Utilisation du cœur CQPIC

(Dernière mise à jour 07/06/10)

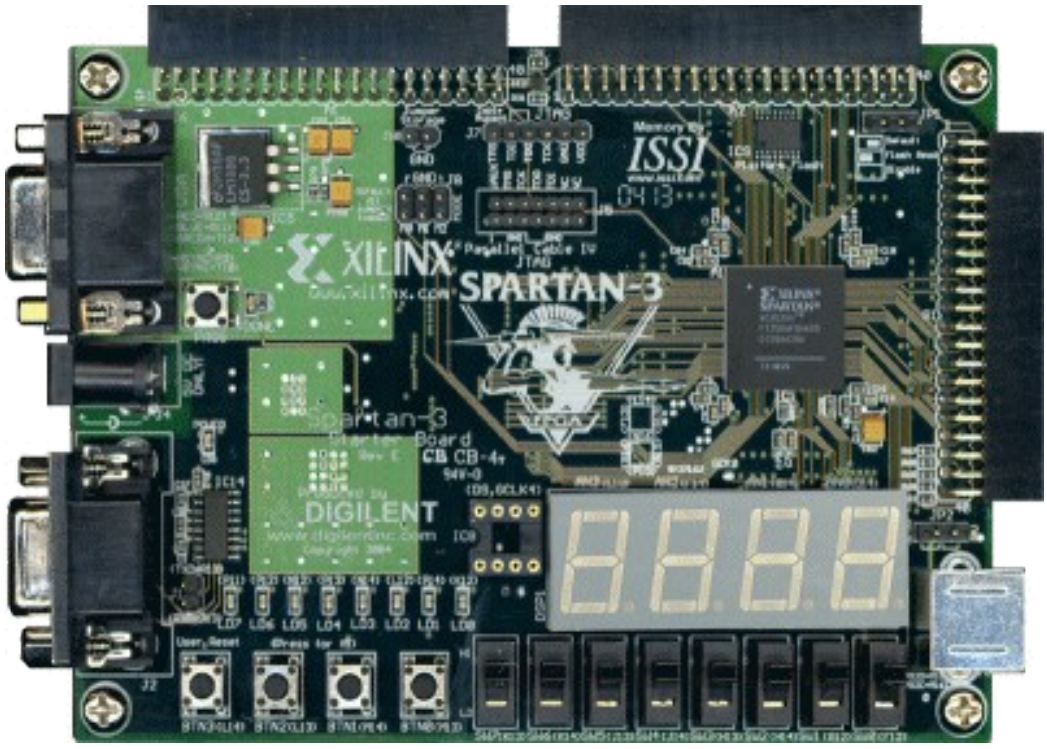
Rapport du tuteur du projet Serge Moutou 2010/2011 (serge.moutou@univ-reims.fr)

Mots clefs : processeur softcore, systèmes mono-puces, systèmes sur puces 8 bits, cœur de processeur, SoC (System On Chip), langage C et FPGA, FPGA Spartan3, Carte Digilent, Écran VGA, PIC®16F84.

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 (CQPIC compatible 16F84) en interaction avec de la logique externe réalisée en VHDL. 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 ainsi :

- 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 16F84, 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).

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

Le CQPIC est un cœur de processeur (ou processeur softcore) compatible avec le PIC® 16F84 de microchip. On pouvait le trouver sur Internet sous la forme d'un fichier "cqpic100d.zip". Comme il devient difficile de le trouver j'ai mis une version (améliorée ?) sur Internet : "http://perso.wanadoo.fr/moutou/ER2/CQPICStart.zip".

Il a été développé par Sumio Morioka (Japon) et publié en Décembre 1999 dans "Transistor Gijutsu Magazine" (sa dernière mise à jour date de 2004)

Choix du coeur

Il existe au moins trois versions de descriptions de cœur PIC® 16F84 sur Internet. La plus à jour me semble être un projet en Verilog disponible sur le site d'opencores : "http://opencores.org/project,risc16f84" avec des mises à jour récentes (2006). Mais ayant des étudiants qui ne pratiquent que VHDL, j'ai décidé d'utiliser le projet **CQPIC**, plus ancien, qui est à l'origine de ce cœur verilog.

J'ai déjà eu l'occasion d'utiliser un cœur de processeur 16F57 et le rapport du projet correspondant se trouve en "http://perso.wanadoo.fr/moutou/ER2/SiliCore1657.pdf". Cet ancien projet, dénommé **silicore1657** dans la suite de ce document, m'a permis d'acquérir un peu d'expérience mise à profit pour le choix du cœur. Il existe en effet un autre cœur VHDL PIC16F84, appelé **PPX** par la suite, (http://opencores.org/project,ppx16) de Daniel Wallner qui a été utilisé et débogué par Patrice Nouel, Maître de Conférence à l'ENSIERB (à la retraite maintenant). Mais mon choix s'est porté sur le CQPIC car il n'utilise pas de PORT bidirectionnel mais décompose les PORTS (comme expliqué plus loin dans ce document). A ce propos, la documentation du Silicore1657 était bien faite car elle mettait en garde l'utilisateur sur ce point. Un cœur bien réalisé doit proposer tout sauf la ROM la RAM et les PORTS parce que leur implémentation est très dépendante du fabriquant du FPGA ciblé. Il n'y a, en effet, aucune norme pour implanter ceux-ci. Personnellement je n'ai jamais réussi à implanter un PORT bidirectionnel sur un composant Xilinx. Cela ne veut pas dire que l'on ne peut pas le faire avec Xilinx, cela veut dire que je ne sais pas le faire (il faut savoir avouer ses faiblesses de temps en temps).

Nous allons commencer par présenter la partie hardware du cœur en nous basant sur la documentation de Microchip.

Architecture du PIC® 16F84

Il s'agit d'un processeur 8 bits avec des instructions codées sur 14 bits.

La version originale de ce PIC avait peu de RAM (68 octets). Une autre de ses limitations est la taille de sa pile : avec une pile de huit niveaux seulement, il

doit être difficile de réaliser un compilateur C.

Ce processeur gère un certain nombre d'interruptions.

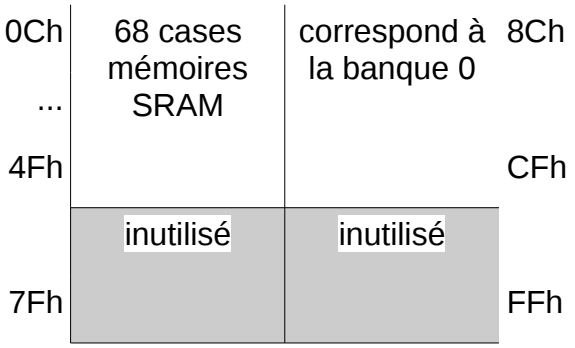
Nous allons commencer par présenter le plus difficile, l'architecture des registres mémoire (Register File dans la terminologie anglo-saxonne que l'on pourrait traduire par dossier/classeur de registres (j'ai trouvé aussi fichier de registres), mais je préfère registres mémoire ou mémoire de registres). 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 deux banques. Nous avons d'abord la série des (vrais) registres jusqu'au **INTCON**. Cette partie de mémoire est suivie par 68 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.

La mémoire du 16F84 est organisée en banques sélectionnées par le(s) bit(s) RP0 (et RP1) du registre **STATUS**. Seul RP0 est utilisé pour le 16F84, soit deux banques.

	File Address	Banque 0	Banque 1	File Address	
	00h	Indirect addr.	Indirect addr.	80h	
bcf status,rp0	01h	TMR0	OPTION_REG	81h	bsf status,rp0
passe en	02h	PCL	PCL	82h	passe en
banque 0	03h	STATUS	STATUS	83h	banque 1
	04h	FSR	FSR	84h	
	05h	PORTA	TRISA	85h	
	06h	PORTB	TRISB	86h	
	07h	---	---	87h	
	08h	EEDATA	EECON1	88h	
	09h	EEADR	EECON2	89h	
	0Ah	PCLATH	PCLATH	8Ah	
	0Bh	INTCON	INTCON	8Bh	



La première partie de cette mémoire (les registres) est parfois appelée **banc de registres** (ou fichier de registres). La partie marquée inutilisée sera en fait utilisée par notre cœur CQPIC. Remarquez l'adresse 07h et 87h inutilisée dans le PIC16F84 mais utilisée par notre cœur.

Les instructions

Il est temps de présenter maintenant l'ensemble des 35 instructions du PIC® 16F84.

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
- b : adresse de bit dans un registre 8 bits (sur 3 bits)
- k : champ littéral (8 ou 11 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	14 bits Opcode	status affected	notes
ADDWF f,d	Additionne W et f	1	00 0111 dfff ffff	C,DC,Z	1,2
ANDWF f,d	ET bit à bit W et f	1	00 0101 dfff ffff	Z	1,2
CLRF f	mise à 0 de f	1	00 0001 1fff ffff	Z	2
CLRW -	mise à 0 de W	1	00 0001 0xxx xxxx	Z	
COMF f,d	Complément de f	1	00 1001 dfff ffff	Z	1,2
DECF f,d	Décrémente f	1	00 0011 dfff ffff	Z	1,2
DECFSZ f,d	Décrémente f (saute si 0)	1,(2)	00 1011 dfff ffff	Z	1,2,3
INCF f,d	Incrémente f	1	00 1010 dfff ffff	Z	1,2

INCFSZ f,d	Incrémente f (saute si 0)	1,(2)	00 1111 dfff ffff	Z	1,2,3
IORWF f,d	Ou inclusif de f	1	00 0100 dfff ffff	Z	1,2
MOVF f,d	déplacement de f (adressage direct)	1	00 1000 dfff ffff	Z	1,2
MOVWF	déplacement de W vers f	1	00 0000 1fff ffff		
NOP -	pas d'opération	1	00 0000 0xx0 0000		
RLF f,d	Rotation gauche avec la retenue	1	00 1101 dfff ffff	C	1,2
RRF f,d	Rotation droite avec la retenue	1	00 1100 dfff ffff	C	1,2
SUBWF f,d	soustrait W de f	1	00 0010 dfff ffff	C,DC,Z	1,2
SWAPW f,d	inverser les quartets dans f	1	00 1110 dfff ffff		1,2
XORWF f,d	Ou exclusif de f	1	00 0110 dfff ffff	Z	1,2

Opérations orientées bits sur les registres					
Mnémonique Opérande	Description	Cycles	14 bits Opcode	status affected	notes
BCF f,b	mise à 0 du bit b dans f	1	01 00bb bfff ffff		1,2
BSF f,b	mise à 1 du bit b dans f	1	01 01bb bfff ffff		1,2
BTFSC f,b	test du bit b 0 de f saute si 0	1,(2)	01 10bb bfff ffff		1,2
BTFSS f,b	test du bit b 0 de f saute si 1	1,(2)	01 11bb bfff ffff		1,2

Opérations littérales (adressage immédiat) et de contrôles					
Mnémonique Opérande	Description	Cycles	14 bits Opcode	status affected	notes
ADDLW k	Addition de W et k	1	11 111x kkkk kkkk	C,DC,Z	
ANDLW k	Et logique de W et k	1	11 1001 kkkk kkkk	Z	
CALL k	appel du sous programme k	2	10 0kkk kkkk kkkk		
CLRWDT -	mise à 0 du timer watchdog	1	00 0000 0110 0100	/TO,/PD	
GOTO k	aller à l'adresse	2	10 1kkk kkkk kkkk		
IORLW	Ou inclusif littéral	1	11 1000 kkkk kkkk	Z	

MOVLW	chargement de W en littéral	1	11 00xx kkkk kkkk		
RETFIE	Retour d'interruption	2	00 0000 0000 1001		
RETLW k	retour avec le littéral dans W	2	11 01xx kkkk kkkk		
RETURN	retour de sous-programme	2	00 0000 0000 1000		
SLEEP	aller au mode standby	1	00 0000 0110 0011	/TO,/PD	
SUBLW k	retire W du littéral	1	11 110x kkkk kkkk	C,DC,Z	
XORLW	Ou exclusif avec littéral	1	11 1010 kkkk kkkk	Z	

Note 1: When an I/O register is modified as a function of itself (e.g., MOVF PORTB, 1), the value used will be that value present on the pins themselves. For example, if the data latch is ’1’ for a pin configured as input and is driven low by an external device, the data will be written back with a ’0’.

2: If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 Module.

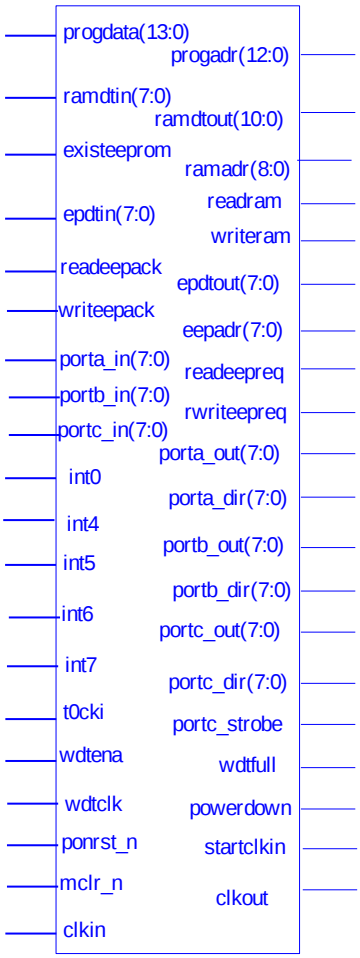
3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

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 piccore is
  generic (
    -- You can change the following parameters as you would like
    STACK_SIZE           : integer := 8;
-- Size of PC stack
    WDT_SIZE             : integer := 255
-- Size of watch dog timer (WDT)
  );
  port (
    -- program ROM data bus/address bus
    progdata : in  std_logic_vector(13 downto 0);  -- ROM read data
    progadr  : out std_logic_vector(12 downto 0); -- ROM address
     -- data RAM data bus/address bus/control signals
    ramdtin    : in  std_logic_vector(7 downto 0); -- RAM read data
    ramdtout : out std_logic_vector(7 downto 0);  -- RAM write data
    ramadr     : out std_logic_vector(8 downto 0);
    -- RAM address; ramadr(8..7) indicates RAM-BANK
    readram    : out std_logic;           -- RAM read strobe (H active)
    writeram : out std_logic;             -- RAM write strobe (H active)
   -- EEPROM data bus/address bus
    existeeprom : in  std_logic;          -- Set to '1' if EEPROM is implemented.
    eepdtin    : in  std_logic_vector(7 downto 0);   -- EEPROM read data
    eepdtout : out std_logic_vector(7 downto 0);   -- EEPROM write data
    eepadr     : out std_logic_vector(7 downto 0);  -- EEPROM address
    readeepreq : out std_logic;                                 -- EEPROM read request (H active)
    readeepack : in  std_logic;                         -- EEPROM read acknowledge (H active)
    writeepreq : out std_logic;                                 -- EEPROM write request (H active)
    writeepack : in  std_logic;                         -- EEPROM write acknowledge (H active)
              -- I/O ports
    porta_in   : in  std_logic_vector(7 downto 0);  -- PORT-A input data
    porta_out : outstd_logic_vector(7 downto 0);  --PORT-A output data
    -- TRISA: PORT-A signal direction (H:input, L:output)
    porta_dir : out std_logic_vector(7 downto 0);
    portb_in : in std_logic_vector(7 downto0);      -- PORT-B input data
    portb_out : out std_logic_vector(7 downto0); -- PORT-B output data
    -- TRISB: PORT-B signal direction (H:input, L:output)
    portb_dir : outstd_logic_vector(7 downto0);
    rbpu : outstd_logic;                -- PORT_B pull-up enable (usually not used)
    -- PORT-B interrupt input
    int0 : in std_logic;                                        -- PORT-B(0) INT
    int4 : in std_logic;                                        -- PORT-B(4) INT
    int5 : in std_logic;                                        -- PORT-B(5) INT
    int6 : in std_logic;                                        -- PORT-B(6) INT
    int7 : in std_logic;                                        -- PORT-B(7) INT
    portc_in : in std_logic_vector(7downto0);       -- PORT-C input data
    portc_out : outstd_logic_vector(7downto0);    -- PORT-C output data
    portc_dir : out std_logic_vector(7downto0);    -- TRISC: PORT-C signal direction
    portc_strob e: out std_logic;                               -- PORT_C strobe
     -- TMR0 Control
    t0cki : in std_logic;                           -- T0CKI (PORT-A(4))
     -- Watch Dog Timer Control
    wdtena    : in  std_logic;                                  -- WDT enable (H active)
    wdtclk : in std_logic;                          -- WDT clock
    wdtfull : outstd_logic;                                     -- WDT-full indicator (H active)
     -- CPU clock stop/start indicators
    powerdown : out std_logic;          -- SLEEP-mode; if H, you can stop system clock clkin
     -- WAKEUP; if H, you should turn on clock for waking up from sleep-mode
    startclkin : out std_logic;
    -- CPU reset
    ponrst_n : in std_logic;                                    -- Power-on reset (L active)
    mclr_n : in std_logic;                          -- Normal reset (L active)
    -- CPU clock
    clkin : in  std_logic;                          -- Clock input
    clkout : out std_logic                          -- Clock output (clkin/4)
  );
end piccore;
```

Pour ce cœur, "progadr" correspond au bus d'adresse de la ROM, tandis que "progdata" correspond au bus de données de cette même ROM. La RAM est gérée quant à elle avec "ramadr" comme bus d'adresses et avec "ramdtout/ramdtin" comme bus de données.

A noter que contrairement au PIC® 16F84, le composant CQPIC utilise une entrée spécifique "t0cki" 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.

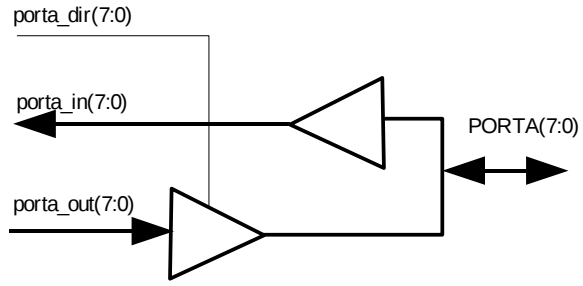
Les deux ports du CQPIC

J'ai fait des modifications sur la gestion des PORTS dans le cœur original qui seront présentées plus loin. Nous nous contenterons dans cette section de généralités.

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 (du FPGA) et donc de l'environnement d'utilisation. Ainsi, la description VHDL des ports est laissée à l'utilisateur du cœur CQPIC.

Un coup d'œil sur le cœur nous montre que chaque port est décomposé en trois ports distincts. Par exemple, pour le **PORTA** on trouve **porta_dir**, **porta_in** et **porta_out**. Cette façon de faire est assez courante dans les "soft core" et j'ai déjà eu l'occasion de la rencontrer lors d'un projet précédent, le silicore1657. La bidirectionnalité des PORTs est gérée par un registre spécial TRIS associé à chacun des ports.

Ces registres spécifiques existent dans le cœur, ils s'appellent **porta_dir** et **portb_dir**. Ils sont tous les deux sur 8 bits contrairement au PIC® 16F84 pour lequel le **PORTA** ne possède que 5 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 **porta_in** et **portb_in** ainsi qu'en ports de sortie **porta_out** et **portb_out**.



Entrées/sorties bidirectionnelle optionnelle

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.

Remarques : pour des raisons de compatibilité le port **porta_dir** n'est pas accessible de manière normale. L'écriture dans **porta_dir** se fait par l'instruction :

```
// en langage C
TRISA=0xFF; // 1 <=> input
```

tandis qu'en assembleur on écrira :

```
; assembleur
bsf STATUS,5 ; select memory bank 1
movlw 255    ;1:input
movwf TRISA ;
```

Comme deuxième remarque, je dirai qu'à l'origine le **PORTA** ne disposait que de 5 bits et que j'ai modifié le cœur et le fichier CQPIC.VHD pour que l'on puisse l'utiliser sur 8 bits. Tous les fichiers correspondants se trouvent dans le répertoire "\CQPICStart\PORTA8" du fichier CQPICStart.zip.

Ma dernière remarque sera pour compléter notre description. Si l'on écrit dans un programme C :

```
PORTA = PORTA; // une seule diode allumee
```

ceci aura comme signification : porta_out <= porta_in. Physiquement, ce qui est à gauche du signe égal, n'est pas la même chose que ce qui est à droite.

La description des PORTs dans cette section n'est pas complète et sera reprise quand nous aurons besoin de les relier à une logique externe.

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

Il existe peu de modèles de mémoire standard dans le monde des FPGA et ASIC. En fait le type le plus commun de mémoire que l'on peut trouver est ce que la norme WHISBONE appelle 'FASM', ou FPGA and ASIC Subset Model. Pour la RAM, sa spécificité est qu'il s'agit d'une mémoire à écriture et lecture synchrone. 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 dataram is
  generic(
    ADDR_WIDTH: integer:=9;
    DATA_WIDTH: integer:=8
  );
  port(
    clk: in std_logic;
```

```
    write: in std_logic;
    read: in std_logic;
    addr: in std_logic_vector(ADDR_WIDTH-1 downto 0);
    datain: in std_logic_vector(DATA_WIDTH-1 downto 0);
    dataout: out std_logic_vector(DATA_WIDTH-1 downto 0)
  );
end dataram;

architecture RTL of dataram 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 (write='1') then
        ram(to_integer(unsigned(addr))) <= datain;
       end if;
       addr_reg <= addr;
    end if;
  end process;
          dataout <= ram(to_integer(unsigned(addr_reg)));
end RTL;
```

La gestion de la RAM dans un système mono-puce n'est pas un problème si simple qu'il n'y parait. Examinons de plus près ce problème et ce qui fait sa difficulté.

Un coup d'œil sur le programme ci-dessus nous indique immédiatement que les adresses mémoires sont sur 9 bits alors que la documentation du PIC16F84 donne des plages inutilisées jusqu'à FFh soit sur 8 bits. En VHDL, donner plus de mémoire à un cœur se fait très facilement s'il y a de la place dans le FPGA pour cela. Comme nous avons décidé d'utiliser un compilateur C, notre problème est de savoir comment faire comprendre au compilateur C que l'on dispose de plus de mémoire que prévu ? Normalement les compilateurs ont une directive de compilation pour cela qu'il n'est pas difficile de trouver en page 60 de la documentation du compilateur HiTech C.

Aller dans :

Project -> Build Options -> project ->onglet "global" et remplir "RAM Range".

Sans directive on obtient :

```
Data space         used     7h (     7) of    44h bytes   ( 10.3%)
```

Avec "default,+50-7f" on obtient :

```
Data space         used     7h (     7) of    74h bytes   (  6.0%)
```

Avec "default,+50-7f,+d0-ff" on obtient :

```
Data space         used     7h (     7) of    A4h bytes   (  4.3%)
```

ce qui montre que l'on peut effectivement augmenter l'espace géré par le compilateur. Nous avons réussi à gérer toutes les adresses sur 8 bits. I nous

reste un 9ème bit à gérer. Mais cela nécessite forcément d'aller regarder dans le cœur comment tout cela est géré, ce que je n'ai pas fait pour le moment. Je pense de toute façon qu'il faut changer de PIC pour prendre un PIC16F87X dans notre projet C, mais ceci n'est pas sans risque puisqu'il y a alors quatre banques et que certain des registres ont été déplacés.

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.

```
-- This file was generated with hex2rom written by Daniel Wallner

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

entity progrom is
	port(
		Clk	: in std_logic;
		romaddr	: in std_logic_vector(12 downto 0);
		romout	: out std_logic_vector(13 downto 0)
	);
end progrom;

architecture rtl of progrom is
	signal A_r : std_logic_vector(12 downto 0);
begin
	process (Clk)
	begin
		if Clk'event and Clk = '1' then
			A_r <= romaddr;
		end if;
	end process;
	process (A_r)
	begin
		case to_integer(unsigned(A_r)) is
		when 000000 => romout <= "00000100101000";	-- 0x0000
		when 000001 => romout <= "00001100010110";	-- 0x0002
		when 000002 => romout <= "11111100110000";	-- 0x0004
		when 000003 => romout <= "00010100000000";	-- 0x0006
		when 000004 => romout <= "00001100010000";	-- 0x0008
		when 000005 => romout <= "00000000110000";	-- 0x000A
		when 000006 => romout <= "00001100011000";	-- 0x000C
		when 000007 => romout <= "00000100110000";	-- 0x000E
		when 000008 => romout <= "00011000000000";	-- 0x0010
		when 000009 => romout <= "00001100010010";	-- 0x0012
		when 000010 => romout <= "00010100001000";	-- 0x0014
		when 000011 => romout <= "00011000000000";	-- 0x0016
		when 000012 => romout <= "00100100101000";	-- 0x0018
		when 000013 => romout <= "00000000101000";	-- 0x001A
		when others => romout <= "--------------";
		end case;
	end process;
```

```
end;
```

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 et qui se trouve dans le répertoire "\CQPICStart\ROM" du fichier CQPICStart.zip.

Mes premiers programmes en C

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. Puisque 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.

Mon premier programme en C a été réalisé pour tester le fonctionnement du cœur CQPIC sur notre carte d'application Digilent.

Notre architecture CQPIC étant une architecture 8 bits avec peu de mémoire RAM, il peut sembler intéressant de commencer par présenter un programme en C mais qui ne contient que de l'assembleur.

Le premier programme simple en C avec assembleur

Le programme présenté montre comment inclure de l'assembleur dans un programme C.

```
#include  <pic1684.h>
//#include <htc.h> serait-il mieux ?
main(void)
{
#asm
Start: bcf _STATUS,5 ; select memory bank 0
movf _PORTA, W ;read PORTA
bsf _STATUS,5 ; select memory bank 1
movwf _TRISA ;Recopie de PORTA
goto Start
#endasm
}
```

Ne prenez pas ce programme à la lettre. Il est donné pour exemple mais fonctionne de manière surprenante.

Le premier programme simple en C pur

Un programme tout simple de test est présenté maintenant : il s'agit tout simplement de recopier le PORTA sur le PORTB. Remarquons aussi que ce programme fonctionne complètement si l'on a pris soin de prendre la version que j'ai réalisée.

```
#include  <pic1684.h>
//#include <htc.h> serait-il mieux ?
main(void)
{ // La gestion de TRISA et TRISB semble importante dans ce coeur
 TRISA = 0xFF; // 8 entrees pour A
 TRISB = 0x00; // 8 sorties pour B
 while(1)
 PORTB = PORTA; // recopie du PORTA dans le PORTB qui allume les LEDs
}
```

Contrairement au Silicore1657 déjà évoqué (projet de l'année précédente), le cœur CQPIC dispose du mécanisme d'interruption que nous allons détailler maintenant avec un exemple simple.

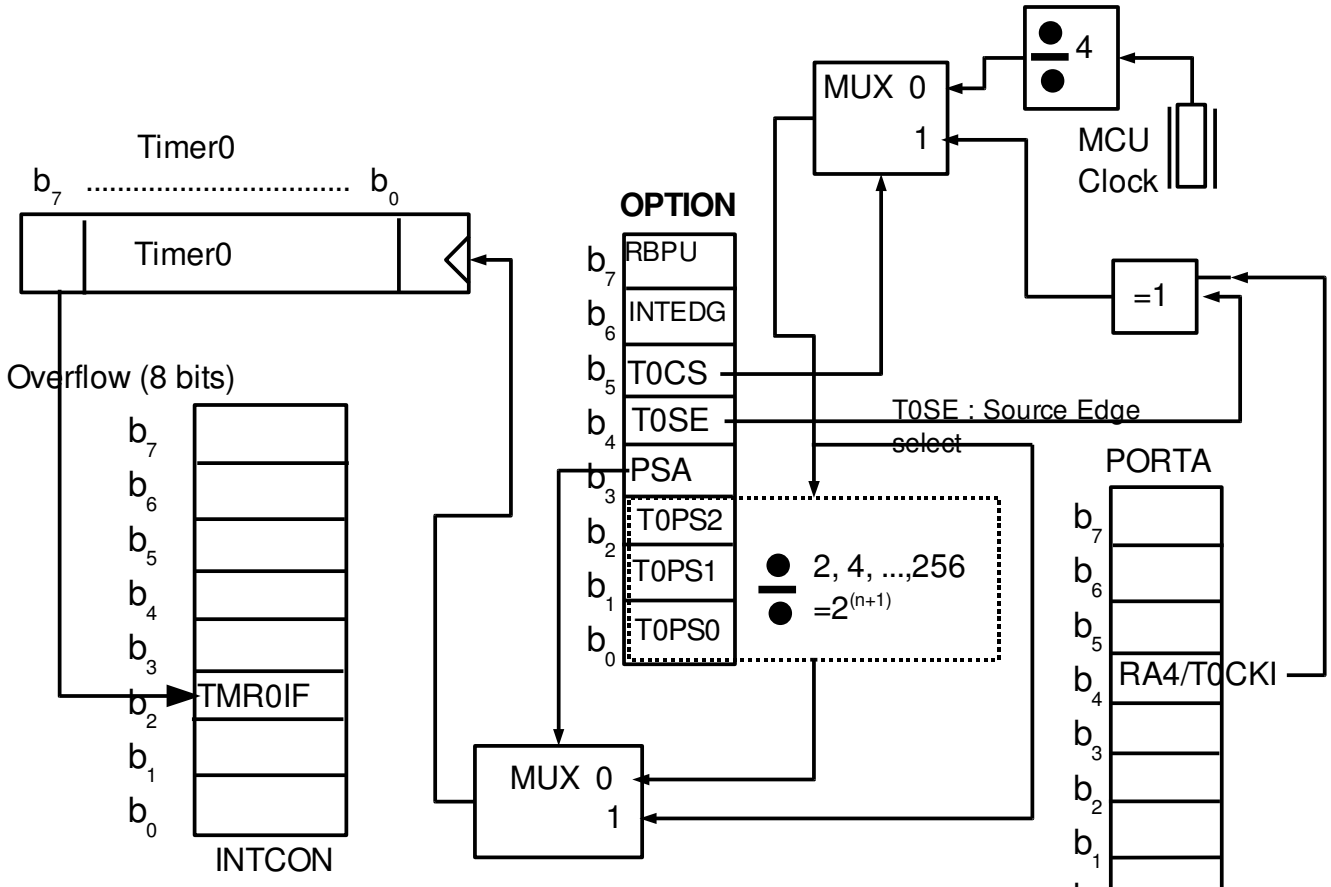
Ajoutons une interruption

Pour tester plus en avant notre cœur CQPIC, je décide d'utiliser une interruption. La plus simple des interruptions à mettre en œuvre étant celle du timer0, nous allons examiner maintenant la documentation correspondante.

La documentation du timer0

Nous présentons maintenant le schéma complet correspondant à ce timer0.

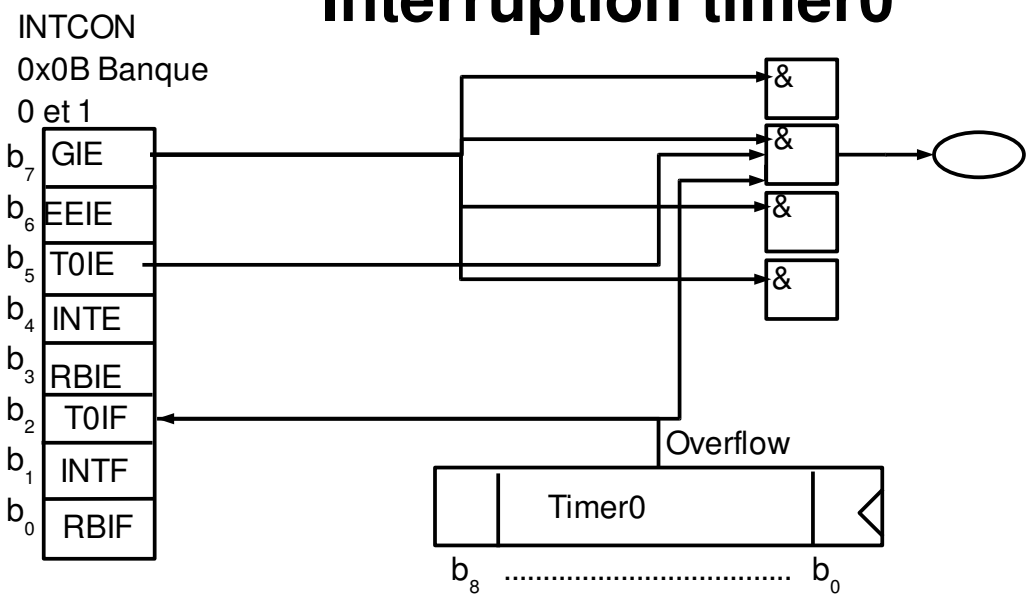
Comme on peut le voir dans la documentation schématique, la gestion du timer0 est essentiellement réalisée par le registre **OPTION**. Seuls les trois bits de poids faible de ce registre sont à positionner à 1 pour avoir une division par 256. Comme on le verra dans le programme, cette division ne suffit pas il faudra encore lui ajouter une division par 16 dans l'interruption même pour que notre chenillar soit visible.



Timer0 dans 16FXXX

Mettre en route le timer0 est une chose, mais déclencher une interruption en est une autre. Il nous faut maintenant documenter ce mécanisme.

Interruption timer0



Pour réaliser l'interruption, il faut réaliser un front montant dans l'ellipse : on en déduit immédiatement les bits à mettre à 1 dans **INTCON**, à savoir T0IE et GIE.

Le programme

Le programme correspondant est donné maintenant.

```
#include  <pic1684.h>
void interrupt decalage(void);

unsigned char nb;
main(void)
{
  TRISA = 0xF9; // 6 entrees, 2 sorties pour A
  TRISB = 0x00; // 8 sorties pour B
  OPTION = 0x07; // prescaler 256 , entree sur quartz
  INTCON = 0xA0;  // autorise l'interruption timer
  PORTB = 0x01; // une seule diode allumee
  TMR0 = 0x00 ;
  nb=0;
  while(1)
    {
   // on ne fait rien que recopier sur 2 segments la valeur de SW1
      if ((PORTA & 0x01) == 1) PORTA = 0x06;
    }
}

void interrupt decalage(void)
{
  nb++;
  //TMR0 = 0x00; //c'est fait car overflow
  if (!(nb % 16))
  PORTB = (PORTB << 1) ;
  if (PORTB == 0x00)   PORTB = 0x01;
  T0IF = 0;  // acquittement interruption
}
```

Remarquez le "`if (!(nb % 16))`" qui réalise la division par 16 dans l'interruption. C'est absolument nécessaire pour notre cœur qui fonctionne à 50 MHz pour voir les LEDs allumées se déplacer.

Le lecteur, fin programmeur, me fera remarquer, à juste titre, que la division est par 16 qui est une puissance de 2. Le calcul du reste peut alors se faire avec un masque. C'est certainement plus optimisé que l'opérateur modulo mais le projet de pong ne nécessite pas ce genre d'optimisation.

Il est maintenant grand temps de revenir sur notre problème original, à savoir interfacer 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).

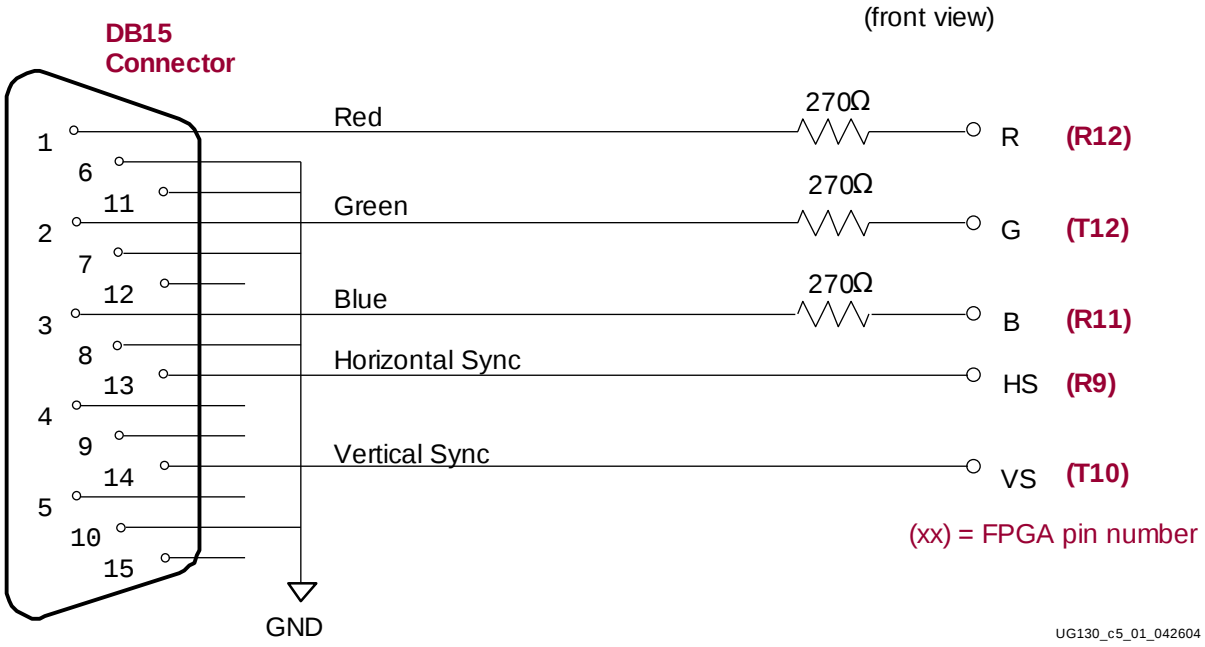
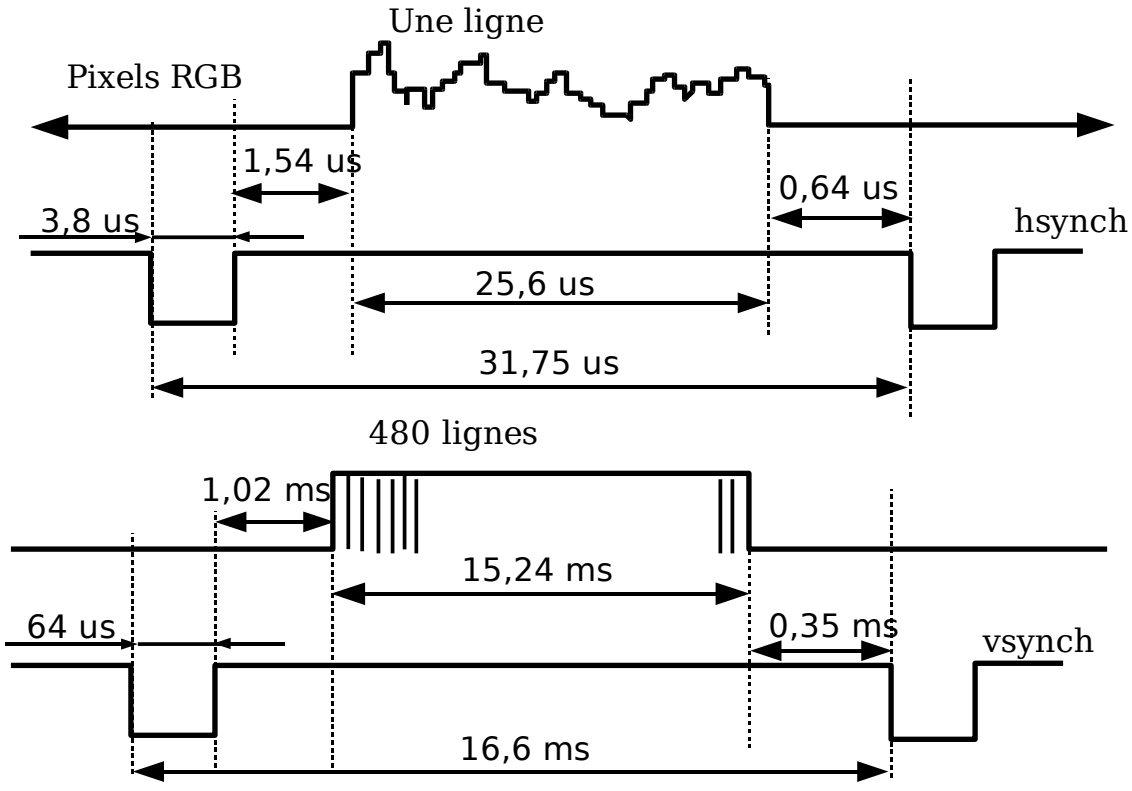


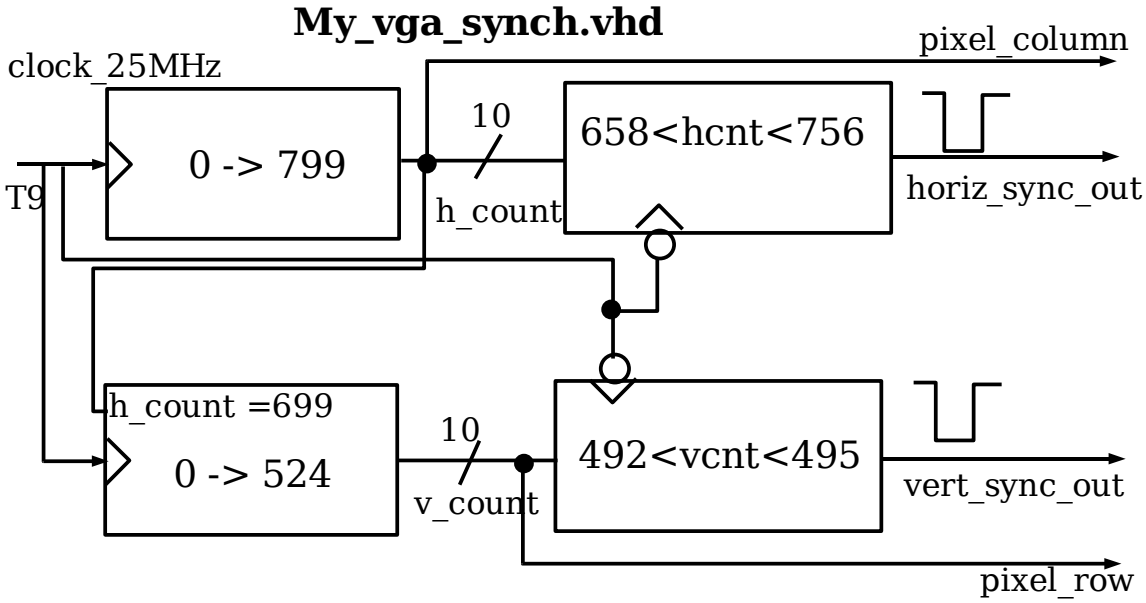
Figure 5-1: **VGA Connections from Spartan-3 Starter Kit Board**

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.



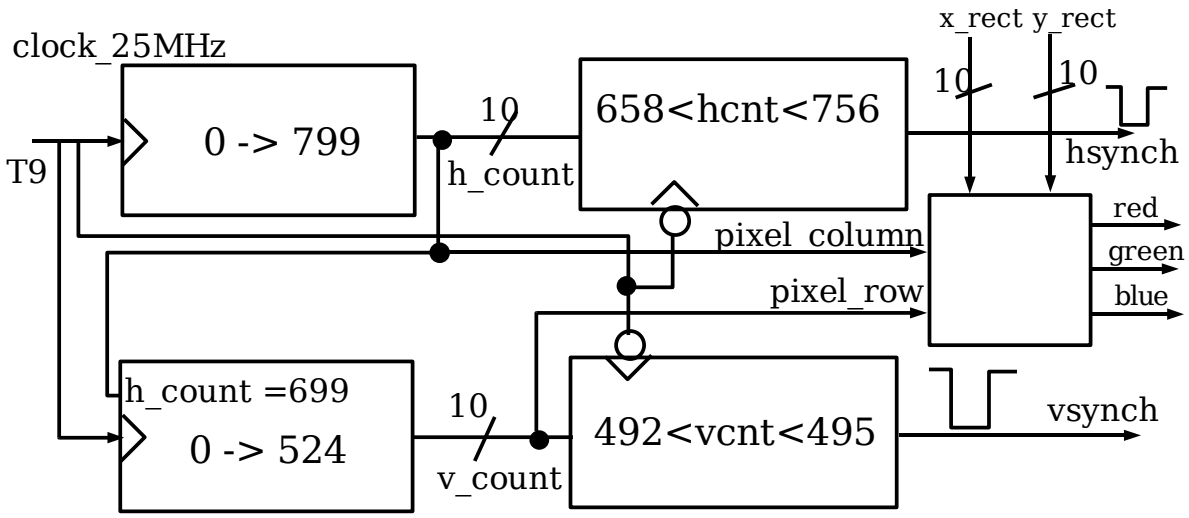
Nous pouvons donc réaliser les deux signaux hsynch et vsynch à l'aide de l'architecture ci-dessous qui représente partiellement le contenu du fichier My_vga_synch.vhd :



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

Comme nous le montre le schéma ci-dessus, nous avons besoin de deux fois 10 bits pour piloter nos coordonnées de balles. Une autre façon de dire les choses est qu'il nous faut 4 ports de 8 bits en sortie. Or, nous ne disposons que de deux PORTs de 8 bits en sortie (et encore parce que j'ai étendu le **PORTA** originel. C'est insuffisant. Il nous faut donc réfléchir sur la façon dont on va pouvoir connecter toutes ces données en sortie.

Pourquoi seulement deux PORTs en sortie

Puisque j'ai modifié le cœur CQPIC pour augmenter le nombre de bits du **PORTA**, j'ai été amené à lire son code VHDL et je suis tombé sur :

```
-- VHDL
for I in 0 to 7 loop
  if (trisa_reg(I) = '1') then
    ramin_node(I) := portain_sync_reg(I);-- PORT A (when input mode)
  else
    ramin_node(I) := portaout_reg(I);-- PORT A (when output mode)
  end if;
end loop;
```

Ce code montre que TRISA est pris en compte en interne dans le cœur CQPIC. Ceci peut sembler normal, mais n'est pas conforme à la seule expérience que j'avais avec le Silicore1657 : puisque TRISA était sorti à l'extérieur il pouvait servir de PORT supplémentaire.

Si vous avez l'idée d'utiliser TRISA comme PORT supplémentaire, vous aurez un fonctionnement étrange d'un programme C :

```
TRISA = PORTA ;
```

Physiquement, on relie **PORTA** (**ra_out**) à des interrupteurs et **TRISA** (**ra_dir**) à des LEDs. Si **PORTA** est à OxFF (par les interrupteurs extérieurs), tout interrupteur mis à 0 éteindra correctement la LED correspondante (reliée à **TRISA**). Mais l'inverse ne marche plus : si ce même interrupteur est remis à 1 on n'allume plus la LED correspondante.

Conclusion :	Les deux PORTs TRIS (physiquement **porta_dir** et **portb_dir**) du cœur CQPIC ne peuvent en aucun cas être utilisé comme PORT supplémentaire de sortie.

Comment contourner cet obstacle ?

Ajouter un PORTC dans le CQPIC

(Le coeur correspondant se trouve dans le répertoire "\CQPICStart\PORTC" du fichier CQPICStart.zip).

Après avoir passé plusieurs heures à étendre à 8 bits le **PORTA** (au lieu des 5 initiaux), j'ai repéré à peu près tous les endroits à modifier pour ajouter un **PORTC** au CQPIC. Le jeu en vaut-il la chandelle ? Cela ne peut pas suffire à résoudre nos problèmes, puisqu'on aura alors en sortie que trois PORTs sur les huit qu'il nous faut. J'ai finalement réalisé ce changement dans le cœur et j'en ai profité pour ajouter un signal "strobe" qui accompagne ce port supplémentaire (voir section suivante). C'est ce signal strobe qui va nous permettre de résoudre notre problème.

Notez que cette solution impose de déclarer **PORTC** et **TRISC** dans les programmes C. Je déconseille une modification du fichier pic1684.h Faites plutôt comme ceci :

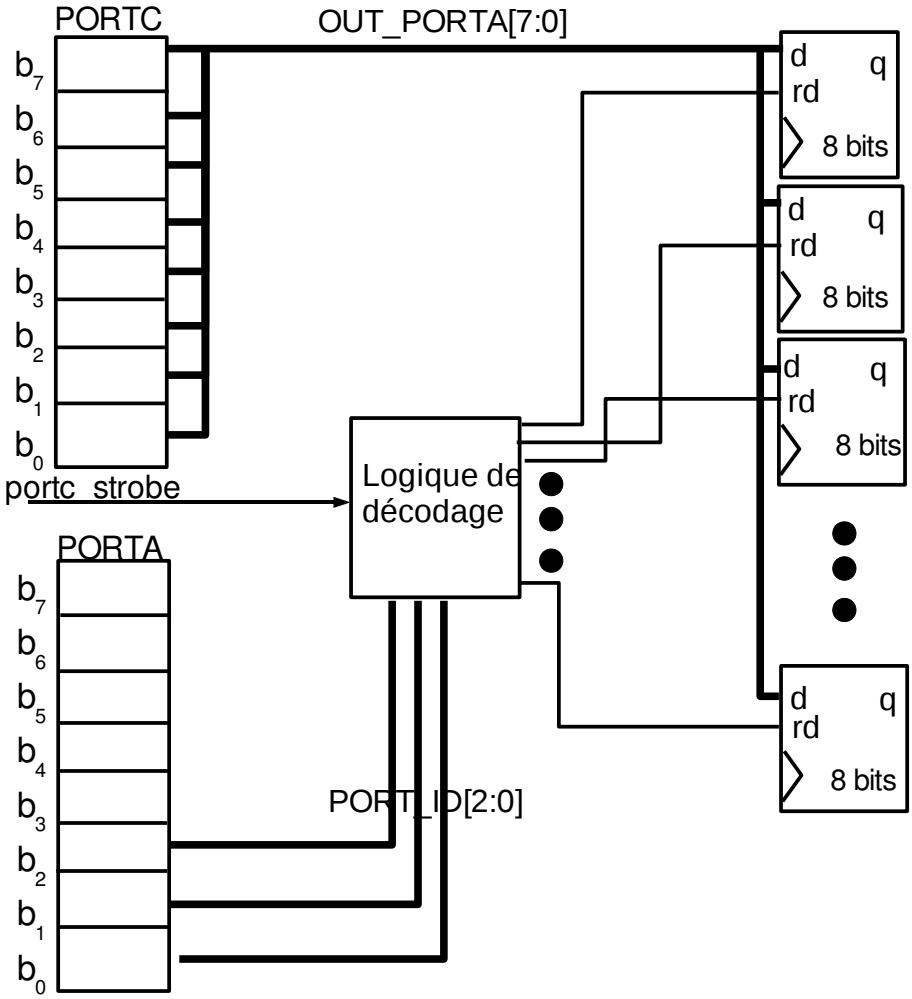
```
#include  <pic1684.h> // ou #include <hct.h>
volatile unsigned char  PORTC @ 0x07;
volatile unsigned char   TRISC @ 0x87;
```

C'est simple, mais cela ne permet pas un accès à des bits individuels (mais nous n'avons pas besoin de cela dans notre projet).

Approfondissons cette solution.

Ajouter un "strobe" au PORTC et augmenter le nombre de ports

Un signal "strobe" est une sortie qui indique une écriture dans le PORTC. Ajouter ce signal strobe a été en fait le plus délicat du travail. L'intérêt d'un tel signal est de permettre facilement la démultiplication des PORTs (jusqu'à 256). Nous présentons ci-dessous un exemple qui montre comment étendre à huit le nombre de PORTs. On met la valeur du port véritable de destination dans **PORTA** puis on met dans **PORTC** la valeur à écrire : le "strobe" lui sera alors envoyé et il recopiera ce qu'il y a dans **PORTC**.



Il est peut être plus facile de raisonner à partir de ce schéma. Le **PORTC** (a gauche) est relié à tous les ports ajoutés (à droite) et la logique de décodage envoie le strobe dans le port qui doit recopier ce qu'il y a dans **PORTC**.

Conclusion :	Le fait que le cœur original ne gère pas de signal "strobe" est un grand handicap si l'on veut ajouter de la logique externe. C'est une des raisons qui m'a fait me décider à ajouter un PORTC et son strobe associé.

Le programme VHDL correspondant au schéma de la page précédente est présenté maintenant. Il comporte deux entrées PORTA et PORTC pour réaliser quatre ports de sorties p1, pB, pC, pD.

```
-- increase the number of available ports with the new portC and
its strobe
-- ver1.10b, 2010/05/22 (Serge Moutou)
library ieee;
use ieee.std_logic_1164.all;
entity ports is
```

```
      port(
        clk : in std_logic;
        strobeC_in : in std_logic;
        portA : in std_logic_vector(7 downto 0);
        portC : in std_logic_vector(7 downto 0);
        pA,pB,pC,pD : out std_logic_vector(7 downto 0) --new ports
      );
end ports;
architecture BehPorts of ports is
signal Internal_strobes : std_logic_vector(7 downto 0);

begin
  dmux_Strobe:process(portA,strobeC_in) begin
    case portA is
      when "00000001" => Internal_strobes(0) <= strobeC_in;
      when "00000010" => Internal_strobes(1) <= strobeC_in;
      when "00000100" => Internal_strobes(2) <= strobeC_in;
      when "00001000" => Internal_strobes(3) <= strobeC_in;
      when others => internal_strobes <= (others =>'0');
      end case;
  end process;

  port_A:process(clk)begin
    if clk'event and clk='1' then
         if Internal_strobes="00000001" then
              pA<=portC;
            end if;
    end if;
  end process;
  port_B:process(clk)begin
    if clk'event and clk='1' then
      if Internal_strobes="00000010" then
              pB<=portC;
            end if;
    end if;
  end process;
  port_C:process(clk)begin
    if clk'event and clk='1' then
      if Internal_strobes="00000100" then
              pC<=portC;
            end if;
    end if;
  end process;
  port_D:process(clk)begin
    if clk'event and clk='1' then
      if Internal_strobes="00001000" then
              pD<=portC;
            end if;
    end if;
  end process;
end BehPorts;
```

Programmation du nouveau cœur avec écran VGA

(L'ensemble se trouve dans le répertoire "\CQPICStart\VGA_Pong" du fichier CQPICStart.zip).

Nous avons déjà présenté quelques programmes en C, mais nous allons

examiner ce qu'il faut changer dans ces programmes pour gérer les nouveaux PORTs.

Programmation en C

Nous commençons par présenter un sous-programme qui écrit une valeur 16 bits dans deux des nouveaux PORTs.

Le sous-programme "setX"

Une version en C pur est montrée ci-dessous :

```
void set(unsigned int x){
  TRISA=0x00;TRISC=0x00;
  PORTA=1;
  PORTC=x; //poids faible
  PORTA=2;
  PORTC=x>>8;//poids fort
}
```

Le contenu de ce programme est complètement déterminé par la partie matérielle.

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 <pic1684.h>
volatile unsigned char      PORTC  @ 0x07;
volatile unsigned char  TRISC      @ 0x87;
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){
  TRISA=0x00;TRISC=0x00;
  PORTA=1; //poids faible
  PORTC=x;
  PORTA=2;//poids fort
  PORTC=x>>8;
```

```
}

void setY(unsigned int y){
  TRISA=0x00;TRISC=0x00;
  PORTA=4; //poids faible
  PORTC=x;
  PORTA=8;//poids fort
  PORTC=x>>8;
}

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

Remarquez l'utilisation du timer0 sans interruption pour faire la temporisation.

Programme complet de gestion simple de la balle

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

```
#include <pic1684.h>
volatile unsigned char      PORTC  @ 0x07;
volatile unsigned char  TRISC      @ 0x87;
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){
  TRISA=0x00;TRISC=0x00;
  PORTA=1; //poids faible
  PORTC=x;
  PORTA=2;//poids fort
  PORTC=x>>8;
}

void setY(unsigned int y){
```

```
  TRISA=0x00;TRISC=0x00;
  PORTA=4; //poids faible
  PORTC=x;
  PORTA=8;//poids fort
  PORTC=x>>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

Comme nous voulons deux raquettes, il nous faut deux coordonnées verticales sur 9 bits (soit 4 ports 8 bits). Nous voulons aussi deux coordonnées de balles soit encore 4 ports de 8 bits. Cela fait en tout 8 ports. Nous avons déjà eu l'occasion de montrer comment on pouvait augmenter le nombre de PORTs.

Solution simple sans bord

Nous avons l'intention dans cette section de mettre en œuvre le programme VHDL de la section Ajouter un "strobe" au PORTC et augmenter le nombre de ports. Puisque ce programme VHDL ne proposait que quatre PORTs de sortie nous allons présenter un ensemble fonctionnant mais avec des positions de raquette fixes.

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

position X de la	x_rect<9:8>	PORTB<1:0>
balle	x_rect<7:0>	PORTA<7:0>
position Y de la	y_rect<9:8>	PORTC<1:0>
balle	y_rect<7:0>	PORTD<7:0>

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 dans ce morceau de programme sont omises.

Voici maintenant le programme C permettant le rebond sur les raquettes. Vous pouvez remarquer que les coordonnées des raquettes sont calculées en fonction des interrupteurs sur le PORTB, mais ne sont pas mises à jour dans le matériel puisque le matériel ne peut pas le faire.

```
#include <pic1684.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(RB2==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 (RB0) if (raqG_y<215) raqG_y++;
                                if (RB1) if (raqG_y>0) raqG_y--;
                                //PORTD=raqG_y;
                                if (RB6) if (raqD_y<215) raqD_y++;
                                if (RB7) if (raqD_y>0) raqD_y--;
                                //PORTE=raqD_y;
                                wait(250);
                                wait(250);
                        }
                }
}
```

Comme on peut le voir nous avons choisi le **PORTB** connecté aux interrupteurs sw0,sw1,sw6 et sw7 de la carte, pour faire descendre et monter les raquettes et sw2 (RB2) pour le départ.

Travail à réaliser

Comprendre la partie matérielle. Vous devez être capable de dessiner les composants et leurs liaisons à partir des fichiers VHDL donnés dans le répertoire "\CQPICStart\VGA_Pong" du fichier CQPICStart.zip..

Étendre la partie matérielle pour gérer le déplacement des deux raquettes. On choisira une coordonnée Y de chacune des raquettes, au choix, sur 8 bits (comme dans le projet silicore1657 de l'année dernière) ou sur 9/10 bits.

Explorer s'il n'est pas possible d'utiliser l'algorithme de tracé de segment de droite Bresenham pour les trajectoires de balles. Cet algorithme est expliqué dans le WIKI : http://fr.wikipedia.org/wiki/Algorithme_de_trac %C3%A9_de_segment_de_Bresenham

Il est possible de trouver directement une version en C : tapez Bresenham en C dans google.

Gérer un affichage des scores dans la partie basse de l'écran, soit avec une mémoire de caractères, soit avec des afficheurs sept segments à construire.

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 plus simple semble être d'utiliser la mémoire RAM qui est sur la carte Digilent pour y mettre les programmes. L'avantage est que cette RAM apparaît avec "Impact" au moment du téléchargement et peut donc être téléchargée seule.

Utiliser un cœur plus récent et plus performant et plus à jour, par exemple

"http://opencores.org/project,risc16f84", même si de nombreux problèmes évoqués dans ce document resteraient irrésolus.

Conclusion (provisoire puisque le projet ne sera donné qu'à partir de Septembre 2010)

Les projets sur des systèmes mono-puces nécessitent des connaissances diverses que peu de nos étudiants maîtrisent. Mais les étudiants ont été aidés cette année car le tuteur avait débogué le projet.

Ce projet nécessite de bonne 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. Combien d'étudiants de nos promotions auraient pu trouver seuls les directives pour gérer la RAM supplémentaire de notre cœur ? Combien d'étudiants auraient pu trouver que le compilateur HiTech C est bogué ? En effet ce compilateur ne compile pas correctement l'instruction "if (sortie == 0)" ou même "if(!sortie)" si sortie est un unsigned int.

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 (avec celui que j'ai rajouté). Si vous vous trouvez dans la situation contraire (besoin de nombreux PORTS), il vous faudra être imaginatif.

Ce projet a vraiment montré l'intérêt de disposer d'un système mono-puce libre. En effet, disposer du code source nous a été d'un grand secours puisque nous avons été amené à le modifier en partie pour ajouter :

- 8 bits au **PORTA** pour le passer sur huit bits

- un **PORTC** bidirectionnel de 8 bits

- un signal strobe lié au PORTC qui indique quand on écrit dans ce port.

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.

Pré-rapport réalisé avec OpenOffice 2.4.1

ANNEXE I (transformer un fichier HEX en VHDL)

Le programme C++ pour transformer le fichier hex en fichier VHDL n'est pas donné car un petit peu trop long. Il se trouve dans le répertoire "\CQPICStart\ROM" du fichier "http://perso.wanadoo.fr/moutou/ER2/CQPICStart.zip".

Attention : Je n'ai pas pris le fichier original hex2vhd.c proposé avec le cœur CQPIC car je n'ai pas réussi à le faire fonctionner correctement. J'ai en fait pris le fichier hex2rom.cpp qui est présent dans un cœur concurrent, à savoir le cœur PPX. Ce fichier a été légèrement modifié pour que le programme généré puisse aller directement dans notre projet VHDL.

Un fois compilé mon utilitaire s'appelle hex2rom.exe (sous windows) ou hex2rom sous Linux.

L'utilisation est faite par la ligne de commande :

```
hex2rom demo16F84.hex progrom 13l14s >progrom.vhd
```

si le fichier à convertir s'appelle demo16F84.hex.

Le fichier généré s'appelle progrom.vhd et a "progrom" comme nom d'entité.

13 est le nombre de bit d'adresses,

l désigne le fait que l'on utilise la convention little indian (b désignerait big indian)

14 est le nombre de bits de données dans la ROM.

ANNEXE II (gestion du VGA)

se trouve dans le répertoire "\CQPICStart\VGA_Pong" du fichier CQPICStart.zip.

```
-- ************** 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)

```
#PORTB sur leds
net "rb0_out" loc="K12";
net "rb1_out" loc="P14";
net "rb2_out" loc="L12";
net "rb3_out" loc="N14";
net "rb4_out" loc="P13";
net "rb5_out" loc="N12";
net "rb6_out" loc="P12";
net "rb7_out" loc="P11";
#PORTB sur interrupteurs
net "rb_in<7>" loc="k13";
net "rb_in<6>" loc="k14";
net "rb_in<5>" loc="j13";
net "rb_in<4>" loc="j14";
net "rb_in<3>" loc="h13";
net "rb_in<2>" loc="h14";
net "rb_in<1>" loc="g12";
net "rb_in<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";
# clock
net "clkin" loc="T9";
net "clkin" TNM_NET = "clkin";
TIMESPEC "TS_mclk" = PERIOD "clkin" 20 ns HIGH 50 %;
# reset
net "mclr_n" loc="M13";
net "ponrst_n" loc="M14";
#VGA
net "hsynch" loc="R9";
net "vsynch" loc="T10";
net "red" loc="R12";
net "blue" loc="R11";
net "green" loc="T12";
```

ANNEXE IV (fichier pong.vhd)

Voici le fichier le plus haut de la hiérarchie(se trouve dans le répertoire "\CQPICStart\VGA_Pong" du fichier CQPICStart.zip) :

```
library IEEE;
use  IEEE.STD_LOGIC_1164.all;

entity pong is
  PORT (
       rb_in         : in  std_logic_vector(7 downto 0);
       ponrst_n      : in  std_logic;
       mclr_n        : in  std_logic;
       clkin         : in  std_logic;
       hsynch,vsynch,red,green,blue : out STD_LOGIC
       );
end pong;
architecture beh_pong of pong is
       component cqpic
       port (
              ra_in         : in  std_logic_vector(7 downto 0);
              rb_in         : in  std_logic_vector(7 downto 0);
              ponrst_n      : in  std_logic;
              mclr_n        : in  std_logic;
              clkin         : in  std_logic;
              wdtena        : in  std_logic;
              wdtclk        : in  std_logic;
              ra_out        : out  std_logic_vector(7 downto 0);
              rb_out        : out  std_logic_vector(7 downto 0);
              ra_dir        : out  std_logic_vector(7 downto 0);
              rb_dir        : out  std_logic_vector(7 downto 0);
              rc_out, rc_dir      : out  std_logic_vector(7 downto 0);    --
PORT-C  data
              rc_in : in std_logic_vector(7 downto 0);
              rd_out : out  std_logic_vector(7 downto 0);   -- (added PORT-D
data)
              re_out : out  std_logic_vector(7 downto 0);   -- (added PORT-E
data)
              clkout        : out std_logic;
              wdtfull             : out std_logic;
              powerdown     : out std_logic;
              startclkin    : out std_logic
       );
       end component;
       component VGAtop
       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;
       signal s_x_rect, s_y_rect : STD_LOGIC_VECTOR(9 DOWNTO 0);
       signal s_rb, s_re : STD_LOGIC_VECTOR(5 DOWNTO 0);
begin
  core : cqpic port map(
       ra_in => "00000000",
       rb_in => rb_in,
       rc_in => "00000000",
       ponrst_n => ponrst_n,
       mclr_n => mclr_n,
       clkin => clkin,
       wdtena => '0',
```

```
        wdtclk => '0',
        ra_out => s_x_rect(7 downto 0),
        rb_out(1 downto 0) => s_x_rect(9 downto 8),
        rb_out(7 downto 2) => s_rb,
        rd_out => s_y_rect(7 downto 0),
        re_out(1 downto 0) => s_y_rect(9 downto 8),
        re_out(7 downto 2) => s_re
        );
    vga:VGATop port map (
     clk_50 => clkin,
        x_rect => s_x_rect,
        y_rect => s_y_rect,
        y_raquG => "00000000",
        y_raquD => "00000000",
        hsynch => hsynch,
        vsynch => vsynch,
        red => red,
        green => green,
        blue => blue
        );
end beh_pong;
```