiliser-les en vous rappelant que VHDL n'a pas de priorités du ET sur le OU. Il vous faudra donc des parenthèses.

VHDL et la librairie IEEE

Les seuls types utilisés jusqu'à maintenant sont les « bit » et « bit_vector ». Un bit prend seulement deux valeurs et ne permet pas de gérer le trois états par exemple. IEEE propose en supplément une librairie appelée std_logic. Son utilisation nécessite l'écriture des deux lignes suivantes

```
1         library ieee;
2         use ieee.std_logic_1164.all;
```

Programme 6: déclaration et utilisation d'une librairie ieee

en début du programme qui l'utilise. On a alors accès aux types «std_logic» et «std_logic_vector». Les valeurs prises par ces types sont :

```
4         'U'  Uninitialised        'Z'  High Impedance
5         'X'  Forcing Unknow       'W'  Weak Unknow
6         '0'  Forcing 0                 'L'  Weak 0
7         '1'  Forcing 1                 'H'  Weak 1
8         '-' -- Don't Care
```

On dispose de plus des fonctions rising_edge et falling_edge pour la détection de fronts montants et descendants (au lieu des if clk'event...).

Indication : il vous faudra écrire les deux lignes d'utilisation de la librairie devant chaque entités si vous en avez plusieurs. Si vous avez un programme avec quatre entités, il vous faudra écrire quatre fois ces lignes.

Exercice 2

Écrire le programme de l'exercice 1 en utilisant la librairie IEEE.

Exercice 3

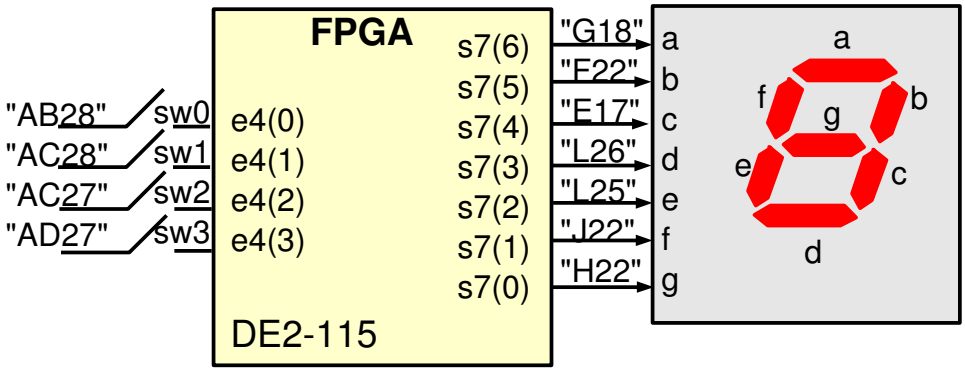


Figure 1: Afficheur 7 segments

1°) Écrire la table de vérité pour un afficheur 7 segments anode commune ('0' pour allumer)

e4(3)	e4(2)	e4(1)	e4(0)	s7(6)	s7(5)	s7(4)	s7(3)	s7(2)	s7(1)	s7(0)
0	0	0	0							
0	0	0	1							
0	0	1	0							
0	0	1	1							
0	1	0	0							
0	1	0	1							
0	1	1	0							
0	1	1	1							

1	0	0	0							
1	0	0	1							
1	0	1	0							
1	0	1	1							
1	1	0	0							
1	1	0	1							
1	1	1	0							
1	1	1	1							

2°) Réaliser le programme VHDL complet du transcodeur

```
1       library ieee;
2       use ieee.std_logic_1164.all;
3       entity transcod is port (
4           e4 : in std_logic_vector(3 downto 0);
5       -- précisez l'ordre
6           s7 : out std_logic_vector(6 downto 0));
7       end transcod;
8
9       architecture arch_transcod of transcod is
        begin
```

TD2 - Câbler des composants ensemble

I) Programme comportant plusieurs composants

Il existe plusieurs façons d'écrire un programme comportant plusieurs composants. Quelque soit la méthode, vous commencez par compter les composants différents et vous en obtenez N. Si vous avez deux composants ET, vous ne le comptez qu'une seule fois. Il vous faudra un couple entité architecture par composant. Par exemple, le schéma ci-dessous comporte N=3 composants (ET, OU, NON). Vous aurez à écrire autant de couples entité - architecture qu'il y a de composants plus un couple entité - architecture pour la description globale. Vous aurez donc N+1 couples.

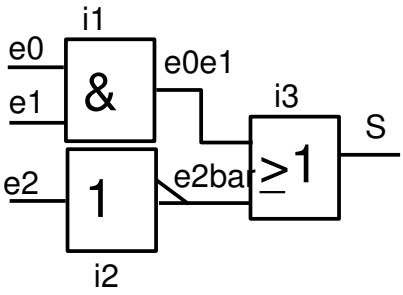


Figure 2: Ensemble de trois composants

Cette programmation s'appelle structurelle et elle revient à décrire un schéma ; dans la terminologie électronique cela s'appelle aussi une netlist.

Commencez votre travail par comprendre la notion de hiérarchie :

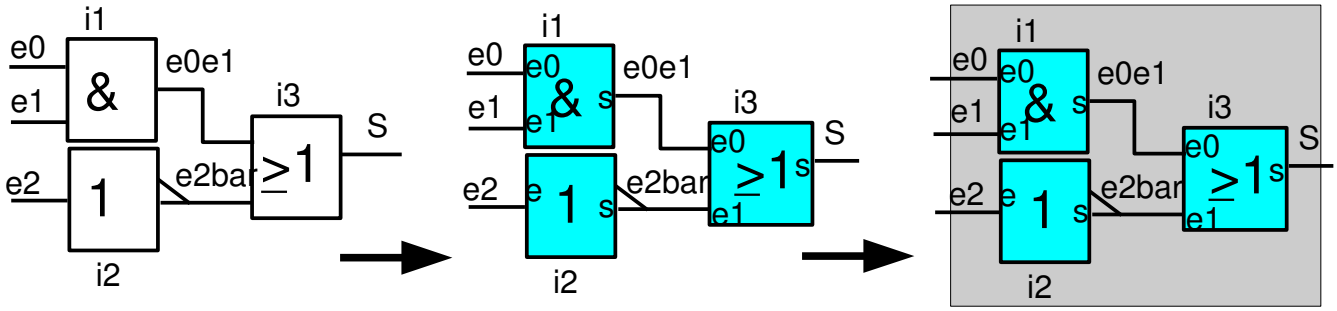
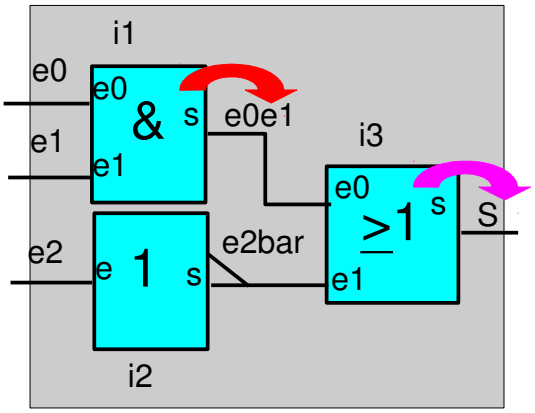


Figure 3: Ensemble de trois composants

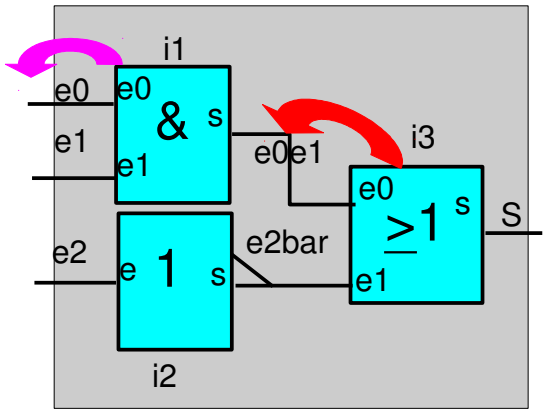
Dans la figure de droite vous avez quatre rectangles. Il y aura donc dans votre projet VHDL quatre entités et quatre architectures. Il y a un rectangle gris qui contient trois rectangles bleus.

Passer du temps à comprendre la figure ci-après : pour les "port map".

Voir aussi pour les explications :
http://fr.wikibooks.org/wiki/TD1_VHDL#Assembler_des_composants_en_VHDL



i1:et PORT MAP(e0=>e0,e1=>e1,s=>e0e1);
i3:ou PORT MAP(e0=>e0e1,e1=>e2bar,s=>S);



i1:et PORT MAP(e0=>e0,e1=>e1,s=>e0e1);
i3:ou PORT MAP(e0=>e0e1,e1=>e2bar,s=>S);

Une méthode consiste à utiliser un seul fichier.

Utiliser un seul fichier pour mettre plusieurs composants

L'utilisation d'un seul fichier se fait en déclarant des signaux et des composants avant le begin de l'architecture globale. Voici l'exemple de la figure 2

```
1         library ieee;
2         use ieee.std_logic_1164.all;
3         ENTITY Fct IS -- entité globale
4         PORT(e0,e1,e2 : IN std_logic;
5           s : OUT std_logic);
6         END Fct;
7         ARCHITECTURE truc OF Fct IS
8         -- signaux et composants avant le begin de l'architecture
9           SIGNAL e0e1,e2bar : std_logic;
10        COMPONENT et
11        PORT(e0,e1 : IN std_logic;
12          s : OUT std_logic);
13        END COMPONENT;
14        COMPONENT ou
15        PORT(e0,e1 : IN std_logic;
16          s : OUT std_logic);
17        END COMPONENT;
18        COMPONENT inverseur
19        PORT(e : IN std_logic;
20          s : OUT std_logic);
21        END COMPONENT;
22        BEGIN
23          i1:et PORT MAP(e0=>e0,e1=>e1,s=>e0e1);
24          i2:inverseur PORT MAP(e=>e2,s=>e2bar);
25          i3:ou PORT MAP(e0=>e0e1,e1=>e2bar,s=>s);
26        END truc;
27        -- fin de l'architecture globale
28        library ieee;
```

```
29        use ieee.std_logic_1164.all;
30        ENTITY et IS
31        PORT(e0,e1 : IN std_logic;
32          s : OUT std_logic);
33        END et;
34        ARCHITECTURE aet OF et IS
35        BEGIN
36          s<=e0 AND e1;
37        END aet;
38        library ieee;
39        use ieee.std_logic_1164.all;
40        ENTITY ou IS
41        PORT(e0,e1 : IN std_logic;
42          s : OUT std_logic);
43        END ou;
44        ARCHITECTURE aou OF ou IS
45        BEGIN
46          s<=e0 OR e1;
47        END aou;
48        library ieee;
49        use ieee.std_logic_1164.all;
50        ENTITY inverseur IS
51        PORT(e : IN std_logic;
52          s : OUT std_logic);
53        END inverseur;
54        ARCHITECTURE ainv OF inverseur IS
55        BEGIN
56          s<= NOT e;
57        END ainv;
```

Programme 7: Un programme VHDL avec plusieurs composants

Exercice 1

Réaliser l'architecture de ce nouveau schéma ci-dessous en prenant soin de décomposer en hiérarchie.

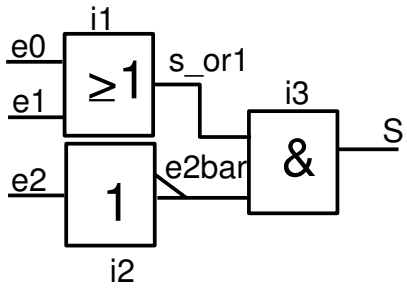


Figure 4: Ensemble de trois composants

Utiliser deux fichiers dont un package (pour information)

Lors du TD2 de ENSL1 (assemblage de fonctions) nous avons passé sous silence le fait que pour faire la description structurelle présentée il fallait utiliser notre propre package définissant ce qu'est un et, un ou et un inverseur. Nous écrivons ci-dessous la version complète du programme, on commence par le fichier principal :

```
1         USE work.mesportes.ALL;
2         ENTITY Fct IS
3         PORT(e0,e1,e2 : IN std_logic;
4           s : OUT std_logic);
5         END Fct;
6
7         ARCHITECTURE truc OF Fct IS
8         SIGNAL e0e1,e2bar : std_logic;
9         BEGIN
10          i1:et PORT MAP(e0=>e0,e1=>e1,s=>e0e1);
11          i2:inverseur PORT MAP(e=>e2,s=>e2bar);
12          i3:ou PORT MAP(e0=>e0e1,e1=>e2bar,s=>s);
13        END truc;
```

Programme 8: Fichier principal pour l'entité globale

La gestion des librairies dépend beaucoup de l'environnement dont vous disposez. Chaque fois que vous en utilisez une vous devez vous poser la question de savoir comment votre compilateur sait dans quel fichier il va trouver votre package. Ci-dessus (programme 8), est présenté la façon simple de l'environnement Xilinx : c'est le gestionnaire de projet dans lequel est rentré tous les fichiers sources qui est capable de trouver où est le package mesportes.

Et voici en condensé comment on réalise un package. La partie package déclare les composants (component) et le corps du fichier va comporter les entités et architectures des composants. Le fichier définitif aura donc la forme suivante :

```
58        PACKAGE mesportes IS
59        COMPONENT et -- tout cela est visible dans un .ALL
60        PORT(e0,e1 : IN std_logic;
61          s : OUT std_logic);
62        END COMPONENT;
63        COMPONENT ou
64        PORT(e0,e1 : IN std_logic;
65          s : OUT std_logic);
66        END COMPONENT;
67        COMPONENT inverseur
68        PORT(e : IN std_logic;
69          s : OUT std_logic);
70        END COMPONENT;
71        END mesportes; --********** fin du package **********
72        ENTITY et IS  --********** debut implantation **********
73        PORT(e0,e1 : IN std_logic;
74          s : OUT std_logic);
75        END et;
76        ENTITY ou IS
77        PORT(e0,e1 : IN std_logic;
78          s : OUT std_logic);
79        END ou;
80        ENTITY inverseur IS
81        PORT(e : IN std_logic;
82          s : OUT std_logic);
83        END inverseur;
84        ARCHITECTURE aet OF et IS
85        BEGIN
```

```
86          s<=e0 AND e1;
87        END aet;
88        ARCHITECTURE aou OF ou IS
89        BEGIN
90          s<=e0 OR e1;
91        END aou;
92        ARCHITECTURE ainv OF inverseur IS
93        BEGIN
94          s<= NOT e;
95        END ainv;
```

Programme 9: Exemple de package

II) Utiliser des process pour éviter les PORT MAP

1°) Équations et process

Il est possible d'éviter les PORT MAP dans les programmes VHDL mais ce n'est pas une technique toujours recommandée. En effet la réalisation matérielle consiste toujours à assembler des fonctions simples pour en faire des fonctions complexes. Il est donc **IMPERATIF** de savoir réaliser un PORT MAP. Les TPs vous le montreront.

Au lieu de câbler, il est donc possible d'utiliser :

- des équations

- des if, case ... qui seront nécessairement entourés par un process

Par exemple, la figure 2 peut être réalisée par le programme :

```
1         library ieee;
2         use ieee.std_logic_1164.all;
3         ENTITY Fct IS -- entité globale
4         PORT(e0,e1,e2 : IN std_logic;
5           s : OUT std_logic);
6         END Fct;
7         ARCHITECTURE truc OF Fct IS
8         -- signaux et composants avant le begin de l'architecture
9           SIGNAL e0e1,e2bar : std_logic;
10        BEGIN
11          e0e1 <= e0 and e1 ;
12          e2bar <= NOT e2 ;
13          s <= e0e1 OR e2bar ;
14        END truc;
```

Programme 10: Un programme VHDL avec plusieurs composants `sans PORT MAP`

qui est un programme bien plus court que le Programme 7 : 14 lignes contre 57.

Exercice 3

A partir du morceau de programme ci-dessous qui réalise une conversion binaire décimale, on vous demande d'en faire un schéma complet :

```
1         entity convertisseur is port (
2             clk, en, init : in std_logic;
3             entreebin8 : in std_logic_vector(7 downto 0);
4             unite, dizaine : out std_logic_vector(3 downto 0));
5         end convertisseur;
6
7         architecture convert of convertisseur is
```

```
8           component CB8EDL is port (
9                 clk :in std_logic;
10                en : in std_logic;
11                load : in std_logic;
12                e8 : in std_logic_vector(7 downto 0);
13                done : out std_logic;
14                s8: out std_logic_vector(7 downto 0)
15                );
16          end component;
17          component cmpt_bcd10 IS -- Définition des entrées/sorties
18              PORT(clk, en, Init : IN std_logic;
19              rco : OUT std_logic;
20              q : OUT std_logic_vector(3 downto 0));
21          END component;
22          component sequenceur is
23            port (
24              clk,en,done,init : in std_logic;
25             endec,load,memorize : out std_logic);
26          end component;
27          signal en_seq,s_done,s_load,s_rco,s_memorize : std_logic;
28          signal regAff : std_logic_vector(7 downto 0);
29          signal s_dizaine, s_unite : std_logic_vector(3 downto 0);
30        begin
31          conv1:CB8EDL port map (
32            clk => clk,en => en_seq,load=>s_load,e8 =>entreebin8,done => s_done,
33           s8(3 downto 0) => open,s8(7 downto 4)=>open);
34          conv2:sequenceur port map (
35            clk=>clk,en => en, endec => en_seq,load=>s_load,
36            done => s_done,init => init,
37           memorize=>s_memorize);
38          conv3:cmpt_bcd10 port map (
39            clk => clk,en => en_seq, Init => s_load,rco => s_rco, q => s_unite);
40          conv4:cmpt_bcd10 port map (
41            clk => clk,en => s_rco, Init => s_load,rco => open, q => s_dizaine);
42        --       registre d'affichage
43          process(clk) begin
44            if rising_edge(clk) then
45             if Init = '1' then
46                  regAff <= x"00";
47               elsif s_memorize='1' then
48               regAff <= s_dizaine & s_unite;
49              end if;
50           end if;
51          end process;
52          dizaine <= regAff(7 downto 4);
53          unite <= regAff(3 downto 0);
54        end convert;
```

Programme 11: Autre programme VHDL avec des composants

III) Les LUTs et leur assemblage en VHDL

1°) Table de vérité et LUT

Une table de vérité de trois entrées peut être représentée par un nombre 8 bits que l'on convertit en hexadécimal. Soit donc la table de vérité suivante (trois entrées notées e0, e1 et e2, une sortie notée s) :

e2	e1	e0	s	
0	0	0	0	b_0
0	0	1	1	b_1
0	1	0	1	b_2
0	1	1	0	b_3
1	0	0	1	b_4
1	0	1	0	b_5
1	1	0	1	b_6
1	1	1	0	b_7

Vous pouvez synthétiser la table de vérité à l'aide d'un seul nombre sur 8 bit (poids faible en haut) :
- en binaire ce nombre vaut : 0b01010110
- en hexadécimal ce nombre vaut : 0x56 (noté X"56" en VHDL)

Aucune simplification à faire ici

La valeur hexadécimale 56 (**notée X"56" en VHDL**) est la valeur avec laquelle il faudra initialiser votre LUT avec un composant **lut3**.

Pour 4 entrées, on utilise une **lut4** avec 4 chiffres hexadécimaux.

2°) Utiliser des LUTs en VHDL

Un exemple est donné maintenant :

```
1         library IEEE; -- transodeur binaire -> 7 segments
2         use IEEE.STD_LOGIC_1164.ALL;
3         ENTITY transcodeur IS PORT(
4           e : in STD_LOGIC_VECTOR(3 DOWNTO 0);   -- 4 entrées
5           s : out STD_LOGIC_VECTOR(6 DOWNTO 0)); -- 7 sorties
6         END transcodeur;
7         ARCHITECTURE atranscodeur OF transcodeur IS BEGIN
8         i1 : LUT4
9           generic map (mask => X"EAAA")
10          port map( I0 => e(0),
11                    I1 => e(1),
12                    I2 => e(2),
13                    I3 => e(3),
14                     O => s(0) );
15        .....
```

Cet exemple vous montre comment on câble une **LUT4** en VHDL (port map) et comment on l'initialise (generic map). Le câblage de ce composant est correct mais pas son initialisation puisqu'on vous demande de la calculer plus loin.

Mais cet exemple ne fonctionne pas tout seul. Il nécessite l'utilisation du composant LUT4 que l'on doit créer :

```
1         library ieee;
2         use ieee.std_logic_1164.all;
3         library altera;
4         use altera.altera_primitives_components.all;
5         entity LUT4 is
6           generic(mask : std_logic_vector(15 downto 0):=X"0000");
7           port (
8            in4 : in std_logic_vector(3 downto 0);
9           out1 : out std_logic);
10        end LUT4;
11        architecture arch_lut4 of LUT4 is
12        signal s_lut : std_logic_vector(3 downto 0);
```

```
13        signal s_out : std_logic;
14        begin
15          ic1:lut_input port map(a_in => in4(0),a_out=>s_lut(0));
16          ic2:lut_input port map(a_in => in4(1),a_out=>s_lut(1));
17          ic3:lut_input port map(a_in => in4(2),a_out=>s_lut(2));
18          ic4:lut_input port map(a_in => in4(3),a_out=>s_lut(3));
19          with s_lut select
20            s_out <= mask(0) when "0000",
21                     mask(1) when "0001",
22                     mask(2) when "0010",
23                     mask(3) when "0011",
24                     mask(4) when "0100",
25                     mask(5) when "0101",
26                     mask(6) when "0110",
27                     mask(7) when "0111",
28                     mask(8) when "1000",
29                     mask(9) when "1001",
30                     mask(10) when "1010",
31                     mask(11) when "1011",
32                     mask(12) when "1100",
33                     mask(13) when "1101",
34                     mask(14) when "1110",
35                     mask(15) when "1111";
36          ic5: lut_output port map(a_in=>s_out,a_out=>out1);
37        end arch_lut4;
```

3°) Exercice 4 (transcodeur binaire vers sept segments)

Réaliser le schéma correspondant au "port map" de l'exemple ci-dessus dans le composant "**transcodeur**" (deux rectangles, un appelé "**transcodeur**" et un appelé "**LUT4**"). Compléter ce schéma avec d'éventuels pointillés en montrant que sept LUTs seront nécessaires pour réaliser le transcodeur complet de l'exercice 3 du TD précédent.

Réaliser une table de vérité complète du décodage demandé.

En déduire les 7 valeurs hexadécimales d'initialisation des LUTs.

Réaliser l'architecture complète de "**transcodeur**" en VHDL.

TD3 - Le séquentiel

I) Le séquentiel

Le séquentiel dispose lui-aussi de ses propres styles. L'équivalent de la table de vérité (spécification sans équation) est le diagramme d'évolution. Il est possible ici encore d'utiliser soit les équations (de récurrence alors) soit le diagramme d'évolution. Nous présentons les deux styles maintenant. Ils sont caractérisés par la détection d'un front montant sur une horloge.

Le séquentiel simple (diagramme d'évolution) avec équations de récurrence

Dans cette section, on ne s'intéresse qu'aux diagrammes d'évolution simples c'est à dire avec des transitions inconditionnelles. Les diagrammes d'évolutions un peu plus complexes sont traités plus loin (chapitre 3).

On établit d'abord un tableau état présent état futur duquel on déduit des équations de récurrences. La méthode est présentée sur un exemple. Nous commençons par le graphe d'évolution (avec ses 4 états et 4 flèches transitions) :

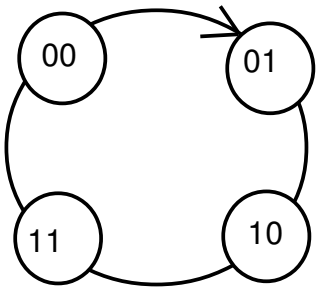
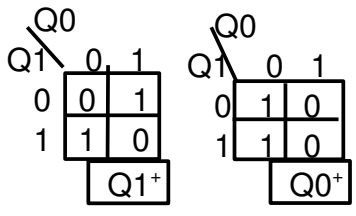


Figure 5: Diagramme d'évolution sur quatre états

Puis le tableau état présent / états futurs ainsi que le tableau de Karnaugh associés sont présentés :

État présent		État futur	
Q1	Q0	$Q1^+$	$Q0^+$
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0



Équations de récurrence :

$Q_1^+ = Q_1 \text{ XOR } Q_0$

$Q_0^+ = /Q_0$

Figure 6: Tableau état présent / état futur et ses équations de récurrences

Cela donne le programme :

```
1         -- compteur premiere version
2         library ieee;
3         use ieee.std_logic_1164.all;
4         ENTITY cmpt IS PORT (
5         clk: IN std_logic;
6         q0,q1: OUT std_logic);
7         END cmpt;
8         ARCHITECTURE acmpt OF cmpt IS
9           SIGNAL s_q0, s_q1 :  std_logic ;
10        BEGIN
11          PROCESS (clk) BEGIN
12          IF (clk'EVENT AND clk='1') THEN
13              s_q0 <= NOT s_q0;
14              s_q1 <= s_q0 XOR s_q1;
15          END IF;
16          END PROCESS;
17        q0 <= s_q0 ;
18        q1 <= s_q1 ;
19        END acmpt;
```

Programme 12: Le séquentiel simple avec équations de récurrences : un compteur

où les deux équations de récurrences se trouvent en ligne 10 et 11 encadrées par une détection de front d'horloge.

Le séquentiel simple (diagramme d'évolution) sans équations de récurrence

La programmation d'un diagramme d'évolution se fait directement à partir du tableau état présent / état futur comme le montre le code VHDL ci-dessous :

```
1         -- compteur deuxieme version voir programme 12
2         library ieee;
3         use ieee.std_logic_1164.all;
4         ENTITY cmpt IS PORT (
5         clock: IN std_logic;
6         q : OUT std_logic_vector(1 DOWNTO 0));
7         END cmpt;
8         ARCHITECTURE mydemo OF cmpt IS
9         SIGNAL s_q :  std_logic_vector(1 downto 0) ;
10          BEGIN
11               PROCESS(clock) BEGIN
12                 IF clock'EVENT AND clock='1' THEN
13                CASE s_q IS  --style case when
14                  WHEN "00" => s_q <="01";
15                  WHEN "01" => s_q <="10";
16                  WHEN "10" => s_q <="11";
17                  WHEN OTHERS => s_q <="00" ;
18                END CASE;
19                 END IF;
```

```
20              END PROCESS;
21          q <= s_q ;
22        END mydemo;
```

Programme 13: Le séquentiel simple sans équations de récurrences : toujours le même compteur

Exercice 1

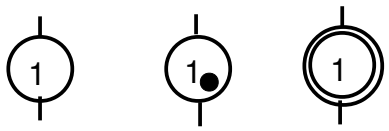
Réaliser un compteur GRAY sur 3 bits en utilisant ces deux méthodes.

II) Graphe d'états

Un graphe d'état est une suite d'états et de transitions réceptives.

1°) États

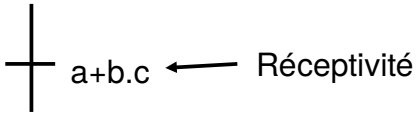
Comme on l'a fait jusqu'à présent, les états sont représentés par des cercles.



inactif actif initial

2°) Transitions

Les transitions par contre deviennent réceptives (barrées par un trait)



3°) Équations de récurrence

Code One-hot-One (une bascule par état)

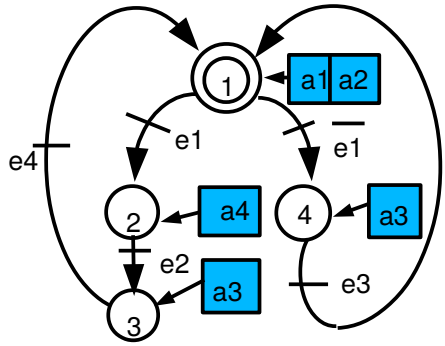
On cherche pour chacun des états i les conditions d'activations AC_i et les déactivations D_i puis on écrit :

$x_i^+ = AC_i + \overline{D_i} \cdot x_i + Init$ pour un état initial et

$x_i^+ = (AC_i + \overline{D_i} \cdot x_i) \cdot \overline{Init}$ pour un état normal.

4°) Implantation

On implante ces équations de récurrence facilement avec des bascules D (voir TD 10).



AC1 = x3.e4+x4.e3 D1 = e1+/e1=1
AC2 = x1.e1 D2 = e2
AC3 = x2.e2 D3 = e4
AC4 = x1./e1 D4 = e3

$x1^+$ = x3.e4+x4.e3 + Init
$x2^+$ = (x1.e1+x2./e2)./Init
$x3^+$ = (x2.e2+x3./e4)./Init
$x4^+$ = (x1./e1+x4./e3)./Init

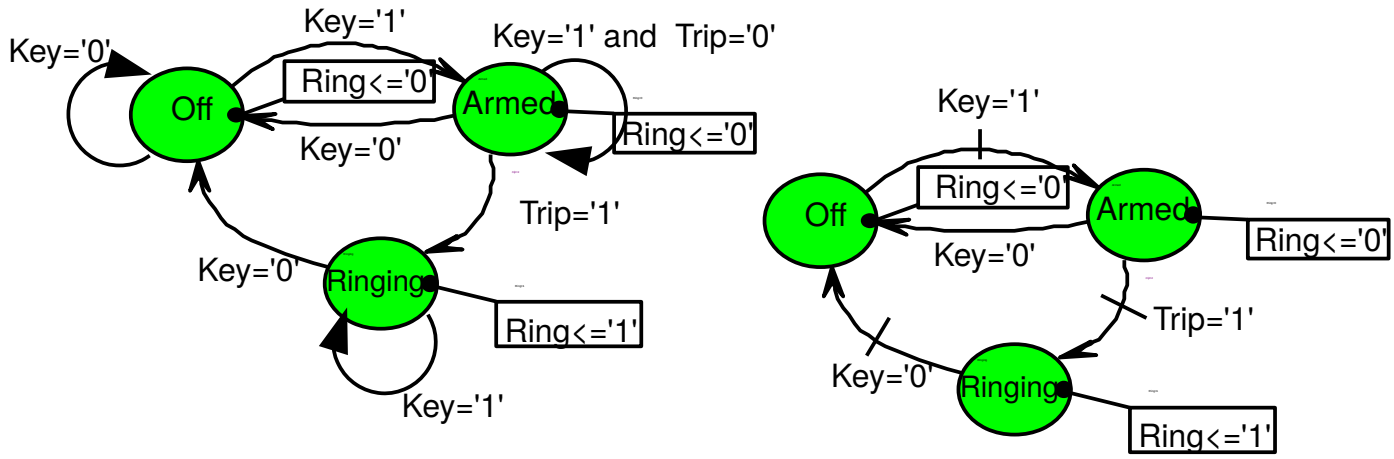
Équations de sorties
a1 = x1 a2 = x1
a3 = x3 + x4
a4 = x2

Figure 7: Diagramme d'évolution sur quatre états

```
1         -- programme VHDL correspondant au graphe d'états précédent
2         library ieee;
3         use ieee.std_logic_1164.all;
4         ENTITY graf1 IS
5         PORT (I,e1,e2,e3,e4,clk : IN std_logic;
6               a1,a2,a3,a4 : OUT std_logic);
7         END graf1;
8         ARCHITECTURE agraf1 OF graf1 IS
9         SIGNAL x1,x2,x3,x4 : std_logic;
10        BEGIN
11          PROCESS(clk) BEGIN
12            IF (clk'event AND clk='1') THEN
13              x1 <= (x3 AND e4) OR (x4 AND e3) OR I;
14              x2 <= (x1 AND e1 AND NOT I) OR (x2 AND NOT e2 AND NOT I);
15              x3 <= (x2 AND e2 AND NOT I) OR (x3 AND NOT e4 AND NOT I);
16              x4 <= (x1 AND NOT e1 AND NOT I) OR (x4 AND NOT e3 AND NOT I);
17            END IF;
18          END PROCESS;
19          a1 <= x1;
20          a2 <= x1;
21          a3 <= x3 OR x4;
22          a4 <= x2;
23        END agraf1;
```

IV) Graphe d'états sans équation de récurrence

Examinons le cas de la sonnerie d'un réveil. On vous présente la description de son fonctionnement sous la forme d'un graphe d'évolution (à gauche) et d'un graphe d'états (à droite).




```
1         library IEEE;
2         use IEEE.STD_LOGIC_1164.ALL;
3         ENTITY Alarm IS
4           PORT(
5                 clock,Key,Trip :IN std_logic;
6                 Ring :OUT std_logic
7                 );
8         END Alarm;
9         ARCHITECTURE ar  OF Alarm IS
10          TYPE typetat IS (Armed, Off, Ringing);
11          SIGNAL etat : typetat;
12          BEGIN
13            PROCESS (clock,etat) BEGIN  -- partie séquentielle
14                 IF Clock ='1' AND Clock'EVENT THEN
15                        CASE etat IS
16                           WHEN Off => IF key ='1' THEN etat <= Armed;
17                                       ELSE etat <= Off;
18                                    END IF;
19                           WHEN Armed => IF Key = '0' THEN
20                                         etat <= Off;
21                                          ELSIF Trip ='1' THEN
22                                           etat <= Ringing;
23                                            ELSE etat <= Armed;
24                                          END IF;
25                           WHEN Ringing => IF Key ='0' THEN
26                                           etat <= Off;
27                                          ELSE etat <= Ringing;
28                                                       END IF;
29                        END CASE;
30                  END IF;
31                  IF etat=Ringing THEN  -- partie combinatoire
32                    Ring<='1';
33                  ELSE Ring <='0';
34                  ENDIF
35             END PROCESS;
36        END ar;
```

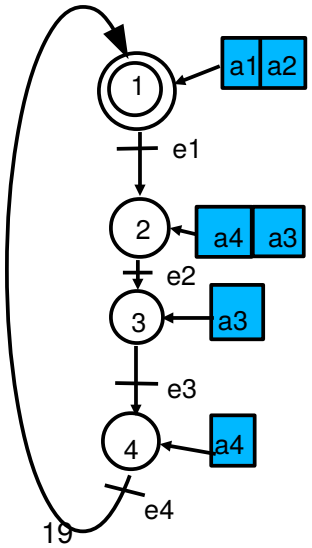
Exercice 2 (Analyse de code)

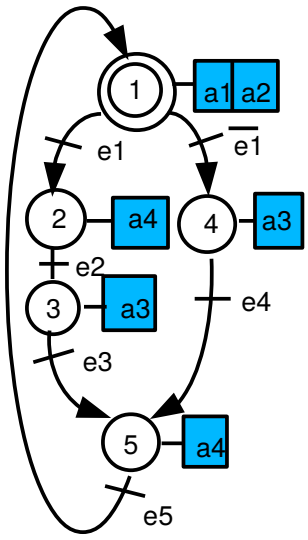
Dessiner le graphe d'état correspondant au morceau de code cuivant :

```
1         etat_suivant: process (reg_etat,d,g)
2                 begin
3                         --next_etat<=reg_etat;sur front pas montré ici
4                         case reg_etat is
5                                 when S1=>
6                                         if d='1' and g='0' then
7                                                 next_etat<=S2;
8                                         elsif g='1' and d='0' then
9                                                 next_etat<=S3;
10                                        else
11                                                next_etat<=S1;
12                                        end if;
13                                when S2=>
14                                        if g='1' then
15                                                next_etat<=S4;
16                                        else
17                                                next_etat<=S2;
18                                        end if;
19                                when S3=>
20                                        if d='1' then
21                                                next_etat<=S5;
22                                        else
23                                                next_etat<=s3;
24                                        end if;
25                                when S4=>next_etat<=S6;
26
27                                when S5=>next_etat<=S7;
28
29                                when S6=>
30                                        if g='0' then
31                                                next_etat<=S1;
32                                        else
33                                                next_etat<=S6;
34                                        end if;
35                                when S7=>
36                                        if d='0' then
37                                                next_etat<=S1;
38                                        else
39                                                next_etat<=S7;
40                                        end if;
41                                end case;
42          end process etat_suivant;
```

Exercice 3

Given these two states flow diagram, build up the corresponding sequential functions. Write a VHDL program with case style.





TD4 - Compteurs et registres

Le séquentiel consiste toujours à calculer un état futur à partir d'un état présent. Le terme calcul est employé ici de manière très générale, au sens booléen ou au sens arithmétique ou autre. Lorsque ce calcul s'écrit avec un opérateur simple on parlera de **séquentiel régulier**. Parmi les opérateurs simples on trouve l'incrémentation qui donne lieu à des compteurs. Ils permettent de gérer tout ce qui est temporisation et évidemment comptage. Un autre opérateur simple est la concaténation (mise en bout à bout) réalisée en VHDL avec l'opérateur et commercial « & ». Il nous permettra de réaliser les registres à décalage.

I) Le compteur simple

Il est possible d'utiliser un style case pour programmer un compteur mais cela devient vite fastidieux lorsque le nombre de bits augmente. On multiplie par deux le nombre d'états chaque fois que l'on ajoute un bit.

L'idéal serait donc de pouvoir écrire quelque chose du style

```
1        compteur <= compteur + 1;
```

Cela peut se faire en respectant les conditions suivantes :

- utilisation de la librairie IEEE 1164
- utilisation de la librairie IEEE ARITH
- utilisation de la librairie IEEE UNSIGNED
- déclaration de compteur comme std_logic_vector

Avec XILINX et Altera cela se fait avec les lignes (devant chacune des entités concernées) :

```
1        library ieee;
2        use ieee.std_logic_1164.all;
3        use ieee.std_logic_arith.all;
4        use ieee.std_logic_unsigned.all;
```

Programme 14: Utilisations des packages Xilinx et Altera (non portables)

Même si cette façon de faire n'est pas portable, on utilisera donc ces 4 lignes.

Exemple de compteur binaire

Voici un listing complet de compteur :

```
1        library ieee;
2        use ieee.std_logic_1164.all;
3        use ieee.STD_LOGIC_ARITH.all;
4        use ieee.STD_LOGIC_UNSIGNED.all;
5        entity free_run_bin_counter is
```

```
6           generic(N: integer := 8);
7           port(
8              clk, reset: in std_logic;
9              max_tick: out std_logic; -- peut être retire
10             q: out std_logic_vector(N-1 downto 0)
11          );
12       end free_run_bin_counter;
13
14       architecture arch of free_run_bin_counter is
15          signal r_reg: unsigned(N-1 downto 0);
16       begin
17          -- register
18          process(clk,reset)
19          begin
20             if (reset='1') then
21                r_reg <= (others=>'0');
22             elsif (clk'event and clk='1') then
23                r_reg <= r_reg + 1;
24             end if;
25          end process;
26          -- output logic
27          q <= std_logic_vector(r_reg);
28          max_tick <= '1' when r_reg=(2**N-1) else '0'; --peut être retire
29       end arch;
```

Programme 15: Compteur binaire

Exercice 1

Vous disposez d'une horloge rapide et vous voulez en réaliser une plus lente dont la fréquence est divisée par 32768. Proposez un compteur avec comme entrée h_rapide et comme sortie h_lente (toutes deux sur un bit).

II) Compteur avec Remise à zéro (raz)

L'entrée raz sur un compteur est une entrée qui permet de mettre la valeur du compteur à 0. Elle peut être synchrone (prise ne compte seulement sur front d'horloge) ou asynchrone. Commençons par l'entité qui nous servira pour nos deux styles de forçage.

```
1        library ieee;
2        use ieee.std_logic_1164.all;
3        use ieee.STD_LOGIC_ARITH.all;
4        use ieee.STD_LOGIC_UNSIGNED.all;
5        ENTITY Compteur IS
6        PORT (
7          clk,raz :IN std_logic;
8          q : OUT std_logic_vector(3 downto 0));
9        END Compteur;
10
```

Programme 16: Entité générale d'un compteur (avec les packages IEEE)

Nous allons présenter tour à tour la méthode synchrone d'abord puis la méthode asynchrone.

Méthode synchrone

La méthode synchrone consiste à réaliser le raz (remise à zéro) dans le front d'horloge. Nous en présentons les éléments essentiels dans le programme ci-dessous.

```
1          -- register
2          process(clk)
3          begin
4             if (clk'event and clk='1') then
5                if (raz = '1') then
6                  r_reg <= (others=>'0');
7                else
8                  r_reg <= r_reg + 1;
9                end if;
10            end if;
11         end process;
```

Programme 17: Raz synchrone pour un compteur

Remarquez le (OTHERS=>'0') en ligne 4 qui permet de remettre le compteur à zéro quelque soit son nombre de bits. L'écriture q<=''0000'' nécessite de savoir qu'il y a 4 zéros dans la chaîne de caractères. Quant au « IF raz » de la ligne 3 il se trouve bien à l'intérieur du « IF clk'event » de la ligne précédente. Bien sûr ce process est à mettre dans une architecture.

Méthode asynchrone

La méthode asynchrone consiste à réaliser le raz (remise à zéro) en dehors du front d'horloge. Ce signal devient alors prioritaire.

```
1       -- register
2          process(clk,reset)
3          begin
4             if (reset='1') then
5                r_reg <= (others=>'0');
6             elsif (clk'event and clk='1') then
7                r_reg <= r_reg + 1;
8             end if;
9          end process;
```

Programme 18: Raz asynchrone pour un compteur

Remarquez en ligne 2 que la liste de sensibilité du process comporte maintenant l'entrée raz en plus de l'horloge.

Exercice 2

Réaliser un compteur avec SET et RESET synchrones et asynchrones.

Modifier ce compteur pour qu'il compte jusqu'à 24.

III) Temporisation

L'application la plus courante des compteurs est la temporisation.

Exercice 3

On désire réaliser les deux signaux hsynch et vsynch nécessaire au bon fonctionnement d'un écran VGA. Ils sont caractérisés par les durées suivantes :

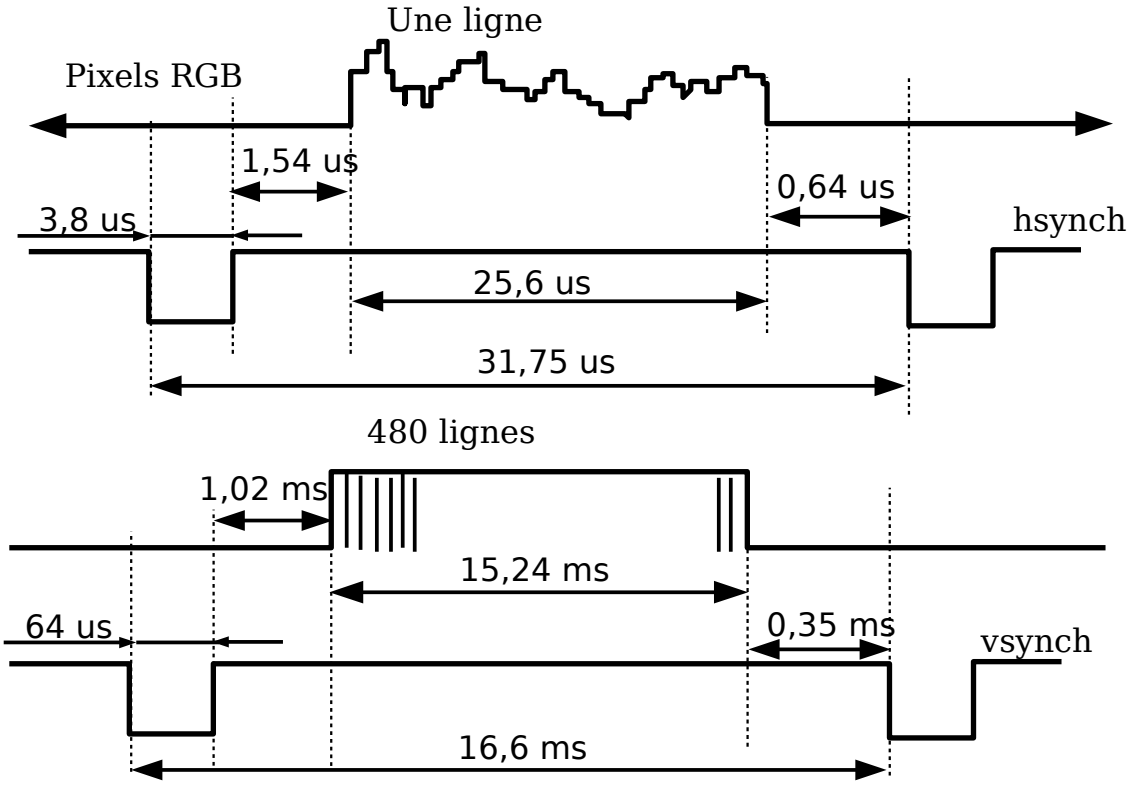


Figure 8: Chronogramme VGA

Techniquement la réalisation est faite de la manière suivante :

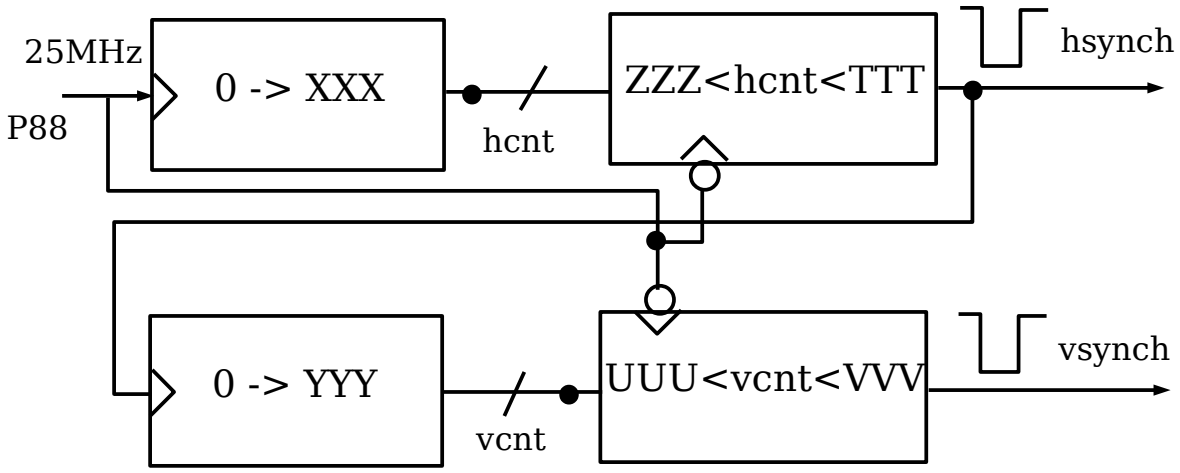


Figure 9: Réalisation matérielle de la synchronisation VGA

On voit apparaître deux compteurs et une partie combinatoire de comparaison qui doit être synchronisée.

1°) Calculer la période de P88.

2°) **Le compteur 0 -> XXX commence à compter au début des 25,6 µs**.

Remplir le tableau ci-dessous :

temps(ns)	25600	25600+640=26240	30040	31750
Compteur1				
Symboles		ZZZ+1	TTT-1	XXX

On arrondit en général XXX à 799 (donc période 800).

Déduire de tout cela la valeur de ZZZ et TTT.

3°) Ce sont ces hsynch qui incrémentent le compteur 0->YYY. Quelle est la période correspondante (si l'on prend XXX=799) ? Combien de temps prendra le balayage vertical des 480 lignes de l'écran (en lieu et place des 15,24 ms) avec cette horloge ?

4°) A partir du résultat de la question précédente, remplir le tableau ci-dessous :

temps(us)	15360	15360+350=15710	15774	16600
Compteur2				
Symboles		UUU+1	VVV-1	YYY

Est-il normal d'arrondir YYY à 520 ?

5°) Déduire les valeurs de UUU et VVV

6°) Imaginer une architecture capable de dessiner un petit rectangle dont les deux coordonnées seront fixées par l'extérieur. On s'aidera pour cet exercice de l'architecture ci-dessous (Figure 10).

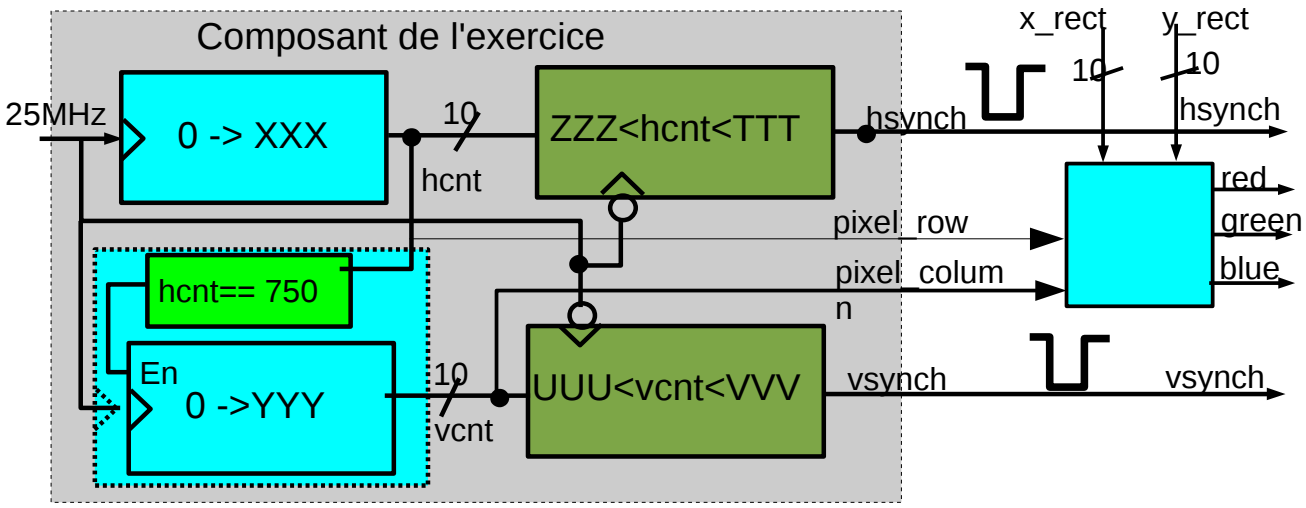


Figure 10: Réalisation matérielle de la synchronisation VGA et d'un dessin sur l'écran

On peut ainsi réaliser un affichage VGA comme ci-dessous :

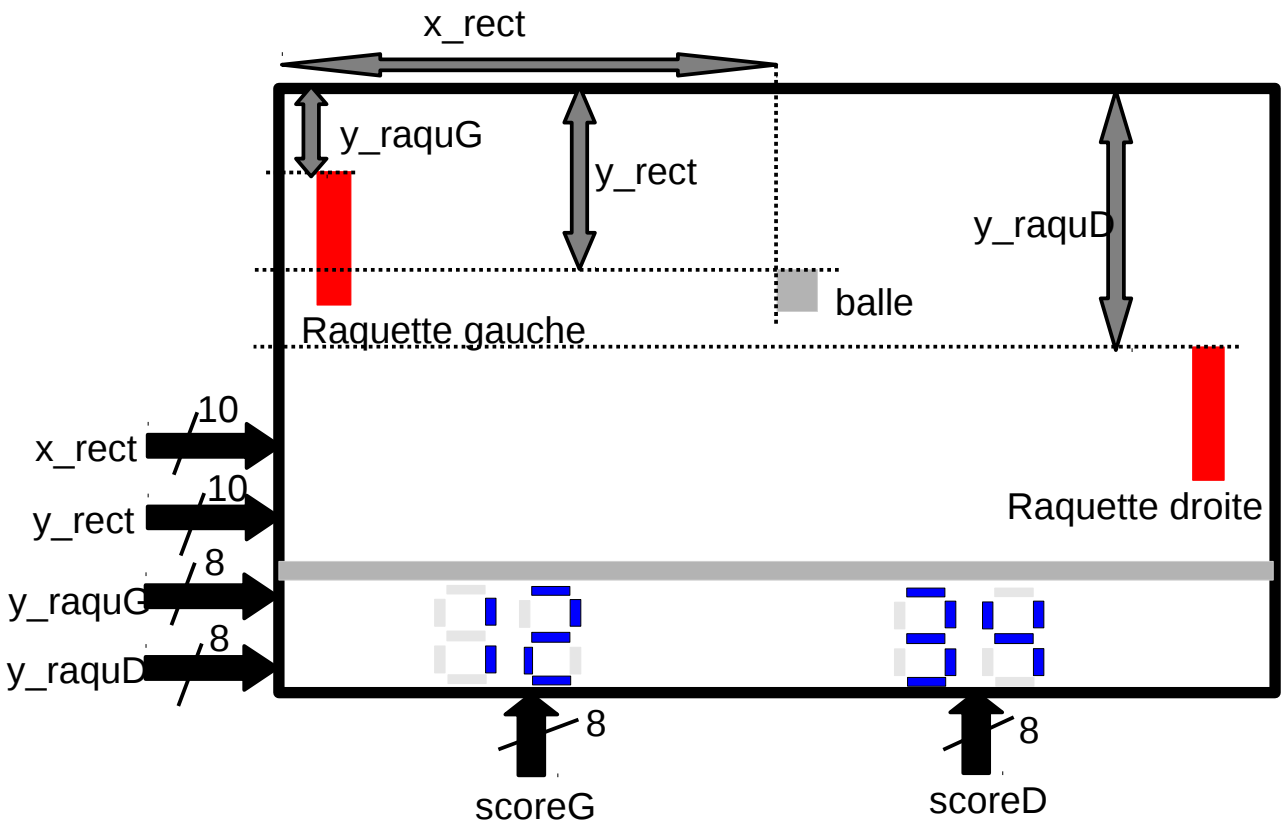


Figure 11: Comment interfacer ces éléments VGA avec un picoBlaze ?

7°) Si l'interface est réalisée avec des registres 8 bits, combien en faut-il ?

TD5 - Embarquer un processeur ATTiny861

A partir de maintenant, nous allons étudier comment mettre un processeur dans un FPGA. Le processeur sera complètement fourni en VHDL : il s'agit d'un AVR compatible ATTiny861 de chez ATMEL. Il sera programmé en C.

I) Architecture générale des micro-contrôleurs modernes

Les micro-contrôleurs comme les micro-processeurs sont composés de :

- logique classique UAL (Unité Arithmétique et Logique), des registres

- mémoire RAM et mémoire programme (anciennement ROM puis EPROM puis EEPROM (FLASH))

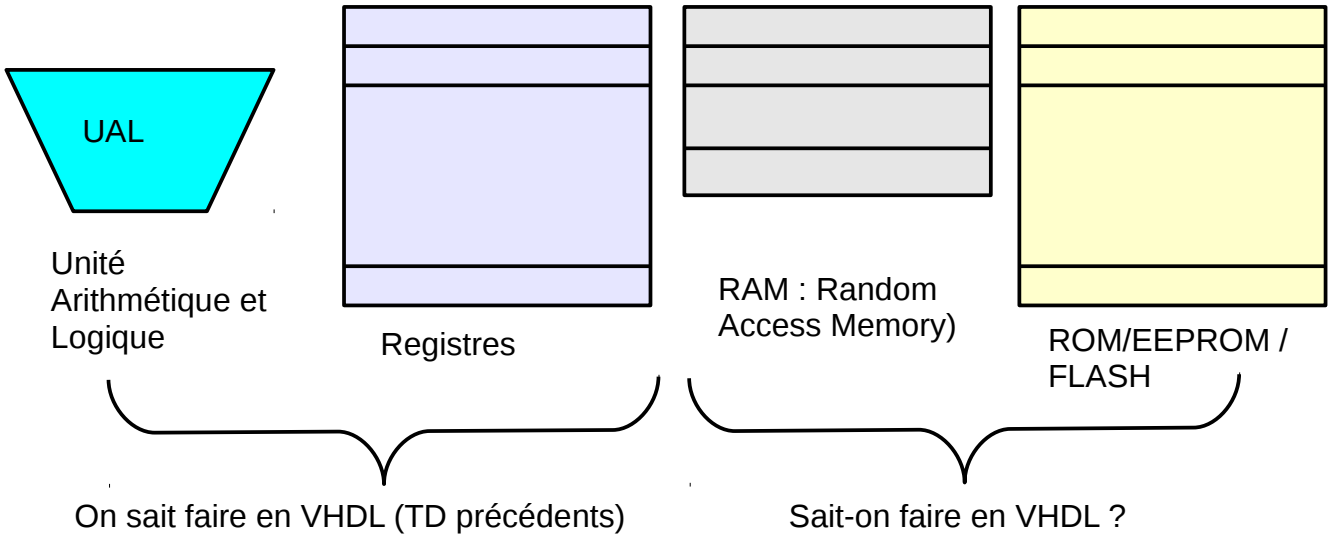


Figure 12: Architecture simplifiée d'un processeur

En cours, on vous expliquera comment réaliser cela dans un FPGA.

II) Architecture de l'ATTiny861

Dans la section précédente, on expliquait comment tout un processeur pouvait se réaliser en VHDL. Pour vous convaincre on vous présente maintenant l'ensemble des fichiers nécessaires à l'ATTiny861.

1°) Architecture mémoire

La donnée de base de AVR (ATTinyXXX) est l'octet. L'octet comporte 8 bits. Le bit 0 est le bit de poids faible et le bit 7 est le bit de poids fort.

Mémoire programme : la mémoire programme de l'AVR (ATTiny861) est organisée en mots de 16 bits et elle peut en contenir 4096 (soit 4k). Le bit 0 est aussi le bit de poids faible et le bit 15 est le bit de poids fort. La mémoire mémoire programme comporte donc 4k x 16 bits = 8 ko (d'où le nom ATTiny861).

Exercice 1

On rappelle qu'une mémoire est caractérisée par un bus de donnée de largeur n (n

bits de D_0 à D_{n-1}) et un bus d'adresse de largeur m (m bits de A_0 à A_{m-1}).
1°) Pour la mémoire programme de l'AVR (ATTiny861) donner m et n.
2°) La mémoire programme de l'AVR (ATTiny261) est 4 fois plus petite : donner les nouveaux m et n.

Mémoire EEPROM : l'AVR (ATTiny861) dispose de 512 octets de mémoire EEPROM.

Exercice 2 (hors TD)
Calculer m et n pour l'EEPROM de l'AVR (ATTiny861).

Mémoire donnée :
Il est difficile de séparer la mémoire de données qui est une RAM statique (SRAM) des registres spécialisés.

Description détaillée pour l'AVR (ATTiny861 en figure ci-après) : la partie basse est aussi appelée Register File et commence en adresse 0x20. Puisque l'on a 64 registres on s'arrête à l'adresse 0x5F et c'est là que commence la RAM (1ko pour ATMega8 et ATMega16 et 2ko pour ATMega328p).

Donnons en une description simplifiée à l'aide d'une figure

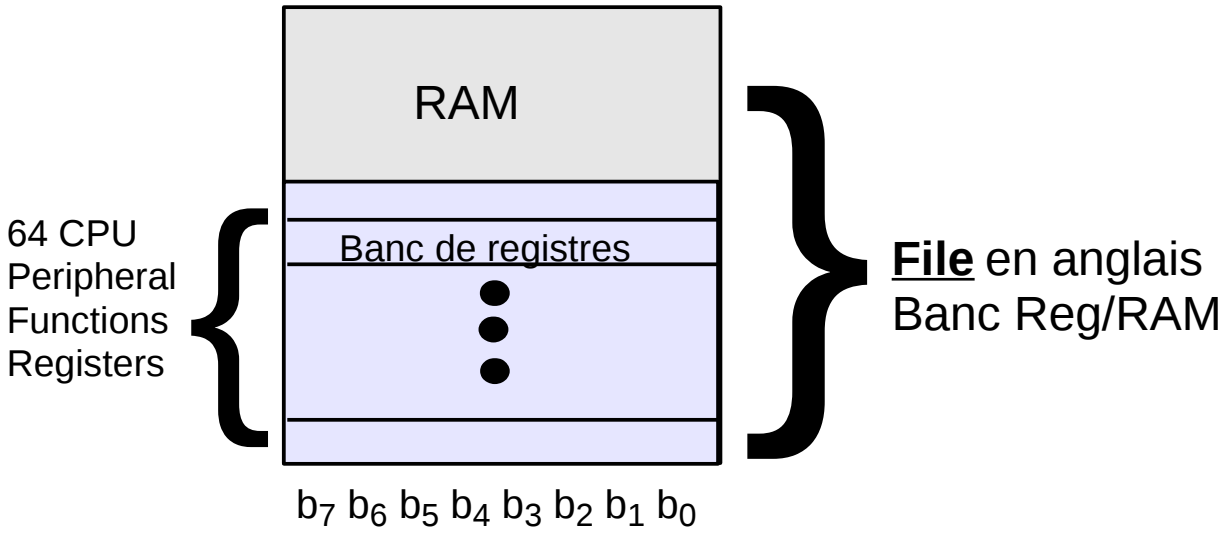


Figure 13: Banc de registres et RAML

2°) Quelques registres spécialisés de l'AVR (ATTiny861)

Nous présentons quelques registres spéciaux dans un tableau :

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x3F(0x5F)	SREG	I	T	H	S	V	N	Z	C
0x32(0x52)	TCTN0L	Timer0 8 bits poids faible							
0x2F (0x4F)	TCCR1B					CS13	CS12	CS11	CS10
0x2E (0x4E)	TCTN1	Timer1 8 bits							
0x2D (0x4D)	OCR1A	Comparaison Timer1 8 bits							
0x2C (0x4C)	OCR1B	Comparaison Timer1 8 bits							
0x1B(0x3B)	PORTA	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
0x1A(0x3A)	DDRA	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
0x19(0x39)	PINA	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
0x18(0x38)	PORTB	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
0x17(0x37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
0x16(0x36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
0x14(0x34)	TCNT0H	Timer0 8 bits : poids fort							
0x05 (0x25)	ADCH								
0x04 (0x24)	ADCL								
0x03 (0x23)	UDR	Registre de données USART I/O							
0x02 (0x22)	UCSRA	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM
0x01 (0x21)	UCSRB	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8
0x00(0x20)	TCCR1E	--	--	OC1OE5	OC1OE4	OC1OE3	OC1OE2	OC1OE1	OC1OE0

Les registres marqués en rouge n'existent pas dans l'AVR Tiny861. En général ils gèrent la liaison série dans les autres familles. On les implantera quand même si besoin. Cela nous permettra d'utiliser des routines qui fonctionnent sur un Arduino par exemple.

Parmi les registres spéciaux, remarquez un registre appelé **SREG** (registre de statut) disposant des bits suivants :
C : retenue (Carry), Z : zéro, N : négatif, V : dépassement de capacité (overflow), S: bit de signe, H : demi-retenue, T : Bit Copy Storage et I : autorisation générale d'interruptions.

En plus de la SRAM et des 64 registres spéciaux, nous disposons de 32 registres d'usage général.

La pile est gérée en RAM par un pointeur de pile.

III) Utiliser les registres de l'AVR (ATTiny861) en C

L'idéal, pour une écriture facile en langage C, serait de pouvoir écrire :

```
1         // affectation d'une valeur dans un registre :
2         PORTB = val;
```

qui pourrait mettre directement la valeur contenue dans la variable val dans un registre appelé **PORTB**. Pour cela, il faut que le compilateur C (gcc pour nous) connaisse le mot **PORTB**. En général, ce sera le cas si vous mettez l'entête

```
1         #include <avr/io.h> // utilisera iom8.h pour ATMega8
```

en tout début de votre programme. Mais qu'en est-il pour les bits des registres ?

La subtilité dans cette inclusion est que c'est en fait le fichier « iom8.h » qui sera inclus en fait.

Le registre **SREG** présenté ci-dessus comporte des bits qui possèdent un nom. Ce n'est pas le seul registre a avoir cette propriété. Mais peut-on en déduire que les bits des registres possèdent tous un nom que le compilateur C connaît ? La réponse est oui mais probablement pas à 100% !

1°) Exemple de fichier d'en-tête pour gcc

Pour illustrer les questions de la section précédente de manière un peu plus concrète, voici un exemple partiel de fichier d'entête « iom8.h » :

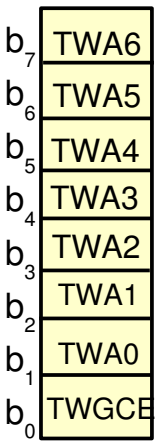
```
1         /* Port A */
2         #define PINA  _SFR_IO8(0x19)
3         #define DDRA  _SFR_IO8(0x1A)
4         #define PORTA _SFR_IO8(0x1B)
5
6         /* Port B */
7         #define PINB  _SFR_IO8(0x16)
8         #define DDRB  _SFR_IO8(0x17)
9         #define PORTB _SFR_IO8(0x18)
10        ....
11        /* PORTB */
12        #define PB7   7
13        #define PB6   6
14        #define PB5   5
15        #define PB4   4
16        #define PB3   3
17        #define PB2   2
18        #define PB1   1
19        #define PB0   0
20        ...
```

Essayez de distinguer la définition d'un registre dans la première partie de la définition d'un bit dans la deuxième partie !

2°) Les possibilités pour atteindre un bit particulier

Voici à travers un exemple comment accéder à un bit particulier pour le registre **TWAR** :

TWAR



Exemple en gcc

```
void main( void) {
....
//*** Toujours pour set *****
  TWAR |=(1<<6);
//**** Si on connaît le nom
  TWAR |=(1<<TWA5);
//*** Toujours pour reset *****
  TWAR &= ~(1<<6);
//**** Si on connaît le nom
  TWAR &= ~(1<<TWA5);
...

}
```

Fichier d'entête

```
....
/* TWAR */
#define TWA6    7
#define TWA5    6
#define TWA4    5
#define TWA3    4
#define TWA2    3
#define TWA1    2
#define TWA0    1
#define TWGCE     0
...
```

Exercice 3

Pour vérifier la bonne compréhension de ce qui vient d'être dit, nous donnons à droite le fichier d'entête des bits correspondant au registre **TWCR**.

Fichier d'entête

```
/* TWCR */
#define TWINT     7
#define TWEA      6
#define TWSTA     5
#define TWSTO     4
#define TWWC      3
#define TWEN      2
/* bit 1 reserved (TWI_TST?) */
#define TWIE      0
```

1°) Dessiner le registre correspondant
2°) En utilisant les masques du TD précédent, écrire les instructions C permettant :
- mise à 1 de TWEA
- mise à 0 de TWWC.
- tester si TWINT est à 1
- tester si TWEN est à 0

3°) Refaire le même travail en utilisant les noms des bits directement.

L'accès aux bits de registre dépend fortement du compilateur. Il n'y a aucune règle générale et c'est la première chose qu'il vous faudra apprendre pour un nouveau compilateur.

Remarquez les multiples utilités des broches de l'ATTiny861 du commerce.

IV) Comment mettre un programme en ROM/RAM

Commençons par nous poser la question de savoir comment on peut compiler un programme en langage C.

La compilation

La compilation se fait en ligne de commande. Son objectif est de réaliser un fichier

au format elf.

Voici un exemple de compilation :

```
#!/bin/bash
avr-gcc -g -mmcu=attiny861 -Wall -Os -c chenillar.c
avr-gcc -g -mmcu=attiny861 -o chenillar.elf -Wl,-Map,chenillar.map chenillar.o
avr-objdump -h -S chenillar.elf > chenillar.lss
avr-objcopy -O binary -R .eeprom chenillar.elf chenillar.bin
./mifwrite chenillar.bin chenillar.mif
```

La dernière ligne met en œuvre un convertisseur maison (dont le code source vous sera donné) qui fait une transformation en fichier .mif

Ce script réalise la compilation, la transformation du fichier .c en fichier .mif. Il est sensé fonctionner dans un sous-répertoire « soft »du projet.

La réalisation de la mémoire programme et son initialisation nécessite donc un fichier mif comme le montre l'exemple ci-dessous.

Réalisation de la mémoire programme

La mémoire programme est décrite dans un fichier appelé pm.vhd. En voici un exemple.

```
1         library IEEE;
2         use IEEE.STD_LOGIC_1164.ALL;
3         use IEEE.STD_LOGIC_ARITH.ALL;
4         use IEEE.STD_LOGIC_UNSIGNED.ALL;
5         -- for memory
6         LIBRARY lpm;
7         USE lpm.lpm_components.ALL;
8         LIBRARY altera_mf;
9         USE altera_mf.altera_mf_components.all;
10
11        entity pm is
12          Port (
13                    clk     : in std_logic;
14                    Rst     : in std_logic;
15                    PM_A    : in std_logic_vector(15 downto 0);
16                    PM_Drd  : out std_logic_vector(15 downto 0));
17        end pm;
18
19        architecture Arch of pm is
20        begin
21        progMemory: lpm_rom GENERIC MAP (
22          lpm_widthad => 12,
23          lpm_outdata => "REGISTERED",
24          --
25          lpm_address_control => "UNREGISTERED",
26          lpm_file => "./soft/chenillar.mif",-- fill ram with content of
          file program.mif
```

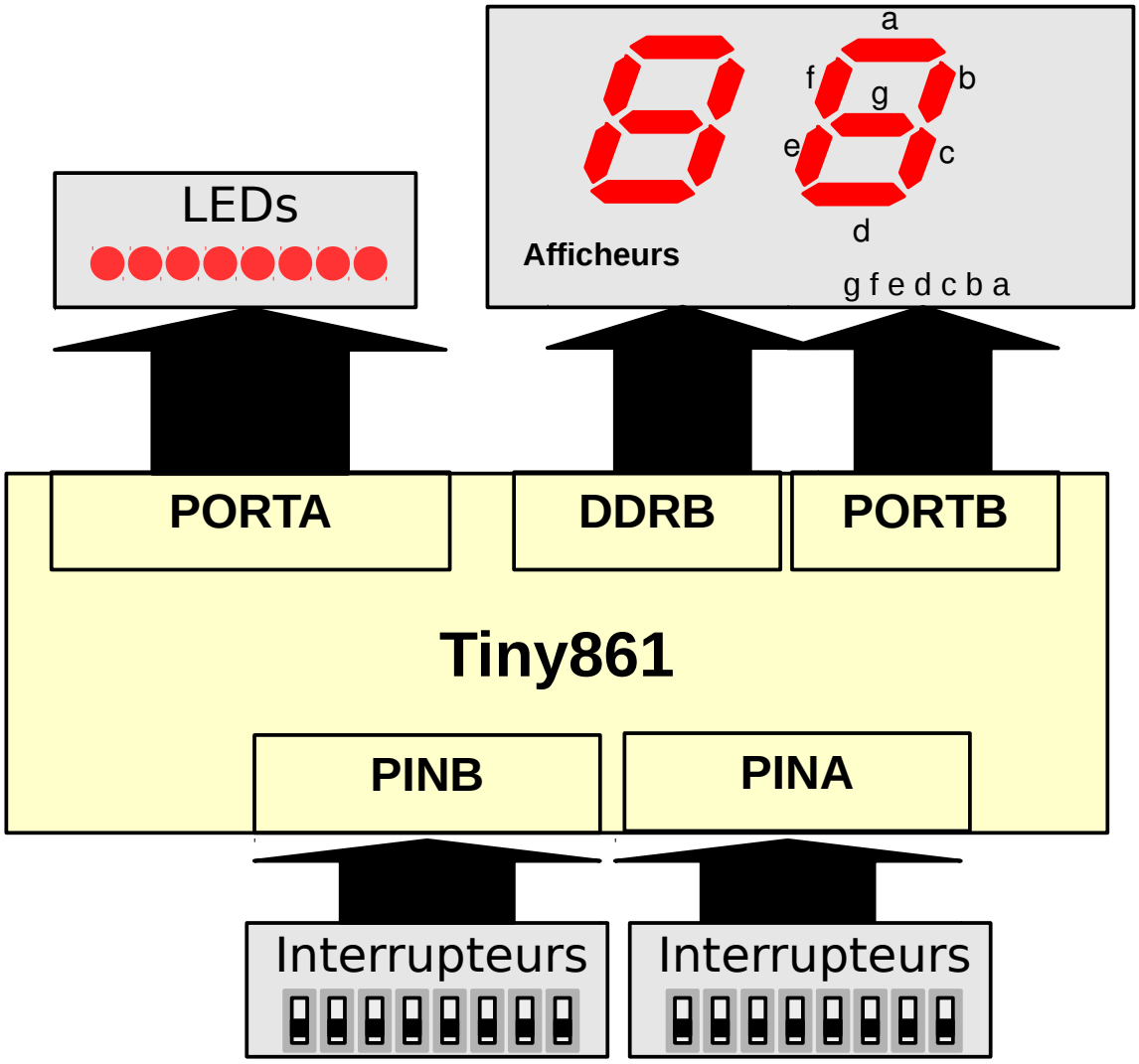
```
27          lpm_width => 16)
28          PORT MAP (
29            address => PM_A(11 downto 0), --8
30            memenab => '1',
31         --   inclock => Clk,
32            outclock => clk,
33            q => PM_Drd);
34        end Arch;
```

Cette mémoire ne fonctionne qu'à condition qu'un sous répertoire soft existe dans le projet... où l'on trouve le fichier .mif

Lorsque vous compilez un nouveau programme, après réalisation du fichier .mif, vous devez compiler de nouveau votre projet. Si vous avez choisi l'option de compilation « **smart** » cela vous prendra environ 2 mn, autrement cela vous prendra une dizaine de minute.

Exercice 4

Réaliser le programme chenillar.c si votre configuration matérielle est :



TD 6 - Interfacer un PORT en sortie

Nous allons interfacer un ou plusieurs registres en sortie que l'on appelle PORTs de sortie.

I) Le processeur et ses interfaces externes

Chaque fois que vous utilisez un processeur, demandez-vous quelles sont les interfaces externes. Les plus simples sont des PORTs, mais d'autres plus sophistiquées sont possibles : FIFO, mémoires et autres ... Par exemple, si vous vous intéressez au compteur de passage, vous pouvez demander au processeur de tout faire, c'est à dire on lui rentre deux bits d'information et il sort 8 bits d'information, les 7 bits des sept segments et le bit de sélection des afficheurs. Mais on peut aussi déporter le compteur transcodage de l'affichage à l'extérieur du processeur... ou même le compteur (deux compteurs BCD) et son transcodeur.

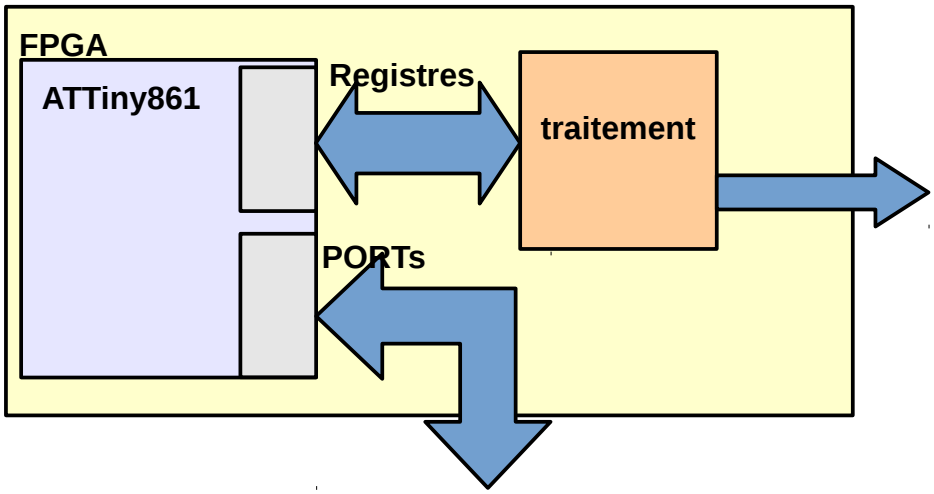


Figure 14: Distinction de principe entre PORTs et Registres

La terminologie utilisée dans les microcontrôleurs sépare la notion de PORT et la notion de registre. Un port est un registre dont les valeurs sortent directement vers l'extérieur, sur des broches. Ce n'est pas le cas des registres. Nous ne suivrons pas toujours cette terminologie ici de manière rigoureuse.

Les PORTs et registres sont décrits dans le fichier microcontroleur.vhd. Chaque fois que vous désirez modifier un ou plusieurs PORTs/registres, c'est dans ce fichier là que cela va se passer.

II) Les PORTs en sortie

Dans le fichier microcontroleur.vhd vous trouverez un certain nombre de choses et en particulier au minimum trois process. Définir des PORTs en sortie se fait dans le process présenté ci-dessous :

```
35          iowr: process(CLK)
36            begin
37                if (rising_edge(CLK)) then
38                    if (IO_wr = '1') then -- ressemble à 'en'
```

```
39                      case IO_A is
40            -- addresses for tiny861 device (use io.h).
41            --
42                          when PORTA  => -- PORTA=X"1B" (0X3B)
43                                         Led <= IO_Dwr;
44                          when  PORTB => -- PORTB=X"18" (0X38)
45                                         Aff7segs <= IO_Dwr;
46                          when  DDRB => -- DDRB=X"19" (0X39)
47                                         Diz7segs <= IO_Dwr;
48                          when others =>
49                      end case;
50                  end if;
51              end if;
52            end process;
```

où vous voyez apparaître une correspondance entre des PORTs/Registres et des numéros. Le nom des PORTs et les numéros correspondants proviennent de la documentation officielle du processeur. On n'est pas obligé de les respecter mais les respecter évite de changer le fichier avr/io.h connu du compilateur.

Vous constatez donc que le cœur de départ gère un certain nombre de PORTs : **PORTA**, **PORTB**, **DDRB** et d'autres registres qui ne nous intéressent pas pour le moment. Nous présentons l'ensemble de ce qui vient d'être dit sur un schéma ci-dessous. Tous les registres et PORTs ne sont pas dessinés et seulement le processus .

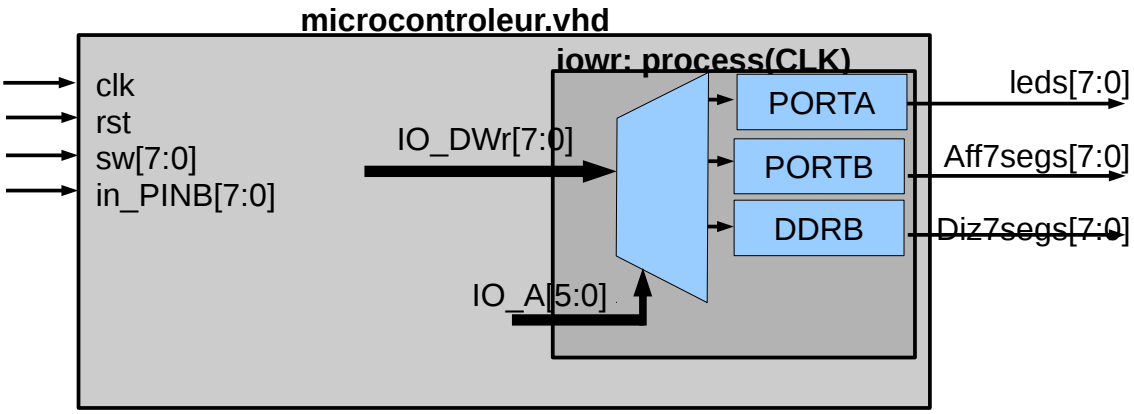


Figure 15: Schéma d'interface des PORTS de sortie de l'ATTiny861 pour nos TPs

Note : pour information CLK est à 50 MHz

Seul le processus "iowr: process(CLK)" permettant une sortie dans un PORT est dessiné : son objectif est de dispatcher "I_DIN[7:0]" dans les différents registres en fonction de la valeur de "I_ADR_IO".

Exercice 1

1°) Quelle instruction C affichera ON OFF OFF ON OFF ON ON OFF sur les 8 leds ?

2°) Quelle instruction C affichera 7 sur l'afficheur des dizaines si g est poids faible ?

III) Des registres en sorties

Comme on vient de le voir les PORTs ne nécessitent pas de signal. Si vous voulez utiliser un registre au lieu d'un PORT, il vous faudra déclarer un signal et l'utiliser ensuite :

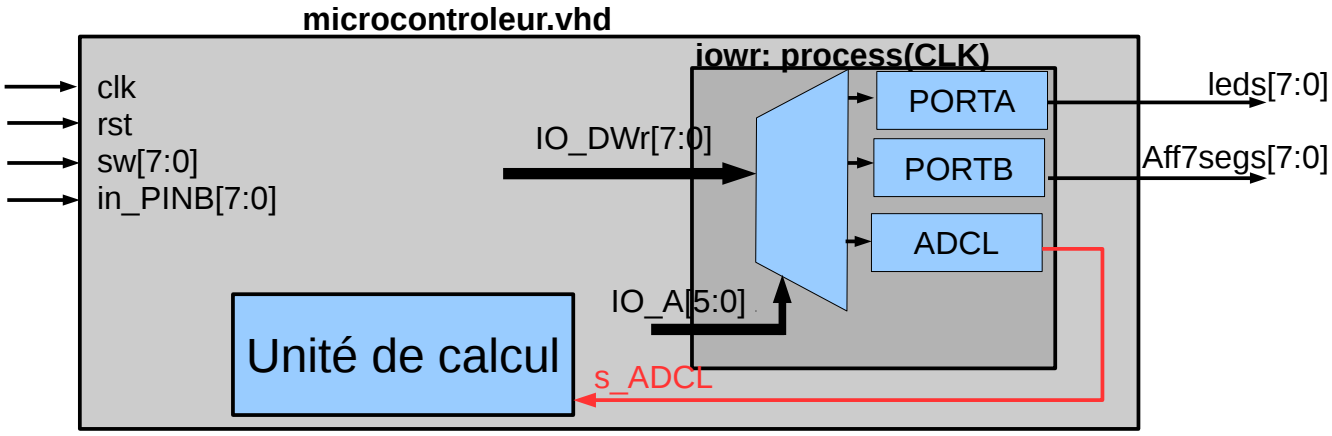


Figure 16: Schéma d'interface d'un registre ADCL en sortie pour l'ATTiny861 pour nos TP

En clair utiliser un registre en sortie consiste à :

- ajouter un signal (un fil interne comme s_ADCL ici)
- ajouter une ligne dans le case
- ajouter un PORT MAP pour l'unité de calcul qui doit se connecter à s_ADCL ici.

La réalisation des trois opérations ci-dessus créera ''s_ADCL'' comme un registre. Ce n'est donc absolument pas un simple fil. Parfois, l'unité de calcul possède déjà son propre registre d'entrée. Dans ce cas, une des techniques possible est :

- connecter directement IO_Dwr en entrée du fameux registre d'entrée
- gérer un signal d'écriture en dehors du case when avec une ligne

```
s_en <= IO_wr when (IO_A = ADCH) else '0';
```

IV) Exercices

Exercice 2

Voici un programme qui compte sans arrêt sur le PORTB et qui sort ses valeurs sur des LEDs

```
1       #include "avr/io.h"
2       #undef F_CPU
3       #define F_CPU 15000000UL
```

```
4         #include "util/delay.h"
5         void main(void) {
6           uint8_t i=0; // ou unsigned char i=0;
7           while(1) {
8             i++; // incrémentation de i
9             PORTB =  i; // on sort vers l'extérieur
10            _delay_ms(1000);
11          }
12        }
```

Note : éviter les instructions du genre "PORTB++;" Cette instruction nécessite une lecture du PORTB, une incrémentation et la sortie dans le PORTB. Or la lecture du PORTB n'est pas implémentée (mais peut l'être). Ainsi PORTB++ vous donnera un peu n'importe quoi.

1°) Faire un schéma matériel de l'ensemble FPGA et LEDs (style Figure 15)

2°) Comment changer ce programme pour qu'il compte de 3 en 3.

3°) Réaliser un chenillard double.

Exercice 3

Un ensemble de modifications a été réalisé en VHDL sur l'ATTiny861 :

```
1         -- IO write process
2         --
3         iowr: process(CLK)
4         begin
5             if (rising_edge(CLK)) then
6                 if (IO_wr = '1') then
7                     case IO_A is
8                      -- addresses for tiny861 device (use io.h).
9                    --
10                            when PORTA  => -- PORTA=X"1B" (0X3B)
11                                           Led <= IO_Dwr;
12                            when DDRA =>
13                                           an <= IO_Dwr(3 downto 0);
14                            when  PORTB => -- PORTB=X"18" (0X38)
15                                              Aff7segs <= IO_Dwr;
16                            when UCSRB =>
17                                                s_UCSRB <= IO_Dwr;
18                            when ADCL => -- managing data in CRC16 CCIT
19                                           s_data <= IO_DWr;
20                            when ADCH => --managing data status in CRT16
21                                         s_soc <= IO_Dwr(7);
22                                         s_eoc <= IO_Dwr(6);
23                            when TCNT1 => s_tcnt1 <= IO_Dwr;
24                            when TCCR1B => s_tccr1b <= IO_Dwr;
25                            when others =>
26                        end case;
27                end if;
28            end if;
29        end process;
```

1°) Réaliser un dessin (dans le style de la figure 16) correspondant au programme ci-dessus.

2°) Si l'entité correspondante (à la question 1°) est :

```
1       entity microcontroleur is
2           Port ( clk : in  STD_LOGIC;
3                  Rst : in  STD_LOGIC;
4                  sw : in STD_LOGIC_VECTOR (7 downto 0);
5                  In_PINB : in STD_LOGIC_VECTOR (7 downto 0);
6                  serial_in : in std_logic;
7                  Led : out  STD_LOGIC_VECTOR (7 downto 0);
8                  Serial_out : out std_logic;
9                  Aff7segs : out  STD_LOGIC_VECTOR (7 downto 0);
10                 an : out STD_LOGIC_VECTOR (3 downto 0));
11      --                    O_reg_pc : out std_logic_vector(15 downto 0));
12      end microcontroleur;
```

Pouvez vous alors préciser quels sont les PORTs et quels sont les registres.

3°) Comment modifier le code C pour que les bits du registre **ADCH** gardent les noms du VHDL (trouver les signaux correspondants dans le code) ?**(voir p 29)**

4°) Comment modifier le code C pour changer de noms aux registres **ADCH** et **ADCL (voir p 29)**.

5°) Comment modifier le code VHDL pour donner aux registres de la question 4°) le même nom qu'en C ?

Info : les noms des registres sont déclarés en tout début de l'architecture de microcontroleur :

```
13      architecture microcontroleur_architecture of microcontroleur is
14      --Registres et PORTs de l'ATTiny861
15      constant TCCR1B : std_logic_vector(5 downto 0) := "101111";
16
```

Exercice 4

On dispose d'un FPGA dans lequel on a embarqué un Tiny861 comme le montre la figure ci-dessous.

Nous allons étudier le programme suivant par morceaux :

```
1       #include <avr/io.h>
2       #undef F_CPU
3       #define F_CPU 15000000UL
4       #include "util/delay.h"
5       const unsigned char
        digit7segs[16]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0
        x90,0x88,0x83,0xC6,0xA1,0x86,0x8E};
6       main(void) {
7         unsigned char cmpt=0;
8         while(1) {
9           cmpt++;
10          if ((cmpt & 0x0F) > 0x09) cmpt += 6;
11          if ((cmpt & 0xF0) > 0x90) cmpt = 0;
12          DDRB = ~digit7segs[(cmpt & 0xF0)>>4];
13          PORTB = ~digit7segs[(cmpt & 0x0F)];
```

```
14            _delay_ms(500);
15          }
16          return 0;
17        }
```

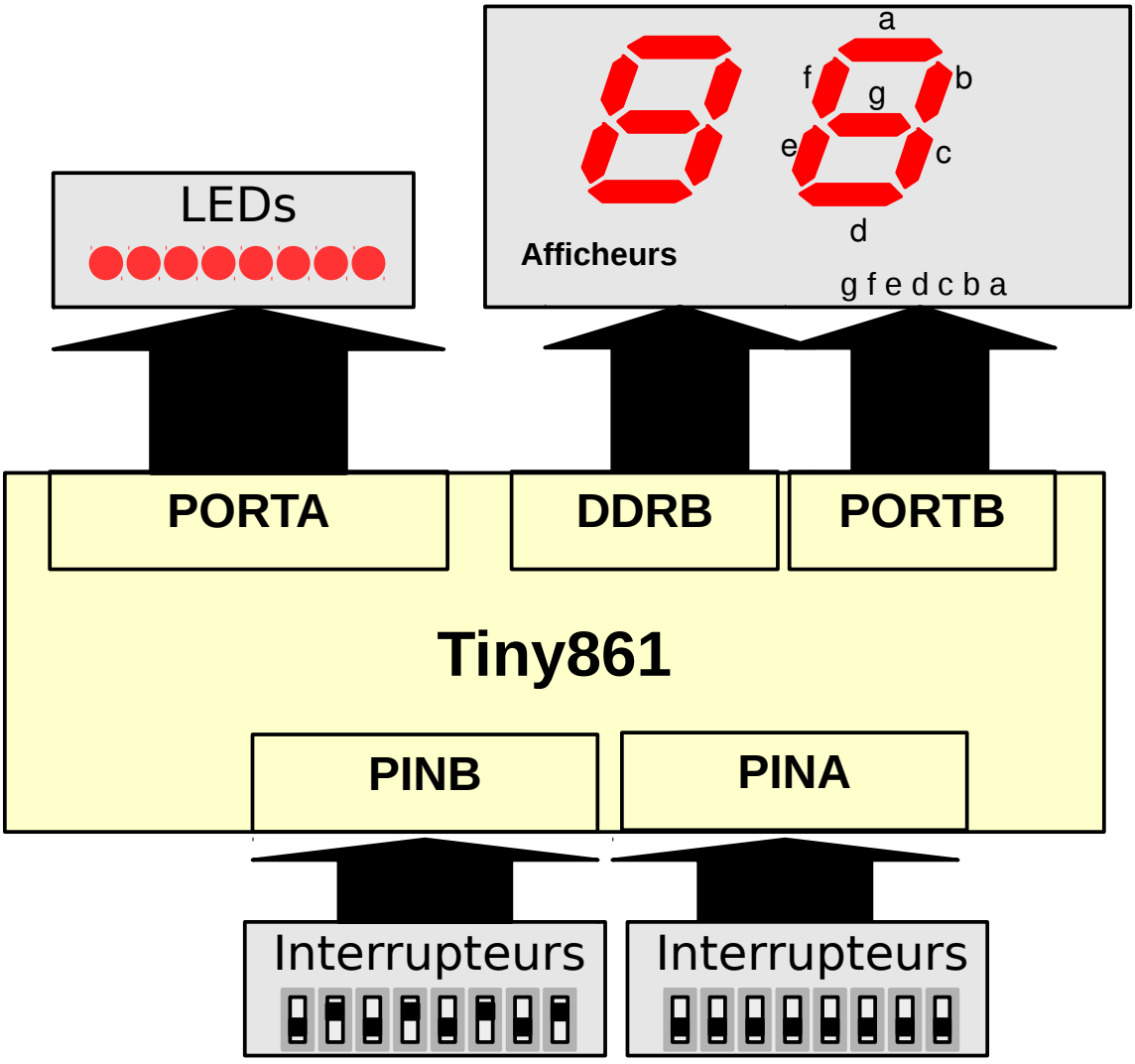


Figure 17: Câblage du Tiny861 enfoui dans le FPGA

Les afficheurs sont à cathode commune et que chacun est piloté par un PORT 8 bits (**PORTB** pour les unités et **DDRB** pour les dizaines).

1°) Que fait l'instruction

```
PORTB = ~digit7segs[(cmpt & 0x0F)];
```

pour les valeurs de cmpt = 0x00, 0x01, 0x02,... sur l'afficheur des unités ? Expliquez alors la présence du tilde ''~''.

2°) Que fait l'instruction

```
PORTB = ~digit7segs[(cmpt & 0xF0)>>4];
```

pour les valeurs de cmpt = 0x00, 0x10, 0x20,... sur l'afficheur des dizaines ?

3°) Pour les valeurs de cmpt 0x29 et 0x99, que font les instructions :

```
1        cmpt++;
2        if ((cmpt & 0x0F) > 0x09) cmpt += 6;
3        if ((cmpt & 0xF0) > 0x90) cmpt = 0;
4        DDRB = ~digit7segs[(cmpt & 0xF0)>>4];
5        PORTB = ~digit7segs[(cmpt & 0x0F)];
```

4°) Écrire les deux sous-programmes incrementBCD() et decrementBCD()

Exercice 5

Comment réaliser une interface avec deux compteurs/décompteurs décimaux cascadés sur donc 8 bit, tel que l'écriture d'un '1' dans **PORTB** (mal choisi, pourquoi ?) incrémente les deux compteurs et l'écriture d'un '0' dans ce même PORT décrémente. Ceci sera réalisé en TP.

TD 7 - Interfacer des PORTs/Registres en entrée

Nous allons nous intéresser dans ce TD aux PORTs/Registres en entrée. Leur manipulation se fait toujours dans le fichier microcontroleur.vhd mais dans un autre « process » que celui étudié au TD précédent.

Les PORTs en entrée

Les PORTs en entrée se gèrent dans le process :

```
1         -- IO read process
2         --
3         iord: process(IO_rd,IO_A,In_PINB,sw)
4         begin
5             -- addresses for tinyX6 device (use iom8.h).
6             --
7             if IO_rd = '1' then
8               case IO_A is
9                     when  PINA => IO_Drd <= sw;  -- PINA=X"19" (0X39)
10                    when  PINB => IO_Drd <= In_PINB;  -- PINB=X"16" (0X36)
11                    when others => IO_Drd <= X"AA";
12              end case;
13            end if;
14        end process;
```

Nous allons maintenant représenter ce morceau de code VHDL de manière schématique.

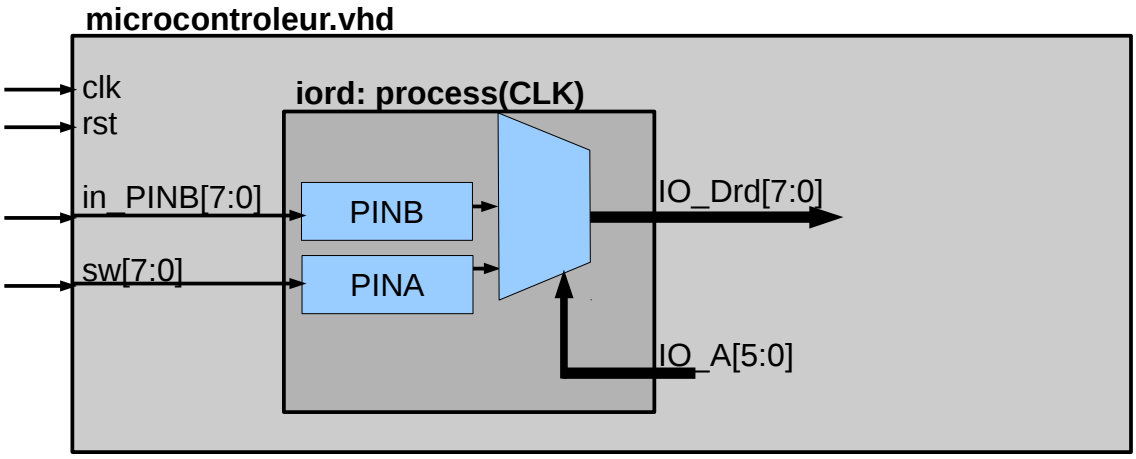


Figure 18: Interfacer des PORTs en entrée

Exercices

Trois exercices sur les PORTs en entrée sont proposés suivis d'un exercice sur les PORTs en sortie.

Exercice 1

1°) Que donne l'instruction toto=PINB ; (voir Figure 17) ? (résultat en binaire et en hexadécimal)

2°) Qu'est-ce qui est affiché sur les LEDs après exécution des insrtructions

```
1        toto = PINB ;
2        PORTA = ~toto ; // (voir Figure 17)
```

Exercice 2

Reprendre l'exercice 1 du TD précédent et le modifier pour que le chenillard réalisé dépende de la valeur de deux boutons poussoirs. Faites un schéma de principe pour le câblage de vos boutons et rappeler ce qu'un fichier de contrainte définit dans ce cas.

Exercice 3

Un FIFO est un composant dont une donnée est toujours accessible en lecture. Mais une fois cette donnée lue, il faut le lui faire savoir : une entrée spécifique est là pour cela (read_buffer en figure 19). **Le problème à résoudre est donc de ne pas réaliser ce signal trop tôt : je lis d'abord et j'évacue ensuite.**

Quelques informations supplémentaires sur le programme VHDL sont présentées :

```
1        -- utilisé dans le registre PINC en lecture
2        --      b7     b6 b5
3        signal s_rxc, s_full : std_logic;
```

Les signaux ci-dessus sont utilisés dans le process de lecture :

```
1        -- IO read process
2        --
3        iord: process(IO_rd,IO_A,In_PINB,sw)
4        begin
5        -- addresses for tinyX6 device (use iom8.h).
6        --
7          if IO_rd = '1' then
8             case IO_A is
9                when  PINA => IO_Drd <= sw;  -- PINA=X"19" (0X39)
10               when PINB => IO_Drd <= In_PINB;  -- PINB=X"16" (0X36)
11               when  PORTB => IO_Drd <= s_PORTB;
12               when UCSRA => IO_Drd <= s_rxc & not (s_full) & not(s_full) &
         "00000";
13               when UDR    => IO_DRD <= s_in_UDR;
14               when others => IO_Drd <= X"AA";
15            end case;
16         end if;
17       end process;
```

Il y a un autre point sur lequel il nous faut revenir : la construction du signal read qui va au FIFO. Il se fait avec le simple code :

```
1        L_RD_FIFO <= I_RD_IO when (I_ADR_IO = PINA) else '0'; -- read  PINA
```

Avec ce signal, toute lecture de **UDR** en C provoquera lecture puis une évacuation de la donnée du FIFO : c'est un décalage dans le FIFO grâce à ce signal qui permet de récupérer la donnée suivante la prochaine fois qu'on viendra le lire.

1°) Qu'est-ce qu'un FIFO 16x8 ?

2°) Comment attendriez-vous en C que le FIFO ait une donnée présente ?

3°) Comment attendriez-vous en C que le FIFO soit complètement plein ?

4°) Comment lire une donnée en provenance de la RS232 (par data_out) ?

5°) Comment transformer le nombre hexadécimal lu deux caractères ASCII ?

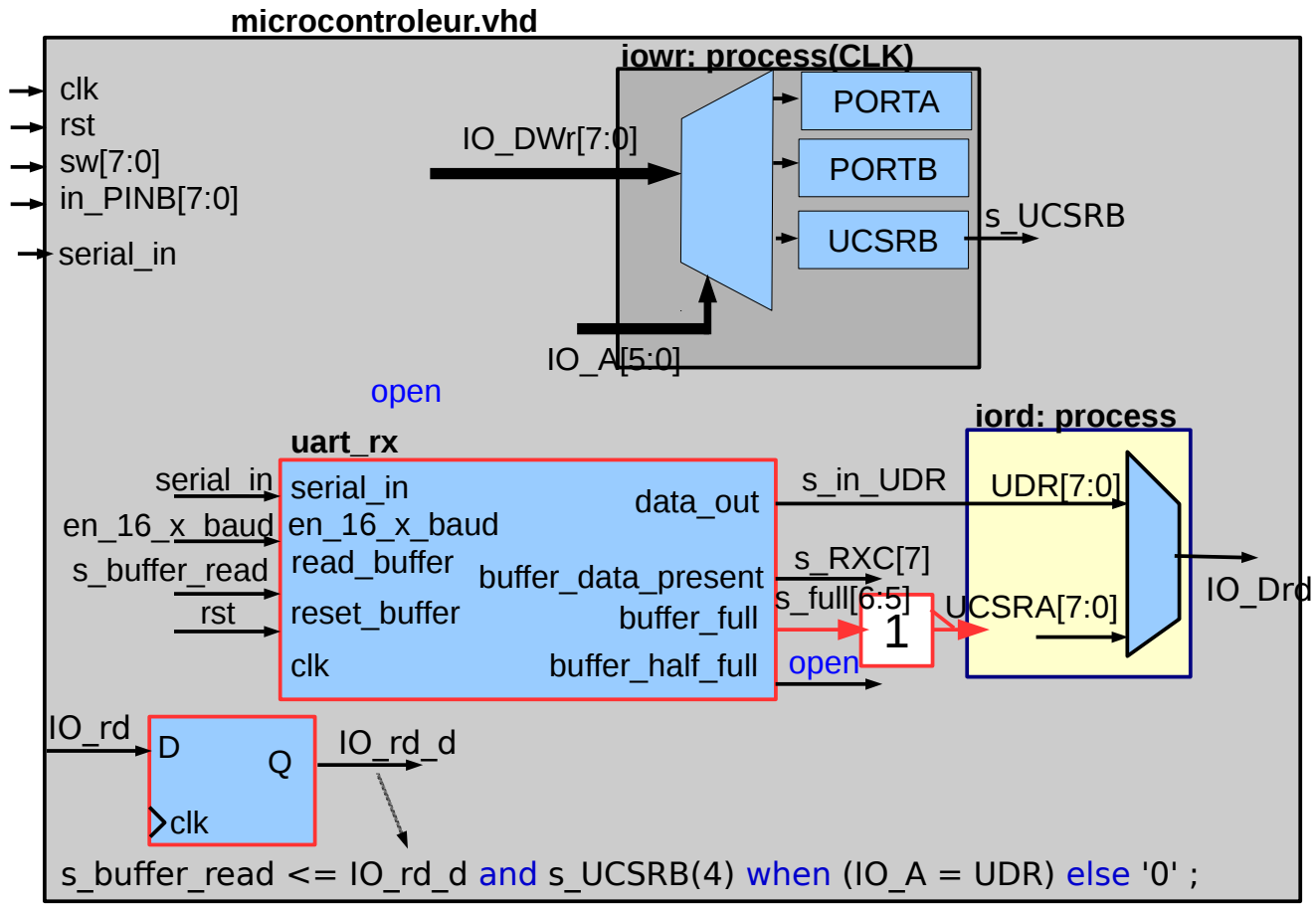


Figure 19: Interfacer une liaison série en lecture

Exercice 4

Nous allons réaliser un périphérique timer capable de se mettre en marche et de s'arrêter en vue de comparer, par exemple, le temps utilisé par deux calculs matériels et logiciels.

Indication 1 :

Nous allons partiellement réaliser le timer 1.

Partiellement car il est normalement sur 10 bits et nous n'en garderons que 8

Partiellement car nous nous contenterons d'une division sur 3 bits (du registre **TCCR1B**) contre 4 bits dans celui du commerce. Seules les divisions par 0, 1, 2, 4, 8, 16, 32 et 64 seront implantées. Il manque donc les divisions par 128, 256, 512, 1024, 2048, 4096, 8192 et 16384.

Partiellement car nous n'implémenterons ni le dépassement de capacité ni les comparaisons....

Partiellement car seuls **TCNT1** (0x2E) et **TCCR1B** (0x2F) seront implémentés.

Tout ce qui est en rouge dans la figure ci-après ne sera pas implémenté

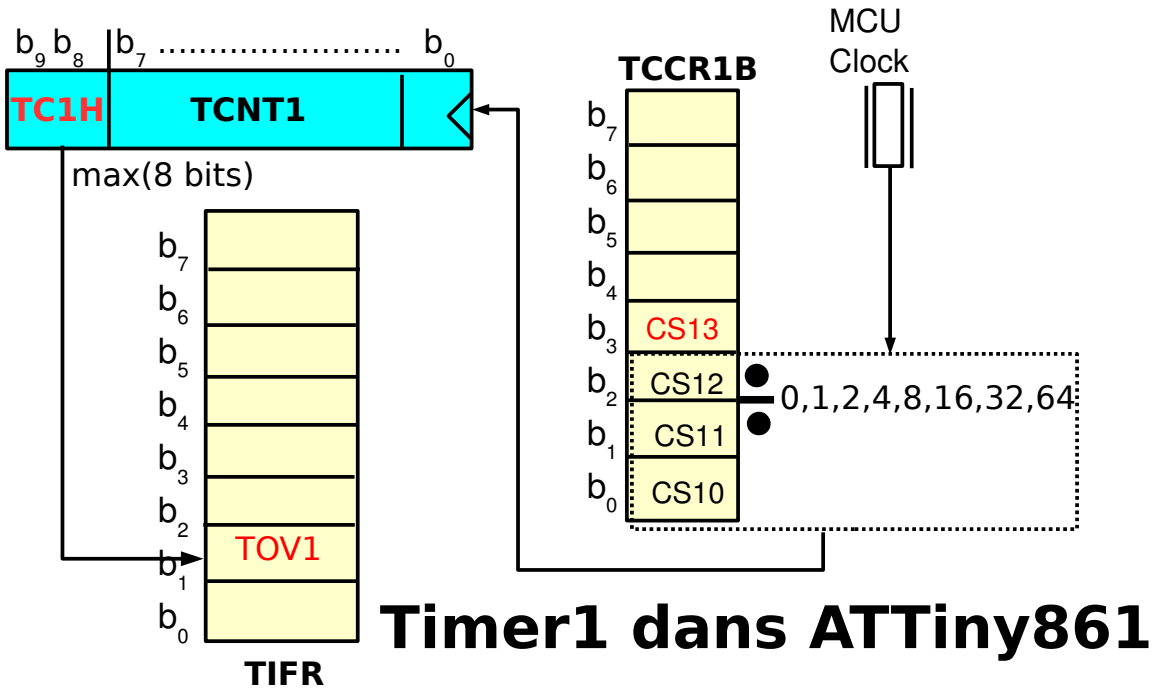


Figure 20: le timer 1 à implanter

Indication 2 :
On prendra soin de définir les deux constantes **TCNT1** et **TCCR1B**. La gestion d'écriture dans TCNT1 est un peu particulière (car multi-source):

```
s_wr_TCNT1 <= IO_wr when IO_A = TCNT1 else '0';
```

Le timer pourra être un simple process dans "microcontroleur.vhd"

```
1         process(clk) begin
2           if rising_edge(clk) then
3             if s_wr_TCNT1 = '1' then
4              s_s_tcnt1 <= s_tcnt1; -- s_tcnt1 vient du process iowr (à ajouter)
5             elsif s_en_timer1 = '1' then
6              s_s_tcnt1 <= s_s_tcnt1 + 1;
7             end if;
8            end if;
9          end process;
```

1°) Dessiner la partie écriture dans **TCNT1** et la partie lecture de **TCNT1** avec le formalisme de des figures 15 et 18. Vous complétez ensuite correctement les deux process iowr et iord en

VHDL.

2°) Le « s_en_timer1 » est réalisé par un préscaler commandé par trois bits. Pouvez-vous donner son code VHDL ?

3°) Un programme d'essai est donné maintenant :

```
1         int main (void) {
2            unsigned char chaine[20],i,j,ta;
3            unsigned int temps;
4            crc crc_chaine;
5            usart_init();
6         // configuration du timer 1
7            TCCR1B = 0x07; // demarrage avec prescaler à ??
8            while(1) {
9              i = 0;
10             usart_puts("Entrez une chaine");
11             do {
12               chaine[i] = usart_receive();
13               usart_send(chaine[i]);
14               i++;
15             } while (chaine[i-1] != 0x0D);
16             // ajout des 16 bits à 0 :
17             chaine[i-1] = 0x00;
18             chaine[i] = 0x00;
19             TCNT1 = 0;
20             crc_chaine = crcSlow(chaine, i+1); // un calcul logiciel
21             ta=TCNT1;
22             usart_puts_hexa(crc_chaine);
23             usart_puts("temp(soft)=");usart_puts_hexa(ta);
24             // preparation matérielle
25             TCNT1 = 0;
26             ADCH = 0x80; // reset
27             ADCH = 0x00; // le même calcul matériel démarre ici
28             for (j=0;j<i+1;j++) {
29               ADCL = chaine[j];  // data_valid <- '1' automatiquement
30             }
31             crc_chaine = ADCH;
32             crc_chaine <<= 8;
33             crc_chaine += ADCL;
34             ta=TCNT1;
35             usart_puts(" : ");usart_puts_hexa(crc_chaine);
36             usart_puts(" temp(hard)=");usart_puts_hexa(temps);
37             usart_send(10);usart_send(13);
38             _delay_ms(500); // on est pas pressé
39           }
40           return 0;
41        }
```

Quelle est la valeur du prescaler ? Comment récupère-t-on le temps écoulé ? Comment voyez-vous que le calcul (non précisé) donne un résultat sur 16 bits ?

Corrections partielles TD1

Exo1

La droite de la table de vérité est la somme binaire des 3 entrées :

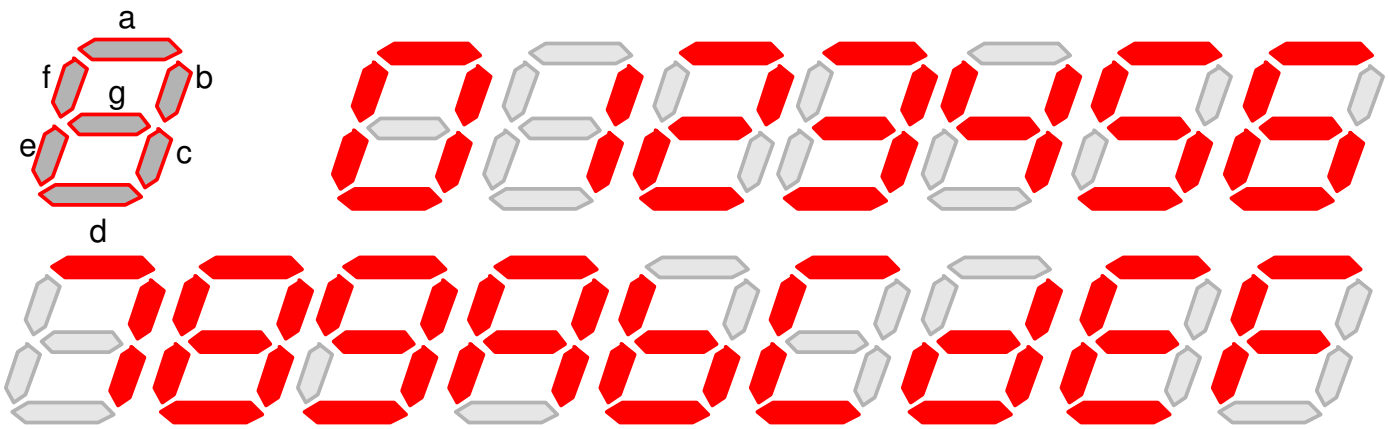
rn	b	a	rn0	s
e(2)	e(1)	e(0)	s(1)	s(0)
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

```
ENTITY adder IS PORT(
  e : in BIT_VECTOR(2 DOWNTO 0);-- 3 entrées
  s : out BIT_VECTOR(1 DOWNTO 0));
  -- 2 sorties
END demo;
ARCITECTURE aAdder of adder IS BEGIN
  With e select
    s <= "00" WHEN "000",
         "01" WHEN "001",
         "01" WHEN "010",
         "10" WHEN "011",
         "01" WHEN "100",
         "10" WHEN "101",
         "10" WHEN "110",
         "11" WHEN OTHERS;
END aAdder;
```

Solution exo2 TD1

```
library ieee;
use ieee.std_logic_1164.all;
ENTITY adder IS PORT(
  e : in std_logic_VECTOR(2 DOWNTO 0);   -- 3 entrées
  s : out std_logic_VECTOR(1 DOWNTO 0)); -- 2 sorties
END demo;
ARCITECTURE aAdder of adder IS BEGIN
  With e select
    s <= "00" WHEN "000",
         "01" WHEN "001",
         "01" WHEN "010",
         "10" WHEN "011",
         "01" WHEN "100",
         "10" WHEN "101",
         "10" WHEN "110",
         "11" WHEN OTHERS;
END aAdder;
```

Solution exo3 TD1



1°) Ecrire la table de vérité pour un afficheur 7 segments cathode commune ('1' pour allumer)

e(3)	e(2)	e(1)	e(0)	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
1	0	1	0	1	1	1	0	1	1	1
1	0	1	1	0	0	1	1	1	1	1
1	1	0	0	1	0	0	1	1	1	0
1	1	0	1	0	1	1	1	1	0	1
1	1	1	0	1	0	0	1	1	1	1
1	1	1	1	1	0	0	0	1	1	1

2°) Voici le programme VHDL partiel (décimal seulement) du transcodeur

```
library ieee;
use ieee.std_logic_1164.all;
ENTITY transcod IS PORT(
  e : in std_logic_VECTOR(3 DOWNTO 0);    -- 4 entrées
```

```
  s : out std_logic_VECTOR(6 DOWNTO 0)); -- 6 sorties
END transcod;
ARCITECTURE arch_transcod of transcod IS BEGIN
  With e select
    s <= "1111110" WHEN "0000",
         "0110000" WHEN "0001",
         "1101101" WHEN "0010",
         "1001111" WHEN "0011",
         "0110011" WHEN x"4",
         "1011011" WHEN "0101",
         "1011111" WHEN "0110",
         "1110000" WHEN "0111",
         "1111111" WHEN "1000",
         "1111011" WHEN "1001",
         "0000001" WHEN OTHERS;
END arch_transcod;
```

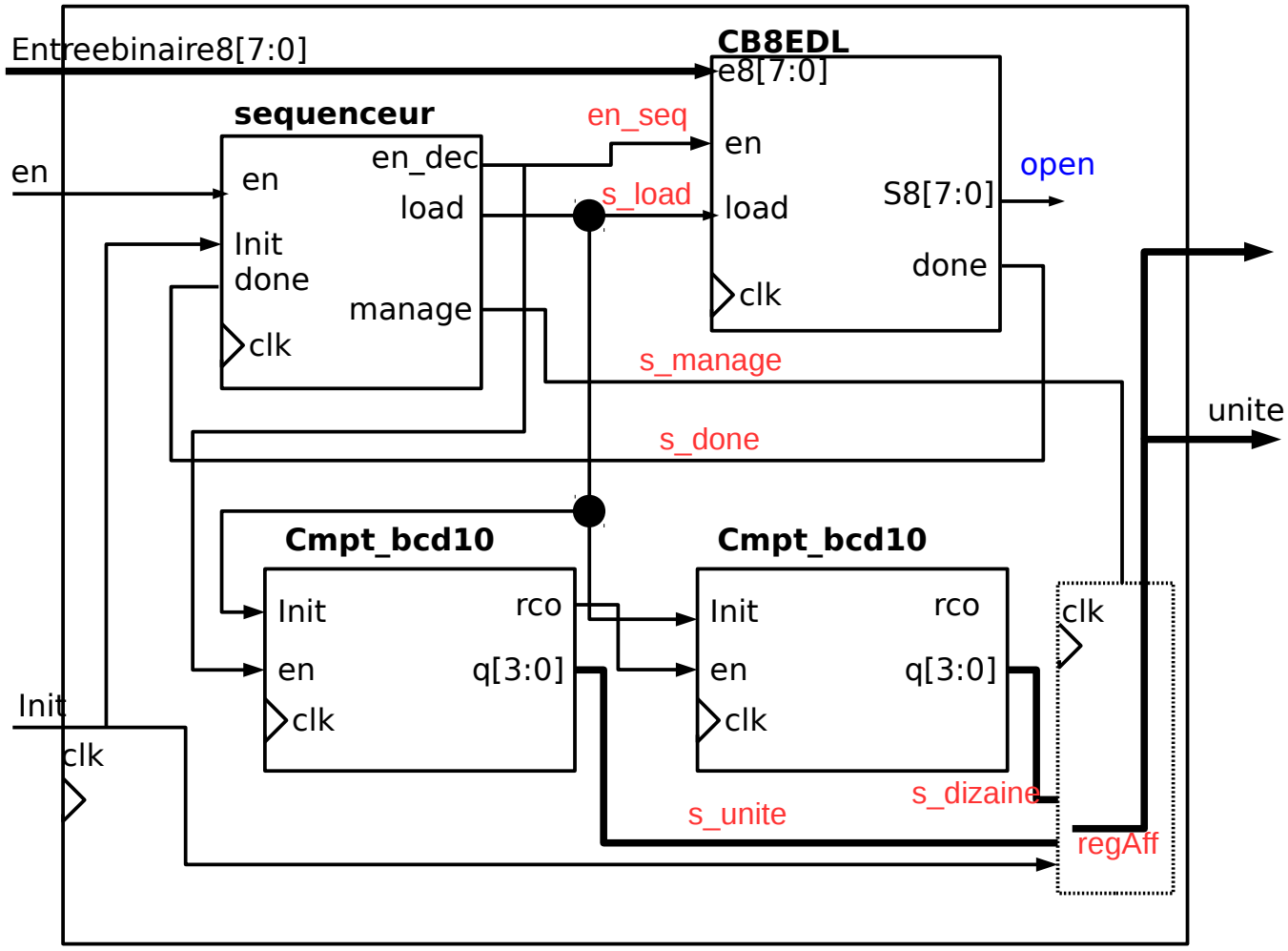
Corrections partielles TD2

Exo1

```
1         library ieee;
1         use ieee.std_logic_1164.all;
2         ENTITY Fct IS -- entité globale
3         PORT(e0,e1,e2 : IN std_logic;
4           s : OUT std_logic);
5         END Fct;
6
7         ARCHITECTURE truc OF Fct IS
8         -- signaux et composants avant le begin de l'architecture
9         SIGNAL e0e1,e2bar : std_logic;
10        COMPONENT et
11        PORT(e0,e1 : IN std_logic;
12          s : OUT std_logic);
13        END COMPONENT;
14        COMPONENT ou
15        PORT(e0,e1 : IN std_logic;
16          s : OUT std_logic);
17        END COMPONENT;
18        COMPONENT inverseur
19        PORT(e : IN std_logic;
20          s : OUT std_logic);
21        END COMPONENT;
22        BEGIN
23          i1:OU PORT MAP(e0=>e0,e1=>e1,s=>e0e1);
24          i2:inverseur PORT MAP(e=>e2,s=>e2bar);
25          i3:et PORT MAP(e0=>e0e1,e1=>e2bar,s=>s);
26        END truc;
27        -- fin de l'architecture globale
28        library ieee;
29        use ieee.std_logic_1164.all;
30        ENTITY et IS
31        PORT(e0,e1 : IN std_logic;
32          s : OUT std_logic);
33        END et;
34        library ieee;
35        use ieee.std_logic_1164.all;
36        ENTITY ou IS
37        PORT(e0,e1 : IN std_logic;
38          s : OUT std_logic);
39        END ou;
40        library ieee;
41        use ieee.std_logic_1164.all;
42        ENTITY inverseur IS
43        PORT(e : IN std_logic;
44          s : OUT std_logic);
45        END inverseur;
46        ARCHITECTURE aet OF et IS
47        BEGIN
48          s<=e0 AND e1;
49        END aet;
50        ARCHITECTURE aou OF ou IS
51        BEGIN
52          s<=e0 OR e1;
53        END aou;
54        ARCHITECTURE ainv OF inverseur IS
55        BEGIN
```

```
56          s<= NOT e;
57        END ainv;
```

Exo3



Exo4

D'après exo3 TD1 on trouve pour :

- segment a <-> INIT => x"D7ED" à mettre dans generic map
- segment b <-> INIT => x"279F" à mettre dans generic map
- segment c <-> INIT => x"2FFB" à mettre dans generic map
- segment d <-> INIT => x"7B6D" à mettre dans generic map
- segment e<-> INIT => x"FD45" à mettre dans generic map
- segment f <-> INIT => x"DF71" à mettre dans generic map
- segment g <-> INIT => x"EF7C" à mettre dans generic map

Voici le programme complet :

```
1         -- Xilinx specific
2         library IEEE;
3         use IEEE.STD_LOGIC_1164.ALL;
4         library unisim;
```

```
5         use unisim.vcomponents.all;
6         ENTITY transcodeur IS PORT(
7           e : in STD_LOGIC_VECTOR(3 DOWNTO 0);   -- 4 entrées
8           s : out STD_LOGIC_VECTOR(6 DOWNTO 0)); -- 7 sorties
9         END transcodeur;
10        ARCHITECTURE atranscodeur OF transcodeur IS BEGIN
11        i1 : LUT4 -- segment a
12          generic map (INIT => X"D7ED")
13          port map( I0 => e(0),
14                    I1 => e(1),
15                    I2 => e(2),
16                    I3 => e(3),
17                     O => s(0) );
18        i2 : LUT4 -- segment b
19          generic map (INIT => X"279F")
20          port map( I0 => e(0),
21                    I1 => e(1),
22                    I2 => e(2),
23                    I3 => e(3),
24                     O => s(1) );
25        i3 : LUT4 -- segment c
26          generic map (INIT => X"2FFB")
27          port map( I0 => e(0),
28                    I1 => e(1),
29                    I2 => e(2),
30                    I3 => e(3),
31                     O => s(2) );
32        i4 : LUT4 -- segment d
33          generic map (INIT => X"7B6D")
34          port map( I0 => e(0),
35                    I1 => e(1),
36                    I2 => e(2),
37                    I3 => e(3),
38                     O => s(3) );
39        i5 : LUT4 -- segment e
40          generic map (INIT => X"FD45")
41          port map( I0 => e(0),
42                    I1 => e(1),
43                    I2 => e(2),
44                    I3 => e(3),
45                     O => s(4) );
46        i6 : LUT4 -- segment f
47          generic map (INIT => X"DF71")
48          port map( I0 => e(0),
49                    I1 => e(1),
50                    I2 => e(2),
51                    I3 => e(3),
52                     O => s(5) );
53        i7 : LUT4 -- segment a
54          generic map (INIT => X"EF7C")
55          port map( I0 => e(0),
56                    I1 => e(1),
57                    I2 => e(2),
58                    I3 => e(3),
59                     O => s(6) );
60        end atranscodeur;
```

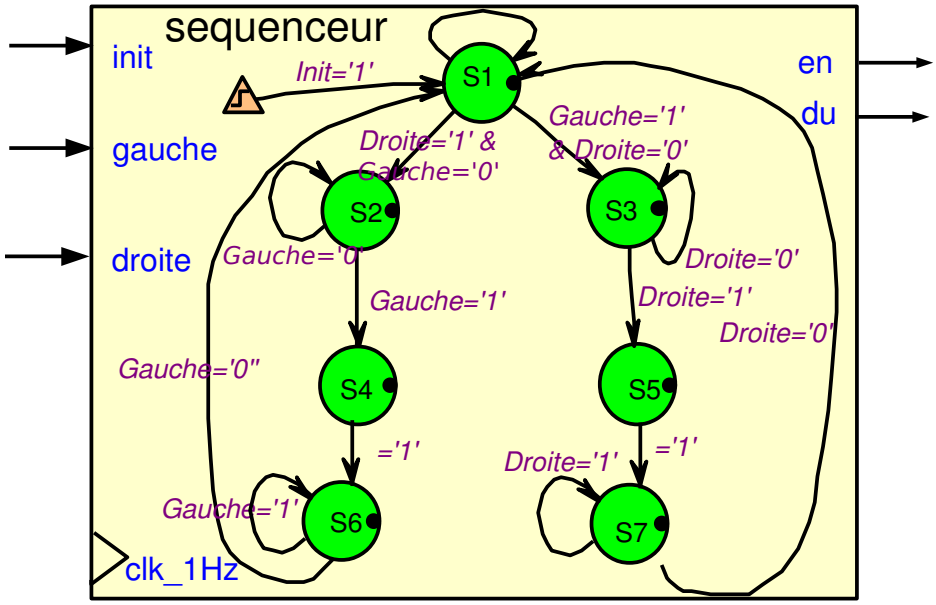
Corrections partielles TD3

Solution exo1 TD3

Utiliser la symétrie du code Gray pour le retrouver sur 3 bits

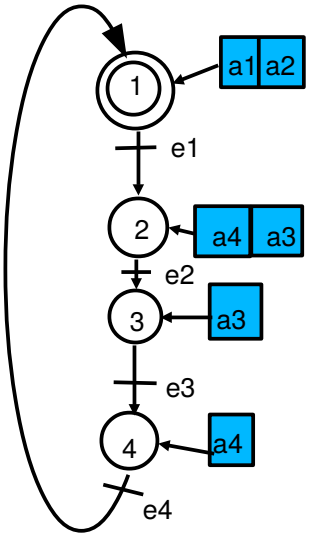
```
1         -- compteur Gray
2         ENTITY cmptGray IS PORT (
3         clock: IN BIT;
4         q : OUT BIT_VECTOR(2 DOWNTO 0));--conforme aux conseils Xilinx
5         END cmptGray;
6         ARCHITECTURE mydemo OF cmptGray IS
7           SIGNAL s_q : BIT_VECTOR(2 DOWNTO 0);
8           BEGIN
9              PROCESS(clock) BEGIN
10               IF clock'EVENT AND clock='1' THEN
11                CASE s_q IS  --style case when
12                  WHEN "000" => s_q <="001";
13                  WHEN "001" => s_q <="011";
14                  WHEN "011" => s_q <="010";
15                  WHEN "010" => s_q <="110";
16                  WHEN "110" => s_q <="111";
17                  WHEN "111" => s_q <="101";
18                  WHEN "101" => s_q <="100";
19                  WHEN OTHERS => s_q <="000";
20                END CASE;
21              END IF;
22            END PROCESS;
23          q <= s_q;
24        END mydemo;
```

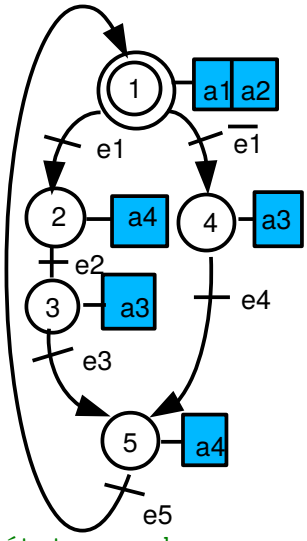
Solution exo2 TD3



Solution partielle exo3 TD3

AC1 = x4.e4
AC2 = x1.e1
AC3 = x2.e2
AC4 = x3.e3
D1 = e1
D2 = e2
D3 = e3
D4 = e4
x1+ = x4e4+x1./e1+Init
x2+ = (x1.e1+x2./e2)/Init
x3+ = (x2.e2+x3./e3)/Init






```
1         -- programme VHDL correspondant au graphe d'états gauche
2         library ieee;
3         use ieee.std_logic_1164.all;
4         ENTITY graf1 IS
5         PORT (I,e1,e2,e3,e4,clk : IN std_logic;
6               a1,a2,a3,a4 : OUT std_logic);
7         END graf1;
8         ARCHITECTURE agraf1 OF graf1 IS
9         SIGNAL x1,x2,x3,x4 : std_logic;
10        BEGIN
11          PROCESS(clk) BEGIN
12            IF (clk'event AND clk='1') THEN
13              x1 <= (x4 AND e4) OR (x1 AND NOT e1) OR I;
14              x2 <= (x1 AND e1 AND NOT I) OR (x2 AND NOT e2 AND NOT I);
15              x3 <= (x2 AND e2 AND NOT I) OR (x3 AND NOT e3 AND NOT I);
16              x4 <= (x3 AND e3 AND NOT I) OR (x4 AND NOT e4 AND NOT I);
17            END IF;
18          END PROCESS;
19          a1 <= x1;
20          a2 <= x1;
21          a3 <= x2 OR x3;
22          a4 <= x2 OR x4;
23        END agraf1;
```

Sans les équations de récurrences comme demandé :

```
1         library IEEE;
2         use IEEE.STD_LOGIC_1164.ALL;
3         ENTITY agraf1 IS
4           PORT(
5                 clock,e1,e2,e3,e4,Init :IN std_logic;
6                 a1,a2,a3,a4 :OUT std_logic
7                 );
8         END Agraf1;
9         ARCHITECTURE ar  OF Alarm IS
10          TYPE typetat IS (S1, S2, S3, S4);
11          SIGNAL etat : typetat;
12          BEGIN
13            PROCESS (clock,etat) BEGIN  -- partie séquentielle
14                 IF Clock ='1' AND Clock'EVENT THEN
15                       CASE etat IS
16                          WHEN S1 => IF e1 ='1' THEN etat <= S2;
```

```
17                                  ELSE etat <= S1;
18                                END IF;
19                          WHEN S2 => IF e2 = '1' THEN
20                                      etat <= S3;
21                                      ELSE etat <= S2;
22                                    END IF;
23                          WHEN S3 => IF e3 ='1' THEN
24                                       etat <= S4;
25                                      ELSE etat <= S3;
26                                      END IF;
27                          WHEN S4 => IF e4 ='1' THEN
28                                       etat <= S1;
29                                      ELSE etat <= S4;
30                                      END IF;
31                      END CASE;
32                END IF;
33          END PROCESS;
34      -- partie combinatoire
35          a1 <= '1' WHEN S1 ELSE '0';
36          a2 <= '1' WHEN S1 ELSE '0';
37          a3 <= '1' WHEN S2 ELSE
38                '1' WHEN S3 ELSE
39                '0';
40          a4 <= '1' WHEN S2 ELSE
41                '1' WHEN S4 ELSE
42                '0';
43      END ar;
```

Corrections partielles TD4

Solution exo1 TD4

Il y a une toute petite subtilité dans cet exercice car le poids 0 divise déjà par 2. Donc le poids 1 par 4 et le poids n par 2^{n+1}. Comme on veut une division par $32768=2^{15}$, il nous faut un poids de 14, d'où le programme ci-dessous :

```
1         -- diviseur de fréquence (Xilinx et Altera)
2         library ieee;
3         use ieee.std_logic_1164.all;
4         use ieee.std_logic_arith.all;
5         use ieee.std_logic_unsigned.all;
6         entity lent is
7                 port(horloge : in std_logic;
8                      h_lente : out std_logic);
9         end lent;
10
11        architecture alent of lent is
12                signal compteur : std_logic_vector(14 downto 0);
13                begin
14                --division de l'horloge par 32768
15                process(horloge) begin
16                        if(horloge'event and horloge='1') then
17                                compteur <= compteur + 1;
18                        end if;
19                end process;
20                h_lente <= compteur(14);
21        end alent;
```

Solution exo2 TD4

Voici la solution asynchrone :

```
1         PROCESS(clk,raz,set) BEGIN
2           IF raz='1' THEN
3               r_reg <= (OTHERS =>'0');
4           ELSIF set='1' THEN
5               r_reg <= (OTHERS =>'1');
6           ELSIF clk'event and clk='1' THEN
7             r_reg <= r_reg + 1;
8           END IF;
9         END PROCESS;
```

et le compteur qui compte jusqu'à 24 (compris) sur 5 bits donc :

```
1         PROCESS(clk,raz,set) BEGIN
2           IF raz='1' THEN
3               r_reg <= (OTHERS => '0');
4           ELSIF set='1' THEN
5               r_reg <= (OTHERS => '1');
6           ELSIF clk'event and clk='1' THEN
7             IF unsigned(q) <25 then
8               r_reg <= r_reg + 1;
9             ELSE
10              r_reg <= (OTHERS=>'0');
11            END IF;
12          END IF;
13        END PROCESS;
```

Solution exo3 VGA TD4

1°) fh = 25 Mhz => Th = 40 ns

2°)

temps(ns)	25600	25600+640=26240	30040	31750
Compteur1	640	656	751	794
Symboles		ZZZ+1	TTT-1	XXX

ZZZ=655 et TTT=752.

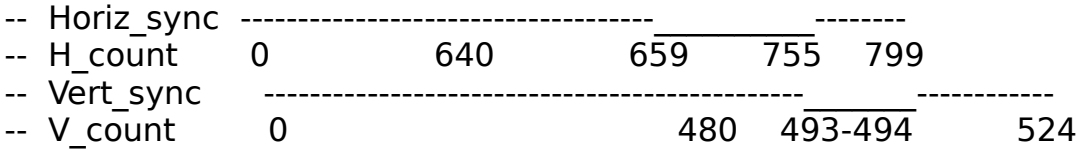
3°) Période verticale : Tv = 40 ns x 800 = 32 us et donc 15,36 ms pour 480 lignes d'où la première valeur des temps dans le tableau ci-dessous.

4°) A partir du résultat de la question précédente, remplir le tableau ci-dessous :

temps(us)	15360	15360+350=15710	15774	16600
Compteur2	480	491	493	519
Symboles		UUU+1	VVV-1	YYY

5°) UUU=490 et VVV=494

Pour votre information, les valeurs typiques que j'utilise en projet sont :
XXX=799, ZZZ=658, TTT=756, et YYY=524, UUU=492, VVV=495



Corrections partielles TD5

Exercice 1

1°) quand rien ne nous permet de choisir n, on prend n=8. Ici, l'information est donnée plus haut, la mémoire programme est organisée en mots de 16 bits, il nous faut donc prendre n=16
Pour m : m=10 1ko, m=11 2ko, m=12 4 ko m= 13 8ko... c'est donc m=13.
2°) n est inchangé et on double la mémoire en ajoutant un bit => m=14.
3°) n= 16 et m=15 pour les mêmes raisons.

Exercice 2 (hors TD)

n=8 et m=9

Exercice 3

1°) **TWCR**

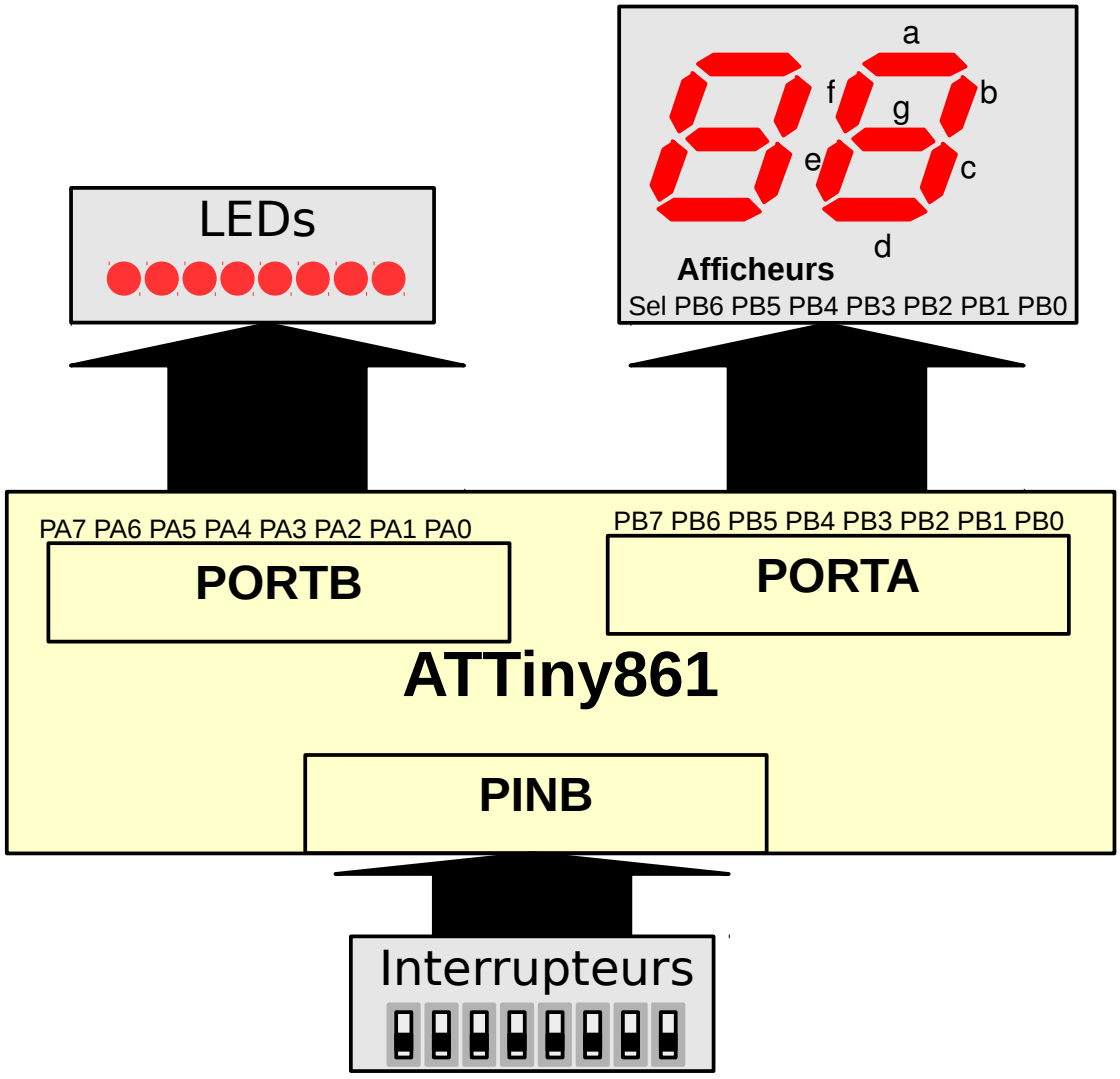
b_7	TWINT
b_6	TWEA
b_5	TWSTA
b_4	TWST0
b_3	TWWC
b_2	TWEN
b_1	----
b_0	TWIE

2°) TWCR |= 0x40;
TWCR &= ~(0x08); ou TWCR &= (0xF7);
(TWCR & 0x80) == 0x80
(TWCR & 0x04) != 0x04
ou
(TWCR & (1<<TWEN)) == 0x00
3°) TWCR |= (1<<TWEA); ou TWCR |= _BV(TWEA);
TWCR &= ~(1<<TWWC);
(TWCR & (1<<TWINT)) == (1<<TWINT)
(TWCR & (1<<TWEN)) != (1<<TWEN)
ou
(TWCR & (1<<TWEN)) == 0x00

Corrections partielles TD6

Exercice 2

1°)



Seule la partie PORTB et leds peut être déduite du programme donné !!!

2°)

```
1         #include "avr/io.h"
2         #undef F_CPU
3         #define F_CPU 15000000UL
4         #include "util/delay.h"
5         void main(void) {
6           uint8_t i=0; // ou unsigned char i=0;
7           while(1) {
8             i=i+3; // NOUVELLE incrémentation de i !!!!
9             PORTB =  i; // on sort vers l'extérieur
10            _delay_ms(1000);
11          }
12        }
```

3°) Le chenillard simple est réalisé par :

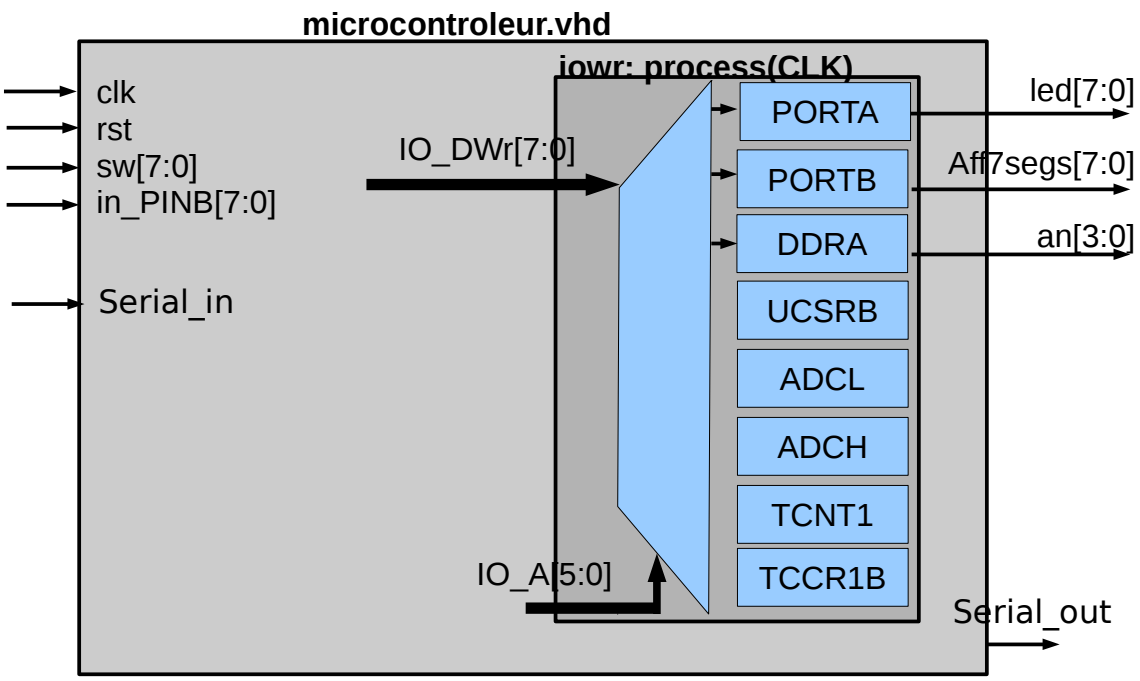
```
1         #include "avr/io.h"
2         #undef F_CPU
3         #define F_CPU 15000000UL
4         #include "util/delay.h"
5         void main(void) {
6           uint8_t i=1; // ou unsigned char i=0;
7           while(1) {
8             i = i << 1; // décalage vers la gauche de i
9             if (i==0) i=1;
10            PORTB =  i; // on sort vers l'extérieur
11            _delay_ms(1000);
12          }
13        }
```

Le chenillard double par

```
1         #include "avr/io.h"
2         #undef F_CPU
3         #define F_CPU 15000000UL
4         #include "util/delay.h"
5         void main(void) {
6           uint8_t i=1,j=128; // ou unsigned char i=0;
7           while(1) {
8             i = i << 1; // décalage vers la gauche de i
9             j = j >> 1; // décalage vers la droite de j
10            if (i==0) {
11               i=1;
12               j=128;
13            }
14            PORTB =  i | j; // on sort vers l'extérieur
15            _delay_ms(1000);
16          }
17        }
```

Exercice 3

1°)



2°) Le schéma ci-dessus répond aux questions 1 et 2. **PORTA**, **PORTB** et **DDRA** sont des PORTs et **UCSRB**, **ADCL**, **ADCH**, **TCNT1** et **TCCR1B** sont des registres et sont donc reliés à des signaux (fils internes)

3°) La question est ambiguë. Le code VHDL fait intervenir deux fils dans cet extrait de code :

```
1         s_soc <= IO_Dwr(7);
2         s_eoc <= IO_Dwr(6);
```

Pour garder les noms (sans le ''s_'') il faut donc :

```
3         #define SOC   7
4         #define EOC   6
```

4°)

```
5         #define DATAL _SFR_IO8(0x04)
6         #define DATAH _SFR_IO8(0x05)
```

5°) définition de ces constantes en tout début de l'architecture.

```
7         constant DATAL : std_logic_vector(5 downto 0) := "000100";
8         constant DATAH : std_logic_vector(5 downto 0) := "000101";
```

Exercice 4

1°) Un compteur compte et son affichage est réalisé sur deux digits avec une temporisation de 500 ms.

Pour 0x00, l'instruction sort

```
PORTB = ~digit7segs[(cmpt & 0x0F)];
```

digit7segs[0] = ~0xC0 = 0x3F allume tous les segments sauf g : affiche 0

Pour 0x01, l'instruction sort

```
PORTB = ~digit7segs[(cmpt & 0x0F)];
```

digit7segs[1] = ~0xF9 = 0x06 allume les segments b,c : affiche 1

Pour 0x02, l'instruction sort

```
PORTB = ~digit7segs[(cmpt & 0x0F)];
```

digit7segs[0] = ~0xA4 = 0x5B allume les segments a,b,d et g : affiche 2

Le tilde ''~'' est là car le tableau a été précalculé pour anode commune.

2°) Pour les mêmes raisons que la question précédente, 0, 1 et 2 sont affichés sur l'afficheur des dizaines car les calculs donnent 0, 1, et 2 comme indice du tableau.

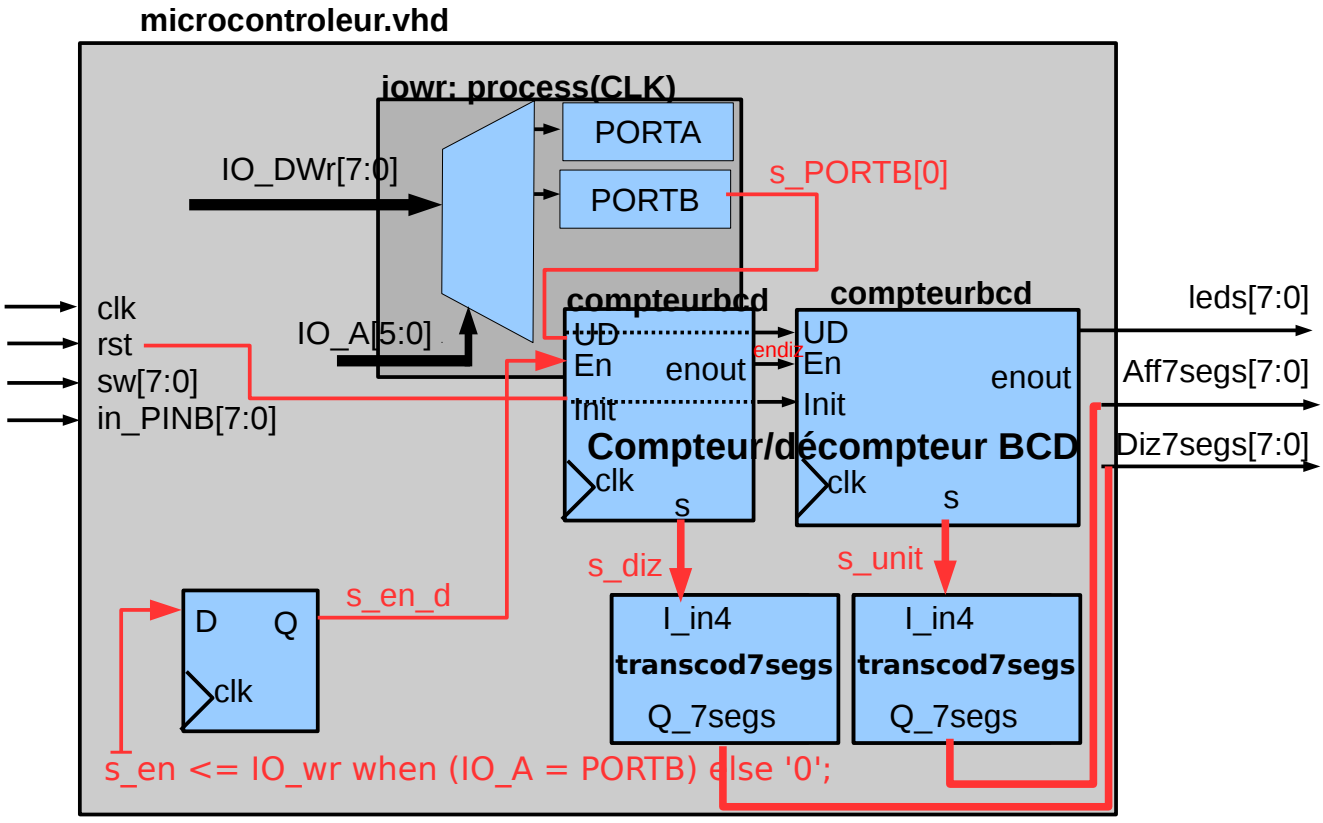
3°) 0x29+1 = 0x2A et cmpt&0x0F vaut 0x0A qui est plus grand que 9 donc on ajoute 0x06 : 0x2A+0x06 = 0x30

0x99+1 = 0x9A et cmpt&0x0F vaut 0x0A qui est plus grand que 9 donc on ajoute 0x06 : 0x9A+0x06 = 0xA0. Mais 0xA0&0xF0 donne 0xA0 plus grand que 0x90 donc on repasse à 0x00.

4°)

```
1         void incrementBCD(unsigned char *cnt) {
2           (*cnt)++;
3           if ((*cnt & 0x0F) > 0x09) *cnt += 6;
4           if ((*cnt & 0xF0) > 0x90) *cnt = 0;
5         }
6
7         void decrementBCD(unsigned char *cnt) {
8           (*cnt)--;
9           if ((*cnt & 0x0F) == 0x0F) *cnt -= 6;
10          if ((*cnt & 0xF0) == 0xF0) *cnt = 0x99;
11        }
```

Exercice 5



s_PORTB[0] est relié directement à l'entrée Down/Up UD du compteur. Elle doit être positionnée avant En, d'où la présence d'une bascule D permettant le retard.

Corrections partielles TD7

Exercice 1

1°) toto contiendra 0b01010101 soit 0x55

2°) Il sera affiché : ON, OFF, ON, OFF, ON, OFF, ON, OFF (soit 0xAA=~0x55)

Exercice 2

On peut reprendre le schéma de la correction du TD6. Donc les interrupteurs sont reliés à PINB.

```
#include "avr/io.h"
#undef F_CPU
#define F_CPU 15000000UL
#include "util/delay.h"
void main(void) {
  uint8_t i=1,j=128,inters; // ou unsigned char i=0;
  while(1) {
    i = i << 1; // décalage vers la gauche de i
    j = j >> 1; // décalage vers la droite de j
    if (i==0) {
       i=1;
       j=128;
    }
    inters = PINB;// lecture position interrupteurs
    if ((inters & 0x01)==0x01)// inter poids faible fait le choix
      PORTB =  i | j; // on sort vers l'extérieur
    else
      PORTB = i; // on sort vers l'extérieur
    _delay_ms(1000);
  }
}
```

Le fichier de contraintes permet de relier les entrées de PINA aux interrupteurs qui sont câblés sur la carte !

Exercice 3

1°) Un FIFO 16x8 est un ensemble de registres organisés de telle manière que la première valeur entrée soit la première lue (First In First out = FIFO). Il peut stocker jusqu'à 16 valeurs de 8 bits.

2)

```
do {
  etatFIFO = UCSRA; // PINC d'après le dessin, etatFIFO à déclarer
} while((etatFIFO & 0x80)!=0x80); //data_present en b7 voir dessin
```

3)

```
do {
  etatFIFO = UCSRA; // PINC d'après le dessin, etatFIFO à déclarer
} while((etatFIFO & 0x20)=0x20); //full en b5 mais complémenté voir dessin
```

4)

```
data = UDR;
```

5) Quelque chose comme :

```
1         digit = (data >> 4) & 0x0F;
2         char_digit_H = digit + 0x30; //0x30 code ASCII de '0'
3         if (char_digit_H > 0x39) char_digit_H += 7;//+7 pour atteindre le 'A'
4         digit = data & 0x0F;
5         char_digit_L = digit + 0x30;
6         if (char_digit_L>0x39) char_digit_L += 7;
```

fera l'affaire.

Exercice 4

1°)

```
7         -- IO write process
8         --
9         iowr: process(CLK)
10        begin
11            if (rising_edge(CLK)) then
12                if (IO_wr = '1') then
13                    case IO_A is
14           -- addresses for tiny861 device (use io.h).
15           --
16                        when PORTA  => -- PORTA=X"1B" (0X3B)
17                                       Led <= IO_Dwr;
18                        when  PORTB => -- PORTB=X"18" (0X38)
19                                    Aff7segs <= IO_Dwr;
20                        when TCNT1 => s_tcnt1 <= IO_Dwr;
21                        when TCCR1B => s_tccr1b <= IO_Dwr;
22                        when others =>
23                    end case;
24                end if;
25            end if;
26        end process;
27
28        -- IO read process
29        --
30        iord: process(IO_rd,IO_A,In_PINB,sw)
31        begin
32            -- addresses for tinyX6 device (use iom8.h).
33            --
34           if IO_rd = '1' then
35              case IO_A is
36                when  PINA => IO_Drd <= sw;  -- PINA=X"19" (0X39)
37                when  PINB => IO_Drd <= In_PINB;  -- PINB=X"16" (0X36)
38                when  TCNT1 => IO_Drd <= s_s_TCNT1;
39                when others => IO_Drd <= X"AA";
40              end case;
41           end if;
42        end process;
```

2°) La prédivision peut être faite par :

```
1         -- Timer 1
2           -- prescaler
3            process(clk) begin
4             if rising_edge(clk) then
```

```
5               s_prescaler <= s_prescaler + 1;
6             end if;
7           end process;
8           -- tick generation
9           s_div2 <= '1' when s_prescaler(0) = '1' else '0';
10          s_div4 <= '1' when s_prescaler(1 downto 0) = "11" else '0';
11          s_div8 <= '1' when s_prescaler(2 downto 0) = "111"  else '0';
12          s_div16 <= '1' when s_prescaler(3 downto 0) = "1111" else '0';
13          s_div32 <= '1' when s_prescaler(4 downto 0) = "11111" else '0';
14          s_div64 <= '1' when s_prescaler(5 downto 0) = "111111" else '0';
15          -- grand multiplexeur maintenant
16          s_en_timer1 <= '0' when s_tccr1b(2 downto 0) = "000" else -- arrêt timer
17                         '1' when s_tccr1b(2 downto 0) = "001" else -- division par 1
18                          s_div2 when s_tccr1b(2 downto 0) = "010" else
19                          s_div4 when s_tccr1b(2 downto 0) = "011" else
20                          s_div8 when s_tccr1b(2 downto 0) = "100" else
21                          s_div16 when s_tccr1b(2 downto 0) = "101" else
22                          s_div32 when s_tccr1b(2 downto 0) = "110" else
23                          s_div64 when s_tccr1b(2 downto 0) = "111";
```

3°) préscaler à 64.

On récupère le temps écoulé par simple lecture de **TCNT1** car on a pris soin de le mettre à 0 au tout début du calcul.

Résultat sur 16 bits car lecture de **ADCL** pui **ADCH** et fabrication d'un nombre sur 16 bits à partir de ces deux lectures.