Utilisation du cœur SLC1657

Dernière mise à jour : 10/08/10

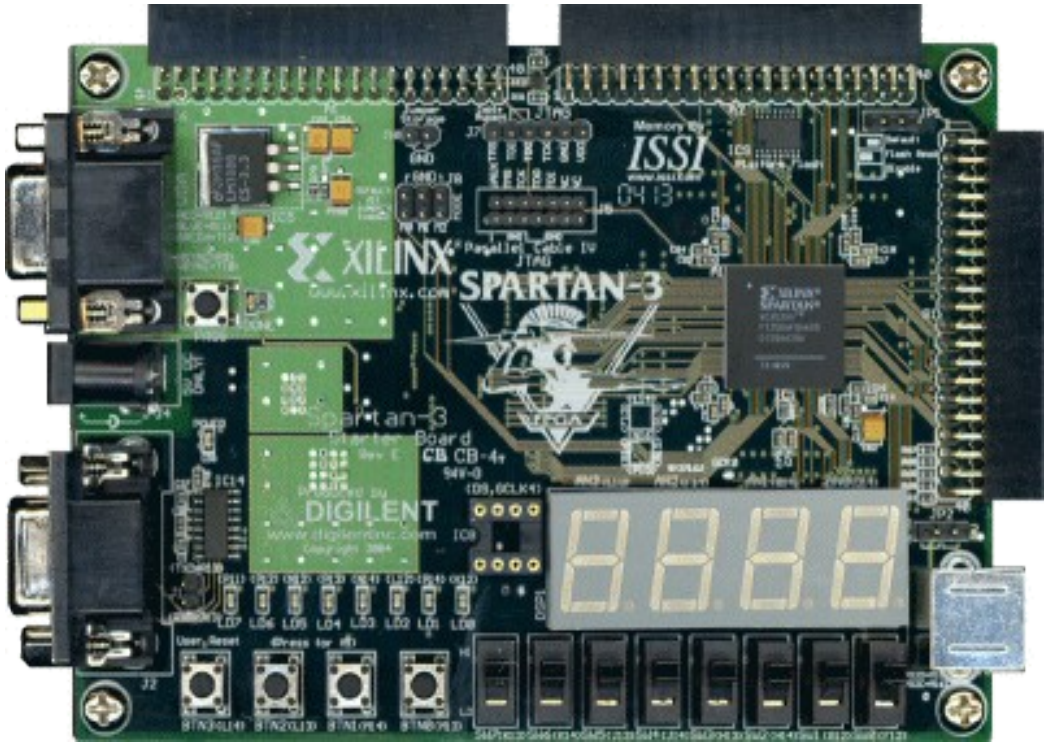
Rapport du tuteur du projet Serge Moutou 2009/2010 (serge.moutou@univ-reims.fr)

Mots clefs : processeur softcore, systèmes mono-puces, systèmes sur puces 8 bits, langage C et FPGA, FPGA Spartan3, Carte Digilent, Écran VGA

Introduction

Ce projet consiste à développer **un jeu de pong sur un écran VGA** dans un FPGA. C'est un sujet assez classique pour lequel plusieurs versions existent sur Internet. Ce qui fait son originalité est l'utilisation d'un cœur (libre) de micro-contrôleur 8 bits (SLC1657) en interaction avec de la logique externe. La programmation du cœur devra si possible, se faire en langage C.

La carte ciblée est une carte Digilent contenant un FPGA Xilinx (Spartan 3).



Les environnements de développement utilisés sont donc :

- ISE Xilinx pour développer en VHDL ou plus exactement le WebPack gratuit (http://www.xilinx.com/support/download/).

- MPLAB (http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en019469) pour développer les programmes pour le 16F57, avec soit l'assembleur par défaut, soit le compilateur C de chez Hi-Tech (http://www.htsoft.com/) qui est fourni par défaut avec MPLAB en version lite (ce qui nous convient parfaitement).

La terminologie française associée à ce type de projet est "**système sur puce**" ou "**système mono-puce**" tandis que la terminologie anglo-saxonne est SoC (System On Chip).

Ce projet a été donné à deux étudiants de deuxième année de DUT Génie Électrique pour une durée de 80 heures.

Le SLC1657 est un cœur de processeur (ou processeur softcore) compatible avec le PIC® 16C57 de microchip. On peut le trouver sur Internet : http://www.embeddedtronics.com/public/misc/slc1657.zip

Il a été développé par SiliCore (sa dernière version date de Septembre 2003) :

http://www.pldworld.com/_hdl/2/_ip/-silicore.net/index.htm

Nous allons commencer par présenter la partie hardware du cœur en nous basant à la fois sur la documentation de Silicore et celle de Microchip.

Architecture du PIC® 16F57

Il s'agit d'un processeur 8 bits avec des instructions codées sur 12 bits. C'est donc une architecture plus réduite encore que le célèbre 16F84 qui code ses instructions sur 14 bits.

Une autre de ses limitations est sa pile : avec une pile de deux niveaux seulement, il doit être difficile de réaliser un compilateur C.

Ce processeur n'a pas d'interruption.

Nous allons commencer par présenter le plus difficile, l'architecture des registres mémoire (Register File dans la terminologie anglo-saxonne que l'on pourrait traduire par dossier/classeur de registres, mais je préfère registres mémoire ou mémoire de registres). Je parle de difficultés parce qu'il y a plusieurs banques mémoires ce qui est une caractéristique des architectures 8 bits Microchip (jusqu'aux séries 18FXXX) qui disparait seulement avec les architectures 16/32 bits (24FXXX et autres)

Architecture des registres mémoire

La mémoire de registres est divisée en quatre banques. Nous avons d'abord la série des (vrais) registres jusqu'au PORTC. Cette partie de mémoire est suivie par 8 registres d'usage général pouvant être utilisés comme mémoire simple. Cet ensemble est identique quelque soit la banque choisie. Ensuite vient une série de registre généraux qui eux dépendent de la banque mémoire choisie.

Voici le schéma simplifié de cette architecture.

	Banque 0	Banque 1	Banque 2	Banque 3	
00h	Indirect addr.				60h
01h	TMR0				
02h	PCL				
03h	STATUS				
04h	FSR				
05h	PORTA				
06h	PORTB				
07h	PORTC				67h
08h 0Fh	8 Registres généraux	28h identiques 2Fh	48h sur les 4 4Fh	68h banques 6Fh	
10h 1Fh	16 registres généraux	30h 16 registres généraux 3Fh	16 registres généraux 5Fh	70h 16 registres généraux 7Fh	

Calculons la taille de la mémoire de registres généraux :

4 x 16 + 1 x 8 = 72 octets.

C'est tout ce qu'il y a de disponible comme mémoire dans cette architecture.

Les instructions

Il est temps de présenter maintenant l'ensemble des 32 instructions du PIC® 16F57.

Opérandes :

- f : register file address (adresse mémoire + registres) de 00 à 7F
- W : registre de travail
- d : sélection de destination : d=0 vers W, d=1 vers f
- bbb : adresse de bit dans un registre 8 bits (sur 3 bits)
- k : champ littéral (8, ou 9 bits)
- PC compteur programme

- TO Time Out bit
- PD Power Down bit

Opérations orientées octets entre registre et mémoire (File en anglais)					
Mnémonique Opérande	Description	Cycles	12 bits Opcode	status affected	notes
ADDWF f[,d]	Additionne W et f	1	0001 11df ffff	C,DC,Z	1,2,4
ANDWF f[,d]	ET bit à bit W et f	1	0001 01df ffff	N,Z	2,4
CLRF f	mise à 0 de f	1	0000 011f ffff	Z	2
CLRW	mise à 0 de f	1	0000 0100 0000	Z	2
COMF f,[d]	Complément de f	1	0010 01df ffff	Z	
DECF f[,d]	Décrémente f	1	0000 11df ffff	Z	2,4
DECFSZ f[,d]	Décrémente f (saute si 0)	1,(2)	0010 11df ffff		2,4
INCF f[,d]	Incrémente f	1	0010 10df ffff	Z	2,4
INCFSZ f[,d]	Incrémente f (saute si 0)	1,(2)	0011 11df ffff		2,4
IORWF f[,d]	Ou inclusif de f	1	0001 00df ffff	Z	2,4
MOVF f[,d]	déplacement de f (adressage direct)	1	0010 00df ffff	Z	2,4
MOVWF f	déplacement de W vers f	1	0010 000f ffff		1,4
NOP -	pas d'opération	1	0000 0000 0000		
RLF f[,d]	Rotation gauche avec la retenue	1	0011 01df ffff	C	2,4
RRF f[,d]	Rotation droite avec la retenue	1	0011 00df ffff	C	2,4
SUBWF f[,d]	soustrait W de f	1	0000 10df ffff	C,DC,Z	1,2,4
SWAPW f[,d]	inverser les quartets dans f	1	0011 10df ffff		2,4
XORWF f[,d]	Ou exclusif de f	1	0001 010df ffff	Z	2,4

Opérations orientées bits sur les registres					
Mnémonique Opérande	Description	Cycles	12 bits Opcode	status affected	notes
BCF f,b	mise à 0 du bit b dans f	1	0100 bbbf ffff		2,4

BSF f,b	mise à 1 du bit b dans f	1	0101 bbbf ffff		2,4
BTFSC f,b	test du bit b 0 de f saute si 0	1,(2)	0110 bbbf ffff		
BTFSS f,b	test du bit b 0 de f saute si 1	1,(2)	0111 bbbf ffff		

Opérations littérales (adressage immédiat) et de contrôles					
Mnémonique Opérande	Description	Cycles	12 bits Opcode	status affected	notes
ANDLW k	Et logique de W et k	1	1110 kkkk kkkk	Z	
CALL k	appel du sous programme n (sur 8 bits)	2	1001 kkkk kkkk		1
CLRWDT -	mise à 0 du timer watchdog	1	0000 0000 0100	/TO,/PD	
GOTO k	aller à l'adresse n (sur 8 bits)	2	101k kkkk kkkk		
IORLW k	OU inclusif logique de W et k		1101 kkkk kkkk	Z	
MOVLW k	chargement du littéral dans W	1	1100 kkkk kkkk		
OPTION k	charge le registre option	1	0000 0000 0010		
RETLW k	retour avec le littéral dans W	2	1000 kkkk kkkk		
SLEEP	aller au mode standby	1	0000 0000 0011	/TO,/PD	
TRIS f	Charge le registr TRIS	1	0000 0000 0fff		3
XORLW k	ou exclusif du littéral avec W	1	1111 kkkk kkkk	Z	

Note 1: Le 9eme bit du compteur programme sera forcé à 0 par toutes les intructions qui écrivent dans PC sauf pour l'instruction GOTO.

2: Quand un registre I/O est modifié en fonction de lui-même (e.g. MOVF PORTB, 1), la valeur utilisée est la valeur présente sur les broches elles-même. Par exemple, si la donnée mémorisée est '1' pour une broche configurée en entrée mais forcée par un '0' par un composant externe, la donnée sera écrite comme un '0'.Si cette instruction est exécutée sur le registre TMR0 (et, où, d = 1), le prescaler sera mis à 0 si assigné au module Timer0.

3: L'instruction TRIS f, où f = 5, 6 or 7 prend le contenu du registre W pour l'écrire dans les registres de direction de donnée et 3 états du PORTA, B ou C respectivement. Un '1' force la broche à un état haute impédance et désactive les buffers de sortie.

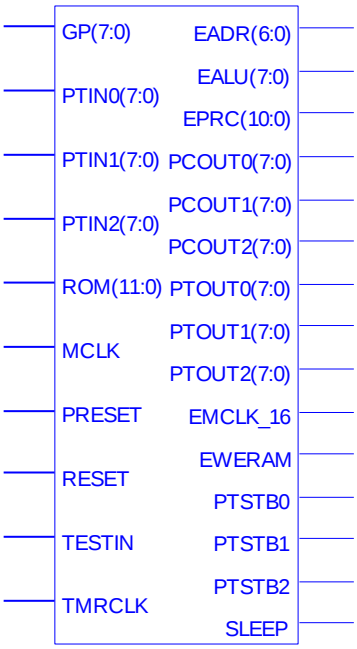
4: Si cette instruction est exécutée sur le registre TMR0 (et, où l'on applique, d = 1), le prescaler sera mis à 0 (si assigné à TMR0).

Assemblage complet du cœur

Il est grand temps de présenter le cœur et ses périphériques.

Le cœur proprement dit

Voici maintenant une description du cœur sous forme schématique et sous forme de programme VHDL.




```
entity TOPLOGIC is
    port (
          EADR:       out std_logic_vector(  6 downto 0 );
          EALU:       out std_logic_vector(  7 downto 0 );
          EMCLK_16:   out std_logic;
          EPRC:       out std_logic_vector( 10 downto 0 );
          EWERAM:     out std_logic;
          GP:          in  std_logic_vector(  7 downto 0 );
          MCLK:        in  std_logic;
          PCOUT0:     out std_logic_vector(  7 downto 0 );
          PCOUT1:     out std_logic_vector(  7 downto 0 );
          PCOUT2:     out std_logic_vector(  7 downto 0 );
          PTIN0:       in  std_logic_vector(  7 downto 0 );
          PTIN1:       in  std_logic_vector(  7 downto 0 );
          PTIN2:       in  std_logic_vector(  7 downto 0 );
          PTOUT0:     out std_logic_vector(  7 downto 0 );
          PTOUT1:     out std_logic_vector(  7 downto 0 );
          PTOUT2:     out std_logic_vector(  7 downto 0 );
          PTSTB0:     out std_logic;
          PTSTB1:     out std_logic;
          PTSTB2:     out std_logic;
          PRESET:      in  std_logic;
          RESET:       in  std_logic;
          ROM:         in  std_logic_vector( 11 downto 0 );
          SLEEP:      out std_logic;
          TESTIN:      in  std_logic;
          TMRCLK:      in  std_logic
        );
end entity TOPLOGIC;
```

Pour ce cœur, EPRC correspond au bus d'adresse de la ROM, tandis que ROM correspond au bus de données de cette même ROM. La RAM est gérée quant à elle avec EALU comme bus d'adresses et avec GP comme bus de données.

A noter que contrairement au PIC® 16F57, le composant SLC1657 utilise une entrée spécifique TMRCLK pour le timer0 (pour le PIC® c'est un bit b_4 T0CKl du PORTA).

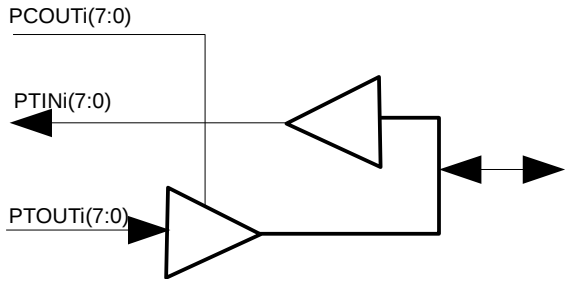
Avant de passer aux périphériques nécessaires au bon fonctionnement, il nous faut aborder le problème très spécifique des ports.

Les trois ports du SLC1657

Les ports sont des entités très spécifiques dans un micro-contrôleur car ils sont bidirectionnels. Ce côté bidirectionnel est difficile à implanter de manière générale dans un FPGA puisqu'il dépend du fabricant et donc de l'environnement d'utilisation. Alors, puisqu'elle dépend de l'environnement de programmation, cette description VHDL est laissée à l'utilisateur du cœur

SLC1657.

En principe, cette bidirectionnalité est gérée par un registre spécial TRIS associé à chacun des ports. Ces registres spécifiques existent dans le cœur, ils s'appellent **PCOUT0**, **PCOUT1** et **PCOUT2**. Ils sont tous sur 8 bits contrairement au PIC® 16F57 pour lequel le PORTA ne possède que 4 bits et peuvent être utilisés soit comme ports généraux en sortie, soit pour la gestion bidirectionnelle. Les ports proprement dit se décomposent alors en port d'entrée **PTIN0**, **PTIN1** et **PTIN2** et en ports de sortie **PTOUT0**, **PTOUT1** et **PTOUT2**.



Vous disposez donc de trois ports pour en faire un seul mais bidirectionnel si vous le désirez, comme indiqué dans la figure ci-contre. Ceci n'est pas une obligation et si vous le désirez vous pouvez conserver ces trois ports intacts : c'est ce que l'on fera dans notre projet.

Entrées/sorties bidirectionnelle optionnelle

Remarque : pour des raisons de compatibilité le port PCOUTi n'est pas accessible de manière normale. Si i=0 l'écriture dans PCOUT0 se fait par l'instruction :

```
// en langage C
TRISA=0xFF;
```

tandis qu'en assembleur on écrira :

```
; assembleur
movlw  255        ;1:input
TRIS    PORTA  ;PORTA en entree
```

même si les commentaires de ce programme ne sont pas appropriés dans le cas où l'on n'a pas un port bidirectionnel (ce qui, je le rappelle, est notre cas).

Abordons maintenant les deux autres périphériques nécessaires, la RAM et la ROM. Ces deux composants ne font pas partie du cœur tout simplement parce que les implantations de ceux-ci sont spécifiques aux FPGA cibles.

La RAM

Nous avons déjà eu l'occasion de présenter celle-ci, en tout cas du point de vue

de sa capacité. Sa spécificité est qu'il s'agit d'une mémoire à écriture synchrone mais à lecture asynchrone. Il est facile de trouver des exemples d'une telle mémoire en VHDL. Nous avons choisi :

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity xilinx_one_port_ram_sync is
  generic(
    ADDR_WIDTH: integer:=7;
    DATA_WIDTH: integer:=8
  );
  port(
    clk: in std_logic;
    we: in std_logic;
    addr: in std_logic_vector(ADDR_WIDTH-1 downto 0);
    din: in std_logic_vector(DATA_WIDTH-1 downto 0);
    dout: out std_logic_vector(DATA_WIDTH-1 downto 0)
   );
end xilinx_one_port_ram_sync;

architecture beh_arch of xilinx_one_port_ram_sync is
  type ram_type is array (2**ADDR_WIDTH-1 downto 0)
      of std_logic_vector (DATA_WIDTH-1 downto 0);
  signal ram: ram_type;
begin
  process (clk)
  begin
    if (clk'event and clk = '1') then
      if (we='1') then
        ram(to_integer(unsigned(addr))) <= din;
        end if;
    end if;
  end process;
           dout <= ram(to_integer(unsigned(addr)));
end beh_arch;
```

Remarque : une mauvaise implémentation de cette mémoire est fatale au bon fonctionnement du processeur, surtout s'il est utilisé avec un compilateur C qui utilise des variables locales

La ROM

C'est là que se situe le programme à exécuter. Puisque tout compilateur ou assembleur génère un fichier hex, il nous est nécessaire de transformer ce fichier hex en fichier VHDL. Ceci est laissé à un programme C décrit un peu plus loin. Commençons à présenter un exemple de ROM.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity PIC_ROM is
  port (
```

```
            Addr    : in   std_logic_vector(10 downto 0);
            Data    : out  std_logic_vector(11 downto 0));
end PIC_ROM;


architecture first of PIC_ROM is
begin
 Data <=
      "000000000000" When to_integer(unsigned(Addr)) = 0000 Else
      "110011111111" When to_integer(unsigned(Addr)) = 0000 Else
      "000000000101" When to_integer(unsigned(Addr)) = 0001 Else
      "110000000000" When to_integer(unsigned(Addr)) = 0002 Else
      "000000000110" When to_integer(unsigned(Addr)) = 0003 Else
      "001000000101" When to_integer(unsigned(Addr)) = 0004 Else
      "000000100110" When to_integer(unsigned(Addr)) = 0005 Else
      "101000000100" When to_integer(unsigned(Addr)) = 0006 Else
      "101000000000" When to_integer(unsigned(Addr)) = 2047 Else
      "000000000000";
end first;
```

Cet exemple montre un contenu mais chaque programme donnera une ROM différente. Il est à noter que ce que l'on présente est du VHDL alors que chaque compilateur ou assembleur ne délivre qu'un fichier au format HEX. Il faudra donc un utilitaire pour transformer le fichier HEX en fichier VHDL car il s'agit d'une opération qui peut se faire automatiquement. Nous utiliserons un programme en C que nous donnons en annexe 1.

Abordons maintenant les problèmes liés à la gestion d'un écran VGA.

Interfacer un écran VGA

La carte que nous utilisons permet une gestion d'écran VGA. Il s'agit de 5 signaux : trois signaux de couleur Rouge, vert et bleu et de deux signaux de synchronisation (horizontal et vertical).

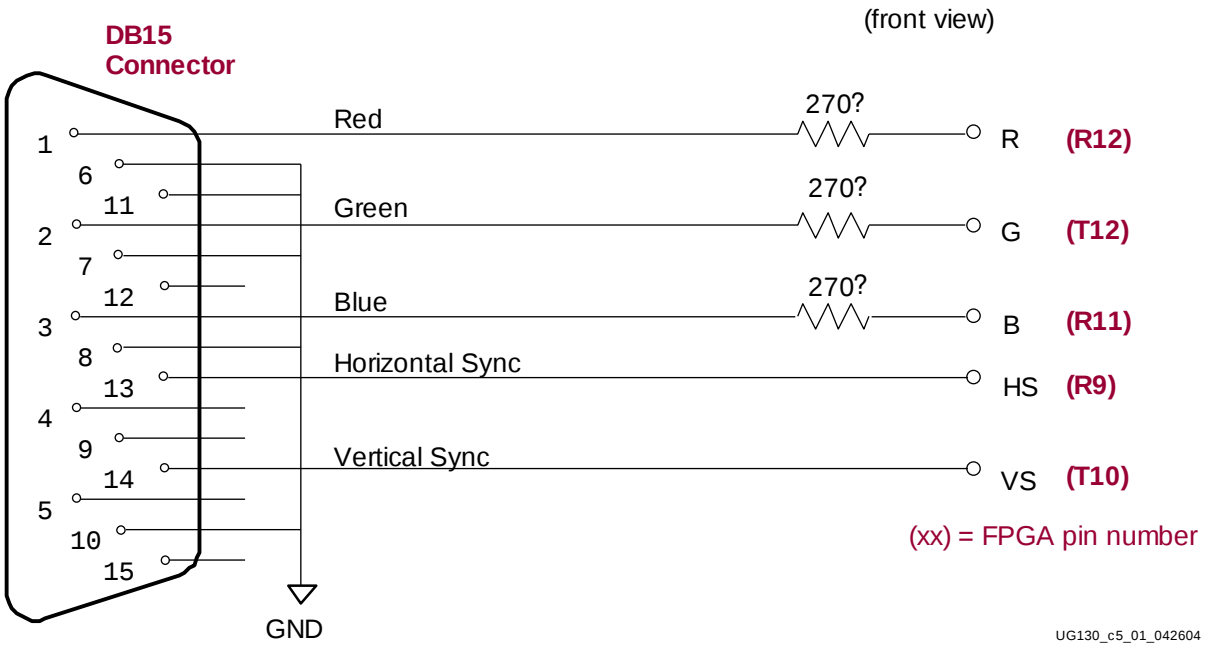
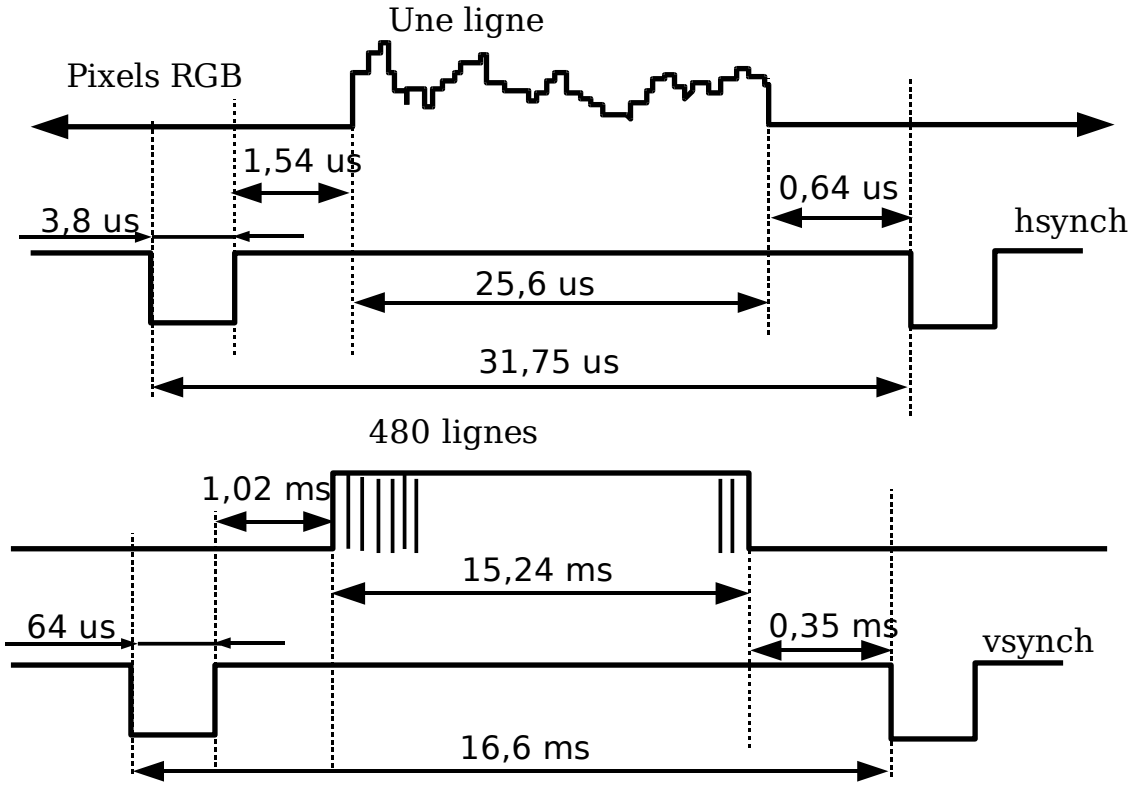


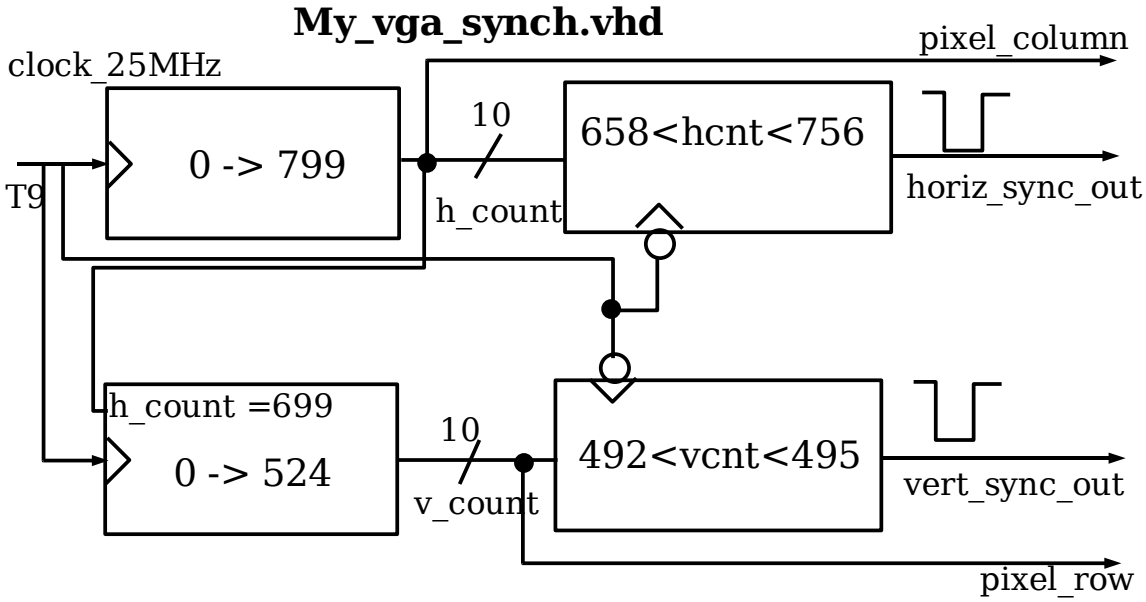
Figure 5-1: **VGA Connections from Spartan-3 Starter Kit Board**

Les signaux à construire

Les signaux de synchronisation à générer sont décrits dans la figure ci-après pour une résolution de 640 x 480. Ils sont simples et nécessitent seulement deux compteurs.



Nous pouvons donc réaliser les deux signaux hsynch et vsynch à l'aide de l'architecture ci-dessous où l'on représente partiellement le contenu du fichier My_vga_synch.vhd :

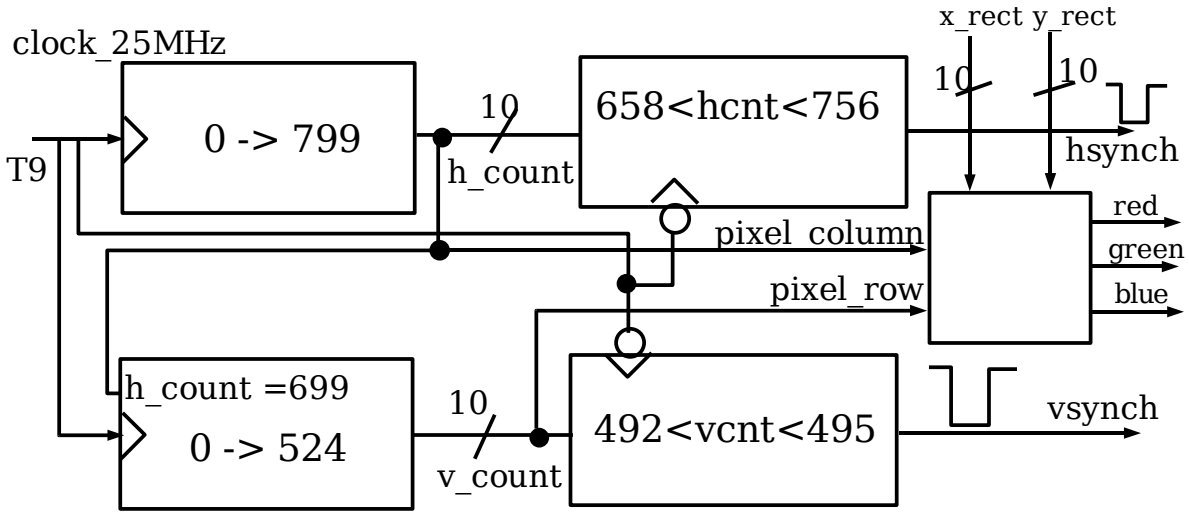


Les comparateurs sont des éléments combinatoires en principe. Mais comme on peut le voir ici, on les synchronise sur des fronts d'horloge descendants. Le programme VHDL capable de générer les signaux de synchronisation est donné

en annexe II. Le code VHDL correspondant permet simplement de synchroniser l'écran VGA mais ne dessine rien à l'intérieur. Voyons comment dessiner un rectangle.

Dessiner un rectangle

On cherche à dessiner un rectangle dont la position est donnée par un programme informatique (que l'on n'a pas encore présenté). Sa taille est fixée une fois pour toute, seule sa position peut changer. Regardez la figure ci-dessous et vous verrez une partie combinatoire ajoutée, dans laquelle on entre les positions du rectangle sur 10 bits (ici x_rect et y_rect). Tout dessin sur l'écran se trouvera dans ce composant. Les coordonnées x_rect et y_rect seront fournies par le processeur.



Avant d'aborder la programmation, il nous faut maintenant mettre ensemble tous ces composants pour un fonctionnement correct.

Assemblage du cœur, des mémoires et du module VGA

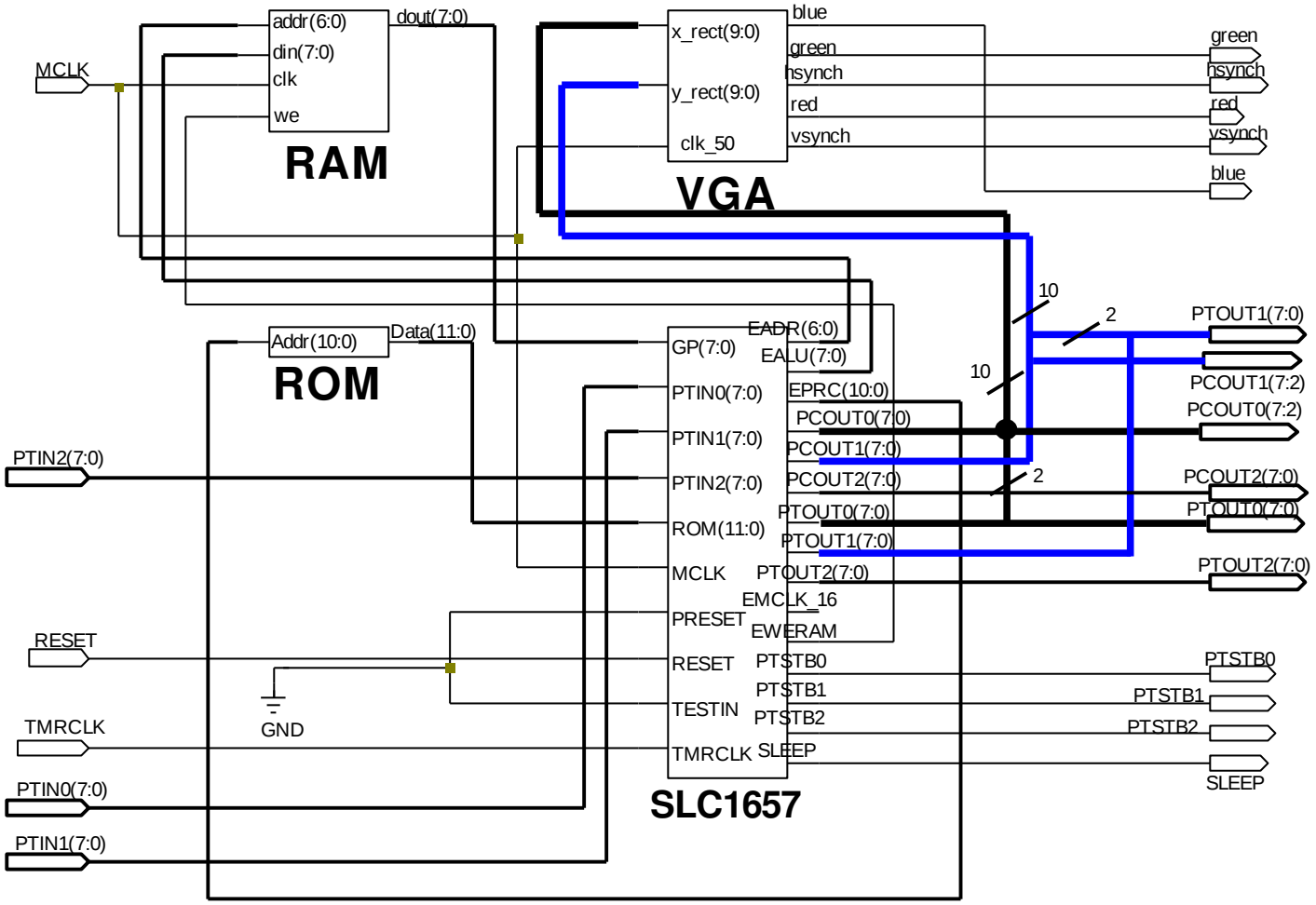
Nous donnons maintenant l'assemblage complet de notre projet. Ce schéma a été généré automatiquement par l'environnement de XILINX (ISE 9.2 ici).

Les choix technologiques ont été les suivants :

x_rect<9:8>	TRISA<1:0>
x_rect<7:0>	PORTA<7:0>
y_rect<9:8>	TRISB<1:0>
y_rect<7:0>	PORTB<7:0>

Un des inconvénients de ce projet est qu'il n'utilise pas d'autres ports que ceux

du cœur. Il reste même encore **PORTC** qui n'est absolument pas utilisé (mais sera utilisé plus tard).



Programmation du cœur

La programmation du cœur se fait à l'aide de l'environnement MPLAB. Ce n'est pas le seul environnement de développement pour les PICs, mais c'est probablement un des plus utilisés et pour ne rien gâcher, il est gratuit. Le téléchargement de MPLAB nous permet d'utiliser l'assembleur ainsi que le compilateur C HiTech. Comme nous avons décidé pour ce projet de n'utiliser que des outils gratuits, le compilateur C n'est pas optimisé, pour cela il faut payer.

Nous commençons par décrire rapidement l'assembleur.

Programmation en assembleur

Un des objectifs de ce projet a été d'éviter la programmation en assembleur

pour laisser place au langage C. Nous n'allons donc pas détailler l'assembleur du PIC16F5x en profondeur, mais seulement donner un exemple. A noter avant d'aborder notre exemple qu'une connaissance même partielle de l'assembleur est nécessaire si l'on veut déboguer un système sur puce.

Nous commençons par donner un programme qui ne fera pas grand chose sur notre partie matérielle (écran VGA) mais qui présente l'écriture dans le registre de direction avec son instruction spécifique TRIS. On rappelle en effet que notre écran VGA utilise les deux bits de poids faible de ce registre pour **PORTA** et **PORTB**.

```
                    list  p=16C57,r=DECIMAL
                    #include "P16C5x.INC"

          R10       equ     10

                    org     0x7FF       ; Reset Vector
                    goto    Start       ; Go back to the beginning

                    org     0x000       ; The main line code starts here
Start:              movlw   255         ;0:output
                    TRIS    PORTA  ;PORTA en entree
                    movlw   0                   ;1:input
                    TRIS    PORTB  ;PORTB en sortie
Loop:     movf      PORTA, W
                    movwf   PORTB
                    goto    Loop

end
```

Voici un autre exemple qui m'a servi a faire des tests.

```
                            list  p=16C57,r=DECIMAL

                    #include "P16C5x.INC"
                    org     0x7FF       ; Reset Vector
                    goto    Start       ; Go back to the beginning

                    org     0x000       ; The main line code starts here
Start:              movf    PORTA, W
                    andlw   03          ;0:output
                    TRIS    PORTA  ;PORTA en entree
                    movf    PORTA, W
                    movwf   PORTA
                    movlw   1                   ;1:input
                    TRIS    PORTB  ;PORTB en sortie
                    movf    PORTA, W
                    movwf   PORTB
                    goto    Start
end
```

Le **PORTA** est relié aux interrupteurs de la carte. Ses deux bits de poids faibles servent à la fois au poids faible et au poids fort de la coordonnée X. Aucun intérêt à part celui de voir si le matériel fonctionne correctement. Le poids fort

de la coordonnée Y est fixé à 1 (sur deux bits).

Programmation en C

La programmation en C est en général plus confortable, mais nous allons commencer par mettre de l'assembleur dans notre programme.

C avec assembleur

Commençons par un programme C qui ne contient que de l'assembleur.

```
#include <pic16F5x.h>
void main(){
#asm
Start:    movf    _PORTA, W
          ;movlw  01        ;0:output
          TRIS    _PORTA          ;PORTA en entree
          movwf   _PORTA
          movlw   1               ;1:input
          TRIS    _PORTB          ;PORTB en sortie
          movf    _PORTA, W
          movwf   _PORTB
          goto Start
#endasm
}
```

Ce programme est donné juste comme démonstration, il ne fait pas grand chose et surtout pas ce qu'il y a dans les commentaires.

Si vous voulez utiliser PORTA au lieu de _PORTA il vous faut rajouter une directive d'inclusion juste après la directive #asm comme montré ci-dessous :

```
#asm
        #include <aspic.h>
```

Pour positionner la balle à un endroit précis nous avons construit deux sous-programmes setX et setY. Les deux étant très similaires nous ne nous intéresserons qu'à setX.

Le sous-programme "setX"

D'abord une version en C pur

```
void setX(unsigned int x){
  PORTA=x;
  TRISA=x>>8;
}
```

Le contenu de ce programme est complètement déterminé par la partie

matérielle. Présentons la même chose avec un contenu en assembleur pour montrer comment on gère les paramètres dans ce cas :

```
void setX(unsigned int x){
#asm
;           bcf       fsr, 5     ;FSR5=0, select bank0
;           bcf       fsr, 6     ;FSR6=0, select bank0
            movf ?_setX,w
            movlw _PORTA
 ;TRISA=x>>8;
            movf ?_setX+1,w
            TRIS _PORTA
#endasm
}
```

Il y aurait certainement beaucoup à explorer autour des banques (en RAM) avec ce genre de programme mais les étudiants y ont renoncé d'où la présence des commentaires pour la gestion des banques.

Déplacement horizontal en continu de notre balle/rectangle

Nous allons présenter maintenant un programme fonctionnel qui déplace sans arrêt le rectangle sur une horizontale.

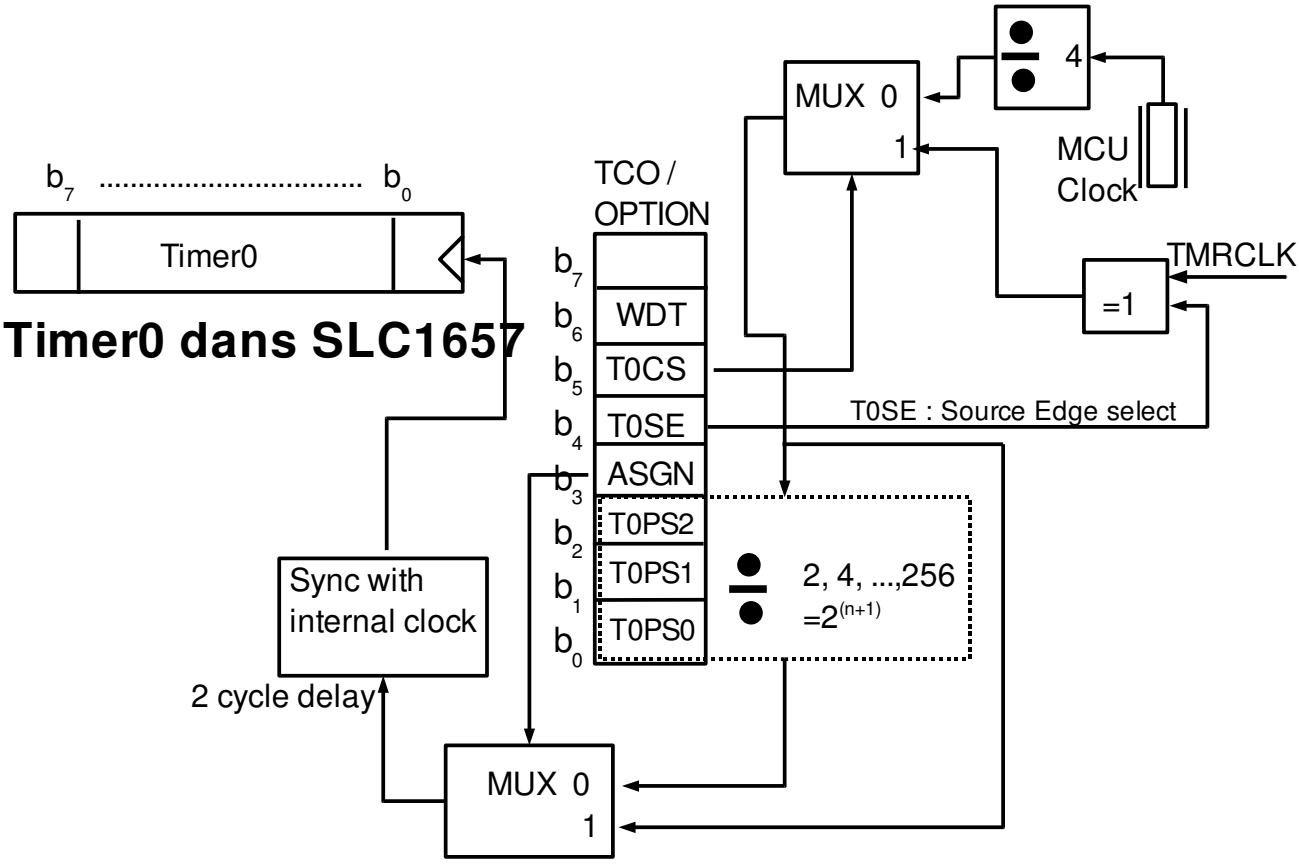
```
#include <pic16F5x.h>

void setX(unsigned int x);
void setY(unsigned int y);
void wait(int tempo);

void main(){
int i;
            while(1){
                        setY(321);
                        for(i=0;i<600;i++){
                                    setX(i);
                                    wait(30000);
                                    wait(30000);
                        }
            }
}

void setX(unsigned int x){
 PORTA=x;
 TRISA=x>>8;
}

void setY(unsigned int y){
 PORTB=y;
 TRISB=y>>8;
}

void wait(int tempo){
```

```
 int i;
 for(i=tempo;i>0;i--);
}
```

Pas mal, mais peut-on faire mieux avec une attente qui utilise le timer0 ?

Utiliser le Timer0 avec le langage C

Le programme précédant nécessite deux appels au sous-programme wait. Nous aimerions bien explorer maintenant la possibilité de faire la même chose mais en utilisant le seul et unique timer de notre architecture, à savoir le timer0. Commençons par présenter la documentation du Timer0 sous la forme d'un schéma :



ASGN=1 : le prescaler est assigné au watchdog (division 1 -> 128)
ASGN=0 : le prescaler est assigné au timer (division 2 -> 256)

Voici donc un exemple d'utilisation :

```
void wait(unsigned char tempo){
 OPTION=0x07; // div par 256 et source=quartz
 TMR0 =0;
 while(TMR0<tempo);
```

```
}
```

Ce nouveau sous-programme "wait" appelé avec une valeur de 250 est un peu plus rapide qu'un appel de

```
void wait(int tempo){
  int i;
  for(i=tempo;i>0;i--);
}
```

avec une valeur de 30000.

Programme complet de gestion simple de la balle

Nous présentons maintenant un programme simple de gestion de la balle avec des rebonds :

```
#include <pic16F5x.h>
void setX(unsigned int x);
void setY(unsigned int y);
void wait(unsigned char tempo);
void main(){
int posX=0,posY=0;
signed char deltaX=1,deltaY=1;
        while(1){
                if ((posX>=620) && (deltaX>0)) deltaX= -deltaX;
                if ((posX<=40) && (deltaX<0)) deltaX= -deltaX;
                posX=posX+deltaX;
                setX(posX);
                if ((posY>=460) && (deltaY>0)) deltaY= -deltaY;
                if ((posY<=10) && (deltaY<0)) deltaY= -deltaY;
                posY=posY+deltaY;
                setY(posY);
                wait(250);
                wait(250);
        }
}

void setX(unsigned int x){
  PORTA=x;
  TRISA=x>>8;
}

void setY(unsigned int y){
  PORTB=y;
  TRISB=y>>8;
}

void wait(unsigned char tempo){
OPTION=0x07; // div 256 et source=quartz
TMR0 =0;
while(TMR0<tempo);
}
```

L'inconvénient de ce programme est le petit nombre des trajectoires gérées, ce qui peut devenir lassant pour un jeu.

Ajouter les bords, et les raquettes

Commençons par des petits calculs. Nous voulons deux raquettes soit deux coordonnées verticales sur 9 bits (soit 4 ports 8 bits) et deux coordonnées de balles soit encore 4 ports de 8 bits. Cela fait en tout 8 ports. Rappelons que l'architecture du SiliCore1657 ne propose que 3 ports (PORTA, PORTB et PORTC) que l'on peut doubler avec TRISA, TRISB et TRISC mais que cela n'en fait que 6 sur les 8 nécessaires. Pour résoudre ce problème, les étudiants ont choisi de gérer les coordonnées des raquettes sur 8 bits.

Solution simple sans bord

Si l'on numérote les huit ports de sotie de 0 à 7 on choisit

position X de la	x_rect<9:8>	TRISA<1:0>
balle	x_rect<7:0>	PORTA<7:0>
position Y de la	y_rect<9:8>	TRISB<1:0>
balle	y_rect<7:0>	PORTB<7:0>
	y_raqG<7:0>	PORTC<1:0>
	y_raqD<7:0>	TRISC<7:0>

On constate donc que les raquettes droites et gauches ont une coordonnée y sur 8 bits. Pour pouvoir gérer des rectangles de tailles différentes et de couleur différentes, on complique un peu la partie destinée à dessiner un rectangle en lui donnant une couleur de rectangle une largeur et une hauteur. Voici donc notre nouveau composant :

```
COMPONENT rect IS PORT(
  row,col,x_rec,y_rec,delta_x,delta_y :in STD_LOGIC_VECTOR(9 DOWNTO 0);
  colorRGB : in  STD_LOGIC_VECTOR(2 DOWNTO 0);
  red1,green1,blue1 : out std_logic);
END component;
```

L'intanciation des rectangles pour la balle et les raquettes se fera alors de la manière suivante :

```
balle:rect port map(row=>srow, col=>scol,r ed1=>sred, green1=>sgreen, blue1=>sblue,
colorRGB=>"111", delta_x=>"0000001010", delta_y=>"0000001100",
            x_rec => x_rect, y_rec => y_rect);
raquetteG:rect port map(row=>srow, col=>scol, red1=>sred1, green1=>sgreen1,
        blue1=>sblue1, colorRGB=>"100", delta_x=>"0000001010",
        delta_y=>"0000111010",       x_rec => "0000010110",
```

```
        y_rec(8 downto 1) => y_raquG, y_rec(9)=>'0',y_rec(0)=>'0');
raquetteD:rect port map(row=>srow, col=>scol, red1=>sred2, green1=>sgreen2,
        blue1=>sblue2,colorRGB=>"100", delta_x=>"0000001010",
        delta_y=>"0000111010", x_rec => "1001001000",
        y_rec(8 downto 1) => y_raquD,y_rec(9)=>'0',y_rec(0)=>'0');

  red <= sred or sred1 or sred2;
  green <= sgreen or sgreen1 or sgreen2;
  blue <= sblue or sblue1 or sblue2;
```

Les déclarations des signaux sont omises. Voici maintenant le programme C permettant le rebond sur les raquettes.

```
#include <pic16F5x.h> // Programme pour Hitech C dans MPLAB
void setX(unsigned int x);
void setY(unsigned int y);
void wait(unsigned char tempo);
unsigned int posRaqu_16;
void main(){
int posX,posY;
unsigned char raqD_y=0,raqG_y=0;
signed char deltaX=1,deltaY=1;
          while(1){
                    posX=113;
                    posY=101;
                    setX(posX);
                    setY(posY);
                    while(RC2==0); // attente départ
                    while( (posX>30) && (posX<580)){
                              posRaqu_16=raqD_y<<1;
                              if ((posX>=574) && (posY<posRaqu_16+58) &&
                                   (posY>posRaqu_16-10) && (deltaX>0)) deltaX= -deltaX;
                              posRaqu_16=raqG_y<<1;
                              if ((posX<=32) && (posY<posRaqu_16+58) &&
                                  (posY>posRaqu_16-10) && (deltaX<0)) deltaX= -deltaX;
                              posX=posX+deltaX;
                              setX(posX);
                              if ((posY>=460) && (deltaY>0)) deltaY= -deltaY;
                              if ((posY<=10) && (deltaY<0)) deltaY= -deltaY;
                              posY=posY+deltaY;
                              setY(posY);
// gestion des raquettes 2bits PORTC/raquette
                              if (RC0) if (raqG_y<215) raqG_y++;
                              if (RC1) if (raqG_y>0) raqG_y--;
                              PORTC=raqG_y;
                              if (RC6) if (raqD_y<215) raqD_y++;
                              if (RC7) if (raqD_y>0) raqD_y--;
                              TRISC=raqD_y;
                              wait(250);
                              wait(250);
                    }
          }
}
```

Comme on peut le voir nous avons choisit le **PORTC** connecté aux

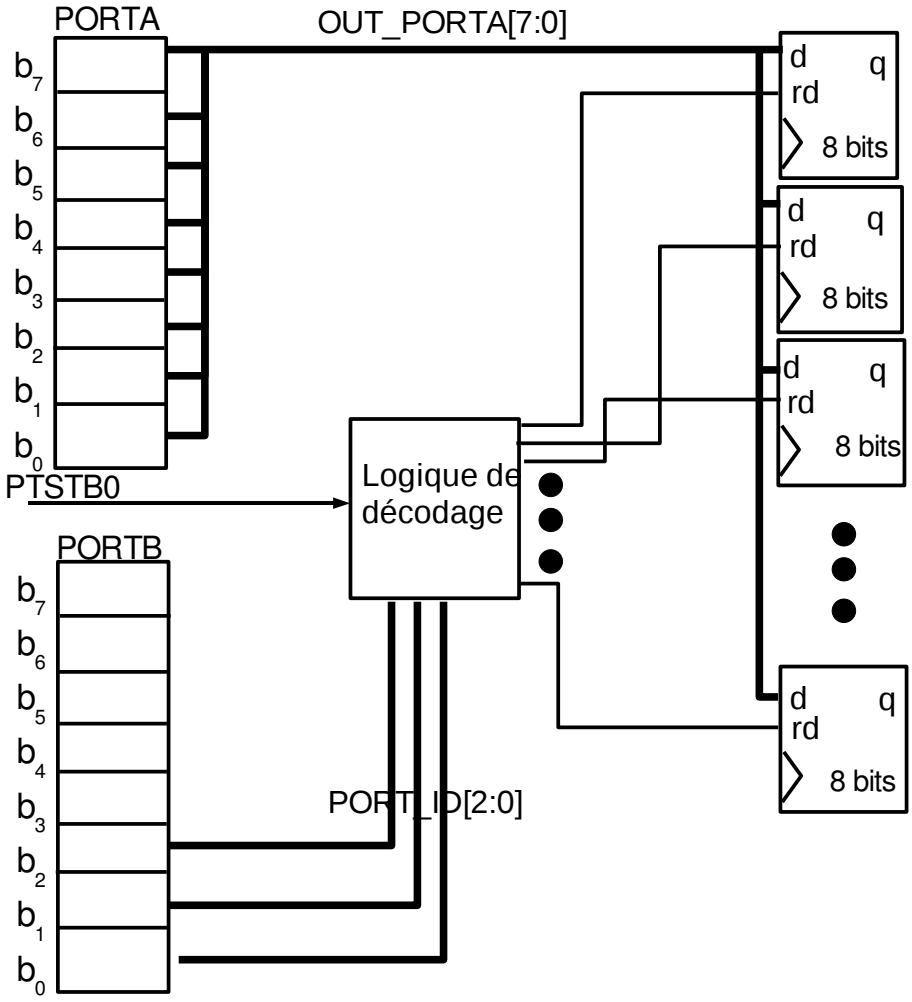
interrupteurs sw0,sw1,sw6 et sw7 de la carte, pour faire descendre et monter les raquettes et sw2 (RC2) pour le départ.

Remarquez aussi la présence de la variable globale `unsigned int posRaqu_16;` (qui ne peut pas être mise en locale) qui est là pour gérer les 9 bits de la position réelle des raquettes alors que notre variable C n'en contient que 8 bits.

Je vais présenter maintenant une autre technique qui consiste à augmenter le nombre de PORTS du cœur. Le cœur a été pourvu de signaux "strobe" qui sont des grands classiques dans les architectures à microprocesseurs : signal indiquant que l'écriture dans un port vient d'être réalisé. Il s'agit ici des signaux PTSTB0, PTSTB1, et PTSTB2.

Créer huit ports avec deux (partie non testée)

Nous allons utiliser **PORTA** et **PORTB** de la façon suivante : les 3 bits de poids faible de PORTB vont nous servir à sélectionner un PORT parmi les 8 disponibles et le signal strobe PTSTB0 sera alors acheminé à ce PORT, ainsi une écriture dans PORTA sera dirigée vers le PORT concerné. Un schéma sera plus parlant :



où l'on a dessiné quatre des huit ports de sortie. Il va nous falloir modifier ensuite les choix technologiques et les sous-programmes de positionnement de la balle et des raquettes.

Choix technologiques(partie non testée)

Si l'on numérote les huit ports de sotie de 0 à 7 on choisit

x_rect<9:8>	PORT1<1:0>
x_rect<7:0>	PORT0<7:0>
y_rect<9:8>	PORT3<1:0>
y_rect<7:0>	PORT2<7:0>
y_raq1<9:8>	PORT5<1:0>
y_raq1<7:0>	PORT4<7:0>
y_raq2<9:8>	PORT7<1:0>
y_raq2<7:0>	PORT6<7:0>

Les sous-programmes en C correspondants (partie non testée)

Les sous-programmes pour positionner les raquettes et la balle sont présentés maintenant.

Le sous-programme "setX" (partie non testée)

Voici une version en C pur

```
void setX(unsigned int x){
  PORTB=1; //poids faible
  PORTA=x;
  PORTB=2;//poids fort
  PORTA=x>>8;
}
```

Pour ne pas alourdir ce document, nous ne donnons pas les autres sous-programmes pour lesquels seules les valeurs à donner à PORTB sont à changer.

Perspectives d'avenir

Explorer s'il n'y a pas moyen d'éviter la compilation de tout le projet avec une

gestion de librairie dans l'environnement Xilinx. Sinon, réaliser une ROM que l'on peut modifier par téléchargement soit série, soit parallèle. Il est en effet très lourd de compiler tout le projet chaque fois que l'on veut essayer un programme.

Le projet initial contient une version de ROM qu'il faudrait mettre en œuvre et détailler (voir le fichier SourceCode/xilinx/VHDL_source/Rev2.0/semrmint.vhd). Ceci représente un travail important d'autant plus que la connexion port parallèle JTAG ne semble pas être documenté dans le projet original. On dispose seulement d'un fichier exécutable sous DOS pour la communication (dans SourceCode/Download/DOWNLOAD.C).

Réaliser un affichage des scores en mode texte dans la partie basse de l'écran.

Conclusion

Les étudiants ont eu un certain nombre de difficultés à mener à bien ce projet. En effet, la balle qui se déplace sur l'écran a été obtenue à mi-projet, c'est à dire au bout de 40 heures. Les raisons ont été multiples :

- le tuteur n'avait pas débogué le projet. Un problème avec la RAM a nécessité 20 heures de travail au tuteur avant de positionner le problème justement dans cette RAM. Combien d'heures auraient été nécessaires à des étudiants de L2 ? Certainement trop pour tenir en 80 heures.

- ce projet nécessite de bonnes connaissances de la compilation et des processeurs. Aujourd'hui, malheureusement, nos étudiants n'ont aucune connaissance approfondie dans ces domaines. Le matériel est peut être le plus connu, mais la compilation est très mal appréhendée : nos étudiants connaissent le langage Java mais n'ont pas beaucoup d'idées sur les passages de paramètres et autres variables locales ou globales.

Intéressant aussi de mentionner que j'ai choisi ce projet uniquement à l'épaisseur de la documentation (194 pages) du cœur, ce qui m'a été très utile. Il existe beaucoup de SoC disponibles mais peu sont aussi bien documentés.

Si ce projet est attractif pour des étudiants, il ne donne pas une vue assez globale de ce que l'on peut mettre comme logique autour d'un cœur. En effet, il n'utilise pas d'autres ports que ceux du cœur. Si vous vous trouvez dans la situation contraire (besoin de nombreux PORTS), il vous faudra être imaginatif, mais sachez que le cœur a été pensé pour gérer ce genre de circonstances (ceci a été finalement documenté dans ce rapport, mais n'a pas été abordé par les étudiants et pas testé par le tuteur).

On a été très vite confronté au problème de la petite taille mémoire RAM du cœur. L'exemple typique a été les comparaisons pour déterminer si la balle heurtait la/les raquettes. Dès que cette comparaison était faite en C avec le sous-programme wait sans timer, on n'avait pas assez de mémoire RAM. Ce calcul a failli être déporté dans la partie matérielle décrite en VHDL avant que l'on s'aperçoive que l'utilisation d'un timer résolvait le problème.

Je n'ai personnellement pas eu le temps de comprendre exactement la façon

dont étaient gérées les variables avec le compilateur HitechC que l'on a utilisé. Des économies de place RAM dans le code généré n'ont ainsi pas pu être obtenues.

Lors de la réalisation de ce projet, j'ai été obsédé par la possibilité de faire "tourner" du C dans un FPGA. C'est la première fois que j'en avais l'occasion en effet. Il me semble après coup, que les limitations RAM de cette architecture, conduisent plutôt à l'utiliser en assembleur qu'en langage C.

Comparaison SiliCore1657 et Picoblaze®

	SiliCore1657	PicoBlaze ®
Statut	Libre	IP Core Xilinx
Code source	Disponible (voir en début de ce rapport)	Disponible au téléchargement chez Xilinx mais illisible
Taille dans FPGA	30% d'un spartan 3	4% d'un spartan 3
RAM	72 octets	64 octets
Registres	7 spécialisés	16 d'utilisation générale
PORTS	3 mais peut être étendu	peut être étendu jusqu'à 256
ROM	2048 x 12 bits	1024 x 18 bits
Pile	2 niveaux	31 niveaux
Environnement de développement	MPLAB gratuit et connu qui trourne aussi sous Linux	PBlazeIDE (mediatronix) et KPicosim sous Linux
Langages	C, assembleur	assembleur
Interruptions	Aucune	dispo mais à gérer soi-même Existe un gestionnaire chez opencores.org
Documentation	Très bien documenté	Très bien documenté

Ces deux architectures ont à peu près les mêmes champs d'application : petits programmes et grosses parties matérielles autour. J'ai une petite préférence pour le Silicore1657 parce qu'il existe un compilateur C qui le cible.

J'ai l'intention d'explorer l'architecture 16F84 l'année universitaire prochaine (2010/2011). Le début du projet correspondant est déjà en ligne sur mon site personnel : "http://perso.wanadoo.fr/moutou/ER2/Core16F84.pdf". L'autre architecture explorée sera un ATMEL disponible à

"http://opencores.org/project,cpu_lecture" pour lequel la rédaction du projet est, elle aussi, déjà en ligne :

"http://perso.wanadoo.fr/moutou/ER2/CoreAtMega8.pdf"

Remerciements

Ce projet a été soutenu financièrement par Xilinx à travers son programme universitaire (XUP) qui nous a permis de disposer de cinq cartes Digilent S3 gratuitement.

Rapport réalisé avec OpenOffice 2.4.1

ANNEXE I (transformer un fichier HEX en VHDL)

```
//from Thomas A. Coonan (tcoonan@mindspring.com) and adapted for Xilinx by S. Moutou
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

FILE *fpi, *fpo;

#define MAX_MEMORY_SIZE  1024
struct {
        unsigned int  nAddress;
        unsigned int  byData;
} Memory[MAX_MEMORY_SIZE];

char szLine[80];
unsigned int  start_address, address, ndata_bytes, ndata_words;
unsigned int  data;
unsigned int  nMemoryCount;
char *MakeBinaryString (unsigned int data);

char *szHelpLine =
"\nThe Synthetic PIC --- HEX File to VHDL Memory Entity conversion."
"\nUsage: HEX2VHDL <filename>"
"\n  Input file is assumed to have the extension 'hex'."
"\n  Output will go to the filename with the extension 'vhd'.\n\n";

char szInFilename[40];
char szOutFilename[40];

int main (int argc, char *argv[])
{
        int  i;
        if (!(argc == 2 || argc == 3)) {
                printf (szHelpLine);
                exit(1);
        }
        if ( (strcmp(argv[1], "?") == 0) ||
                 (strcmp(argv[1], "help") == 0) ||
                 (strcmp(argv[1], "-h") == 0) ||
                 (strcmp(argv[1], "-?") == 0)) {
                printf (szHelpLine);
                exit(1);
        }
        strcpy (szInFilename, argv[1]);
        if ( strstr(szInFilename, ".") == 0)
                strcat (szInFilename, ".hex");

        strcpy (szOutFilename, argv[1]);
        strcat (szOutFilename, ".vhd");
```

```
	if((fpi=fopen(szInFilename, "r"))==NULL){
		printf("Can\'t open file %s.\n", szInFilename);
		exit(1);
	}
	nMemoryCount = 0;
	while (!feof(fpi)) {
		fgets (szLine, 80, fpi);
		if (strlen(szLine) >= 10) {
			sscanf (&szLine[1], "%2x%4x", &ndata_bytes, &start_address);
			if (start_address >= 0 && start_address <= 20000 && ndata_bytes > 0) {
				i = 9;
				ndata_words   = ndata_bytes/2;
				start_address = start_address/2;
				for (address = start_address; address < start_address +
ndata_words; address++) {
					sscanf (&szLine[i], "%04x", &data);
					data = ((data >> 8) & 0x00ff) | ((data << 8) & 0xff00);
					i += 4;
					Memory[nMemoryCount].nAddress = address;
					Memory[nMemoryCount].byData   = data;
					nMemoryCount++;
				}
			}
		}
	}
	if((fpo=fopen(szOutFilename, "w"))==NULL){
		printf("Can't open VHDL file '%s'.\n", szOutFilename);
		exit(1);
	}
	fprintf (fpo, "\nlibrary IEEE;");
	fprintf (fpo, "\nuse IEEE.std_logic_1164.all;");
	fprintf (fpo, "\n--use IEEE.std_logic_arith.all;");
	fprintf (fpo, "\nuse IEEE.numeric_std.all;");
	fprintf (fpo, "\n\nentity PIC_ROM is");
	fprintf (fpo, "\n  port (");
	fprintf (fpo, "\n    Addr    : in   std_logic_vector(10 downto 0);");
	fprintf (fpo, "\n    Data    : out  std_logic_vector(11 downto 0));");
	fprintf (fpo, "\nend PIC_ROM;");
	fprintf (fpo, "\n\n\narchitecture first of PIC_ROM is");
	fprintf (fpo, "\nbegin");
	fprintf (fpo, "\n  Data <= ");
	for (i = 0; i < nMemoryCount; i++) {
		fprintf (fpo,"\n       \"%s\" When to_integer(unsigned(Addr)) = %04d Else",
			MakeBinaryString(Memory[i].byData)+4,
			Memory[i].nAddress
		);
	}
	fprintf (fpo, "\n       \"000000000000\";");
	fprintf (fpo, "\nend first;");
	fclose (fpi);
	fclose (fpo);
}

char *MakeBinaryString (unsigned int data)
{
	static char szBinary[20];
	int  i;
```

```
        for (i = 0; i < 16; i++) {
                if (data & 0x8000) {
                        szBinary[i] = '1';
                }
                else {
                        szBinary[i] = '0';
                }
                data <<= 1;
        }
        szBinary[i] = '\0';
        return szBinary;
}
```

Attention : Le fichier VHDL généré par ce programme comporte parfois une erreur sur le premier when : deux fois un `When to_integer(unsigned(Addr)) = 0000` . C'est d'ailleurs le cas du seul exemple donné dans ce document. Je n'ai pas vérifié la gravité de ce problème mais il me semble que je n'ai jamais rencontré de problème de ROM dans ce projet... ce qui me laisse penser que ce n'est pas très grave... mais il serait plus sage de vérifier tout cela.

ANNEXE II (gestion du VGA)

```
-- ************* My_vga_synch.vhd ************
library IEEE;
use  IEEE.STD_LOGIC_1164.all;
use  IEEE.STD_LOGIC_ARITH.all;
use  IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY VGA_SYNC IS
        PORT(   clock_25Mhz      : IN       STD_LOGIC;
        horiz_sync_out, vert_sync_out       : OUT    STD_LOGIC;
        pixel_row, pixel_column: OUT STD_LOGIC_VECTOR(9 DOWNTO 0));
END VGA_SYNC;
ARCHITECTURE a OF VGA_SYNC IS
        SIGNAL horiz_sync, vert_sync : STD_LOGIC;
        SIGNAL h_count, v_count :STD_LOGIC_VECTOR(9 DOWNTO 0);
BEGIN
--Generate Horizontal and Vertical Timing Signals for Video Signal
-- H_count counts pixels (640 + extra time for sync signals)
--
--  Horiz_sync  ------------------------------------__________--------
--  H_count      0              640           659      755    799
--
gestion_H_Count:PROCESS(clock_25Mhz) BEGIN
        IF(clock_25Mhz'EVENT) AND (clock_25Mhz='1') THEN
          IF (h_count = 799) THEN
            h_count <= (others =>'0');
          ELSE
            h_count <= h_count + 1;
          END IF;
     END IF;
END PROCESS;
gestion_Horiz_sync: PROCESS(clock_25Mhz,h_count) BEGIN
--Generate Horizontal Sync Signal using H_count
 IF(clock_25Mhz'EVENT) AND (clock_25Mhz='0') THEN
        IF (h_count <= 755) AND (h_count >= 659) THEN
                horiz_sync <= '0';
        ELSE
                horiz_sync <= '1';
        END IF;
 END IF;
END PROCESS;
--V_count counts rows of pixels (480 + extra time for sync signals)
--
--  Vert_sync      ------------------------------------------------_________------------
--  V_count         0                                480    493-494         524
--
gestion_V_Count: PROCESS(clock_25Mhz,h_count) BEGIN
        IF(clock_25Mhz'EVENT) AND (clock_25Mhz='1') THEN
          IF (v_count >= 524) AND (h_count >= 699) THEN
                v_count <= (others =>'0');
          ELSIF (h_count = 699) THEN
                v_count <= v_count + 1;
          END IF;
        END IF;
END PROCESS;
```

```
gestion_Vertical_sync:PROCESS(clock_25Mhz,v_count) BEGIN
          IF(clock_25Mhz'EVENT) AND (clock_25Mhz='0') THEN
-- Generate Vertical Sync Signal using V_count
            IF (v_count <= 494) AND (v_count >= 493) THEN
                  vert_sync <= '0';
            ELSE
                  vert_sync <= '1';
            END IF;
          END IF;
END PROCESS;
 pixel_column <= h_count;
 pixel_row <= v_count;
 horiz_sync_out <= horiz_sync;
 vert_sync_out <= vert_sync;
END a;



library IEEE;
use  IEEE.STD_LOGIC_1164.all;
ENTITY VGAtop IS
 PORT (clk_50 : in STD_LOGIC;
          x_rect, y_rect: IN STD_LOGIC_VECTOR(9 DOWNTO 0);
          y_raquG, y_raquD: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
          hsynch,vsynch,red,green,blue : out STD_LOGIC);
END VGAtop;
ARCHITECTURE atop of VGAtop is
COMPONENT VGA_SYNC IS
          PORT(   clock_25Mhz      : IN       STD_LOGIC;
             horiz_sync_out, vert_sync_out   : OUT    STD_LOGIC;
             pixel_row, pixel_column: OUT STD_LOGIC_VECTOR(9 DOWNTO 0));
END COMPONENT;
COMPONENT rect IS PORT(
 row,col,x_rec,y_rec,delta_x,delta_y :in STD_LOGIC_VECTOR(9 DOWNTO 0);
 colorRGB : in  STD_LOGIC_VECTOR(2 DOWNTO 0);
 red1,green1,blue1 : out std_logic);
END component;
signal clk_25,sred,sgreen,sblue,sred1,sgreen1,sblue1,sred2,sgreen2,sblue2 : std_logic;
signal srow,scol : STD_LOGIC_VECTOR(9 DOWNTO 0);
begin
 process(clk_50) begin
  if clk_50'event and clk_50='1' then
            clk_25 <= not clk_25;
          end if;
 end process;
 i1:vga_sync port map(clock_25Mhz =>clk_25, horiz_sync_out=>hsynch,
           vert_sync_out=>vsynch, pixel_row=>srow, pixel_column=>scol);
 balle:rect port map(row=>srow, col=>scol, red1=>sred, green1=>sgreen,
            blue1=>sblue,colorRGB=>"111",
            delta_x=>"0000001010",delta_y=>"0000001100",
            x_rec => x_rect, y_rec => y_rect);
 raquetteG:rect port map(row=>srow, col=>scol, red1=>sred1, green1=>sgreen1,
            blue1=>sblue1,colorRGB=>"100",
           delta_x=>"0000001010",delta_y=>"0000111010",
           x_rec => "0000010110", y_rec(8 downto 1) => y_raquG,
           y_rec(9)=>'0',y_rec(0)=>'0');
 raquetteD:rect port map(row=>srow, col=>scol, red1=>sred2, green1=>sgreen2,
            blue1=>sblue2, colorRGB=>"100",
```

```
                delta_x=>"0000001010",delta_y=>"0000111010",
                x_rec => "1001001000", y_rec(8 downto 1) => y_raquD,
                y_rec(9)=>'0',y_rec(0)=>'0');
  red <= sred or sred1 or sred2;
  green <= sgreen or sgreen1 or sgreen2;
  blue <= sblue or sblue1 or sblue2;
end atop;

library IEEE;
use  IEEE.STD_LOGIC_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
--use ieee.numeric_std.all;

ENTITY rect IS PORT(
  row,col,x_rec,y_rec,delta_x,delta_y :in STD_LOGIC_VECTOR(9 DOWNTO 0);
  colorRGB : in  STD_LOGIC_VECTOR(2 DOWNTO 0);
  red1,green1,blue1 : out std_logic);
END rect;
ARCHITECTURE arect of rect is begin
  PROCESS(row,col,x_rec,y_rec) BEGIN
    if row > y_rec and row < y_rec+delta_y then
      if col >x_rec and col < x_rec+delta_x then
                red1 <= colorRGB(2);
                green1 <= colorRGB(1);
                blue1 <= colorRGB(0);
      else
                red1 <= '0';
                green1 <= '0';
                blue1 <= '0';
      end if;
   else
                 red1 <= '0';
                green1 <= '0';
                blue1 <= '0';
    end if;
  end process;
end arect;
```

ANNEXE III (fichier ucf)

```
net "mclk" loc="t9";
net "mclk" TNM_NET = "mclk";
TIMESPEC "TS_mclk" = PERIOD "mclk" 20 ns HIGH 50 %;
#PORTB=PTOUT1 sur leds
net "PTOUT1<0>" loc="K12";
net "PTOUT1<1>" loc="P14";
net "PTOUT1<2>" loc="L12";
net "PTOUT1<3>" loc="N14";
net "PTOUT1<4>" loc="P13";
net "PTOUT1<5>" loc="N12";
net "PTOUT1<6>" loc="P12";
net "PTOUT1<7>" loc="P11";
#PORTC=PTIN2 sur interrupteurs
net "PTIN2<7>" loc="k13";
net "PTIN2<6>" loc="k14";
net "PTIN2<5>" loc="j13";
net "PTIN2<4>" loc="j14";
net "PTIN2<3>" loc="h13";
net "PTIN2<2>" loc="h14";
net "PTIN2<1>" loc="g12";
net "PTIN2<0>" loc="f12";
#sept segments
	#net "sorties<6>" loc="E14";
	#net "sorties<5>" loc="G13";
	#net "sorties<4>" loc="N15";
	#net "sorties<3>" loc="P15";
	#net "sorties<2>" loc="R16";
	#net "sorties<1>" loc="F13";
	#net "sorties<0>" loc="N16";
#selection afficheurs
	#net "affpdsfaible" loc="D14";
	#net "affpdsfort" loc="G14";
# reset
net "reset" loc="M13";
#VGA
net "hsynch" loc="R9";
net "vsynch" loc="T10";
net "red" loc="R12";
net "blue" loc="R11";
net "green" loc="T12";
```

ANNEXE IV (fichier toptoplogic.vhd)

Voici le fichier le plus haut de la hierarchie :

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

ENTITY TopTopLogic IS
port (

            MCLK:       in  std_logic;
            PCOUT0:     out std_logic_vector(  7 downto 0 );
            PCOUT1:     out std_logic_vector(  7 downto 0 );
            PTIN0:      in  std_logic_vector(  7 downto 0 );
            PTIN1:      in  std_logic_vector(  7 downto 0 );
            PTIN2:      in  std_logic_vector(  7 downto 0 );
            PTOUT0:     out std_logic_vector(  7 downto 0 );
            PTOUT1:     out std_logic_vector(  7 downto 0 );
            PTSTB0:     out std_logic;
            PTSTB1:     out std_logic;
            PTSTB2:     out std_logic;
            RESET:      in  std_logic;
            SLEEP:      out std_logic;
            TMRCLK:     in  std_logic;
            affpdsfaible,affpdsfort : out std_logic;
            sorties : out std_logic_vector(6 downto 0);
            hsynch,vsynch,red,green,blue : out STD_LOGIC
        );
END TopTopLogic;


ARCHITECTURE aTopTopLogic OF TopTopLogic IS
signal EADR: std_logic_vector(6 downto 0 );
signal EALU: std_logic_vector(7 downto 0 );
signal EMCLK_16:  std_logic;
signal EPRC: std_logic_vector(10 downto 0 );
signal EWERAM: std_logic;
signal GP: std_logic_vector(7 downto 0 );
signal PRESET: std_logic;
signal MCLK_16 : std_logic;
signal ROM: std_logic_vector(11 downto 0 );

COMPONENT VGAtop IS
  PORT (clk_50 : in STD_LOGIC;
              x_rect, y_rect: IN std_logic_vector(9 DOWNTO 0);
              y_raquG, y_raquD: IN std_logic_vector(7 DOWNTO 0);
        hsynch,vsynch,red,green,blue : out std_logic);
END COMPONENT VGAtop;

component xilinx_one_port_ram_sync
   generic(
      ADDR_WIDTH: integer:=7;
      DATA_WIDTH: integer:=8
   );
   port(
      clk: in std_logic;
      we: in std_logic;
```

```
      addr: in std_logic_vector(ADDR_WIDTH-1 downto 0);
      din: in std_logic_vector(DATA_WIDTH-1 downto 0);
      dout: out std_logic_vector(DATA_WIDTH-1 downto 0)
    );
end component xilinx_one_port_ram_sync;

COMPONENT PIC_ROM
   port(
      addr: in std_logic_vector(10 downto 0);
      data: out std_logic_vector(11 downto 0)
   );
end COMPONENT PIC_ROM;

component TOPLOGIC
    port (
            EADR:       out std_logic_vector( 6 downto 0 );
            EALU:       out std_logic_vector( 7 downto 0 );
            EMCLK_16:   out std_logic;
            EPRC:       out std_logic_vector( 10 downto 0 );
            EWERAM:     out std_logic;
            GP:         in  std_logic_vector( 7 downto 0 );
            MCLK:       in  std_logic;
            PCOUT0:     out std_logic_vector( 7 downto 0 );
            PCOUT1:     out std_logic_vector( 7 downto 0 );
            PCOUT2:     out std_logic_vector( 7 downto 0 );
            PTIN0:      in  std_logic_vector( 7 downto 0 );
            PTIN1:      in  std_logic_vector( 7 downto 0 );
            PTIN2:      in  std_logic_vector( 7 downto 0 );
            PTOUT0:     out std_logic_vector( 7 downto 0 );
            PTOUT1:     out std_logic_vector( 7 downto 0 );
            PTOUT2:     out std_logic_vector( 7 downto 0 );
            PTSTB0:     out std_logic;
            PTSTB1:     out std_logic;
            PTSTB2:     out std_logic;
            PRESET:     in  std_logic;
            RESET:      in  std_logic;
            ROM:        in  std_logic_vector( 11 downto 0 );
            SLEEP:      out std_logic;
            TESTIN:     in  std_logic;
            TMRCLK:     in  std_logic
        );
end component;

signal s_x, s_y : std_logic_vector(9 downto 0);
signal s_PORTB, s_PTOUT2,s_PCOUT2 : std_logic_vector(7 downto 0);
BEGIN
  i0b: VGAtop PORT MAP (clk_50 => MCLK,
            hsynch=>hsynch,vsynch=>vsynch,
            red=>red,green=>green,blue=>blue,
            y_raquG =>s_PTOUT2, y_raquD=>s_PCOUT2,
            x_rect=>s_x, y_rect=>s_y);
  i1: PIC_ROM PORT MAP (addr =>EPRC,
                        data => ROM);

  i2:xilinx_one_port_ram_sync PORT MAP(
      clk=>MCLK,
      we=>EWERAM,
      addr=>EADR,
```

```
      din=>EALU ,
      dout=>GP   );

  i3: TOPLOGIC PORT MAP (
                      TESTIN=>'0',
                      EADR=>EADR,
                      EALU=>EALU,
            EMCLK_16=>EMCLK_16,
            EPRC=>EPRC,
            EWERAM=>EWERAM,
            GP=>GP,
            MCLK=>MCLK,       PCOUT0(7 downto 2)=>PCOUT0(7 downto
2),
            PCOUT0(1 downto 0)=>s_x(9 downto 8),
            PCOUT1(7 downto 2)=>PCOUT1(7 downto 2),
            PCOUT1(1 downto 0)=>s_y(9 downto 8),
            PCOUT2=>s_PCOUT2,
            PTIN0=>PTIN0,
            PTIN1=>PTIN1,
            PTIN2=>PTIN2,
            PTOUT0=>s_x(7 downto 0),
            PTOUT1=>s_y(7 downto 0),
            PTOUT2=>s_PTOUT2,
            PTSTB0=>PTSTB0,
            PTSTB1=>PTSTB1,
            PTSTB2=>PTSTB2,
            PRESET=>'0',--PRESET,
            RESET=>RESET,
            ROM=>ROM,
            SLEEP=>SLEEP,
            TMRCLK => TMRCLK
        );
              -- sorties
              PTOUT1 <= s_y(7 downto 0);
              PTOUT0 <= s_x(7 downto 0);
END aTopTopLogic ;
```