

# Utilisation du cœur "CoreAtMega8"

(Dernière mise à jour 03/09/10)

Serge Moutou

Institut Universitaire de Technologie - Genie Électrique et Informatique Industrielle  
9, rue de Québec - BP 396 - 10026 TROYES cedex France  
([serge.moutou@univ-reims.fr](mailto: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, ATMEGA8®.

Pour accompagner ce document, télécharger le fichier :

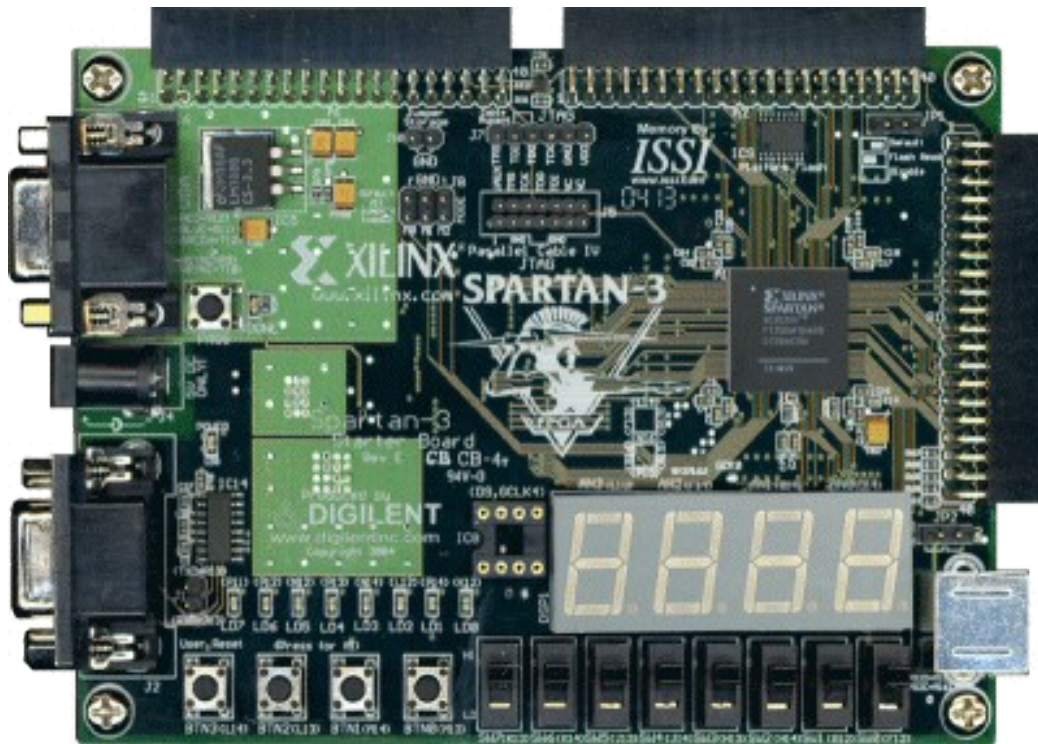
[http://moutou.pagesperso-orange.fr/ER2/ATMega8\\_pong\\_VGA.zip](http://moutou.pagesperso-orange.fr/ER2/ATMega8_pong_VGA.zip)

## 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 en interaction avec de la logique externe réalisée en VHDL. Le processeur soft core sera appelé **CoreAtMega8** dans la suite de ce document. Il est compatible avec ATMEGA8® de chez Atmel. La programmation du cœur devra si possible, se faire en langage C.

La carte ciblée est une carte Digilent contenant un FPGA Xilinx, la carte **Spartan-3 Starter Board** voir :

<http://www.digilentinc.com/Products/Catalog.cfm?NavPath=2,400&Cat=10>



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/>).
- AVRStudio ([http://www.atmel.com/dyn/products/tools\\_card.asp?tool\\_id=2725](http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2725)) pour développer les programmes pour les AVR, avec soit l'assembleur par défaut, soit le compilateur C du GNU qui est fourni avec AVRStudio. Si ce n'est pas le cas, il est facile à trouver et installer.

Ce projet va être donné à trois étudiants de deuxième année de DUT Génie Électrique pour une durée de 80 heures.

## Choix du coeur

Il existe plusieurs versions de descriptions de cœur compatibles avec les RISC de chez Atmel sur Internet. J'ai tendance à choisir les cœurs de processeur en fonction de l'épaisseur de leur documentation. Cela avait déjà été le cas avec le cœur de processeur PIC16C57 l'année dernière (le rapport du projet correspondant se trouve en "<http://perso.wanadoo.fr/moutou/ER2/SiliCore1657.pdf>" pour l'année 2009/2010).

Même s'il existe d'autre cœurs concurrents : [http://opencores.org/project,avr\\_core](http://opencores.org/project,avr_core) ou encore un autre Atmel AVR ATtiny261/461/861 : <http://opencores.org/project,avrtinyx61core>, j'ai choisi celui-ci car il est accompagné d'un cours. Le **CoreAtMega8** est un cœur de processeur (ou processeur softcore) compatible avec le ATMEGA8® de chez Atmel. Il a été développé par Dr. Juergen Sauermann (Allemagne) et publié en Janvier 2010 chez OpenCore sous forme d'un cours pour apprendre à développer un "soft processor" en VHDL ([http://opencores.org/project,cpu\\_lecture](http://opencores.org/project,cpu_lecture)).

Un cœur bien réalisé doit proposer tout sauf la ROM la RAM et les PORTS parce que leur implémentation est très dépendante du fabricant 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 Atmel.

## **Architecture d' ATMEGA8® de chez Atmel**

Il s'agit d'un processeur 8 bits avec des instructions codées sur 16 bits ou 32 bits. Ce processeur gère un certain nombre d'interruptions.

### ***Architecture des registres mémoires***

Il existe 3 espaces distincts en RAM :

- Les 32 premières adresses correspondent aux 32 registres. Ces registres sont directement reliés à l'ALU et tous les calculs ne peuvent être effectués qu'à partir de ces registres (addition, soustraction, multiplication, opérations logiques, tests ).
- L'espace IO de l'adresse 0x20 à 0x5F. Dans cet espace se trouvent beaucoup de registres de gestion du matériels (les ports, Timers, UART,...). Pour accéder à ces registres les instructions in et out utilisent l'adresse 0 pour le premier registre et non l'adresse 0x20. Les instructions cli et sbi permettent de mettre à 0 ou à 1 un bit d'un registre
- Enfin à partir de l'adresse 0x60 se trouve la mémoire à proprement parler (le ATMEGA16 possède 1ko de RAM). Pour accéder à tout l'espace mémoire (Registre et zone I/O incluse) les instructions ST,LD et STS et LDS peuvent être utilisées. L'adressage peut être direct (STS ou LDS) ou indirect (ST et LD). Un adressage indirect utilise les registres X, Y ou Z (respectivement les registres R27:R26, R29:R28 et R31:R30) pour pointer sur une zone mémoire.

Les deux premiers espaces sont désignés en anglais par "register file". On utilisera indistinctement : mémoire de registres, fichier de registres et banc de registres pour les désigner en français.

### ***Quelques registres importants du banc de registres***

Ce tableau (partiel) des registres respecte les fichiers d'inclusion du compilateur avr-gcc. Par exemple, dans la documentation officielle, le bit  $b_0$  du PORTB s'appelle PORTB0 tandis que dans le fichier d'inclusion du GNU-C il s'appelle PB0. **Les registres grisés sont implantés dans notre architecture** et tout le reste est libre.

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x3F(0x5F)	SREG	I	T	H	S	V	N	Z	C
0x32(0x52)	TCTN0	Timer0 8 bits							
0x2D(0x4D)	TCTN1H	Timer1 8 bits de poids fort							
0x2C(0x4C)	TCTN1L	Timer1 8 bits de poids faible							
0x23(0x43)	OCR2	Timer/Counter2 output compare register							
0x21(0x41)	WDTCR	-	-	-	WDCE	WDE	WDP2	WDP1	WDP0
0x20(0x40)	UCSRC	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL
0x18(0x38)	PORTB	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
0x17(0x37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
0x16(0x36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
0x15(0x35)	PORTC	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
0x14(0x37)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
0x13(0x33)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
0x12(0x32)	PORTD	PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0
0x11(0x31)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
0x10(0x30)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
0x0C(0x2C)	UDR	Registre de données USART I/O							
0x0B(0x2B)	UCSRA	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM
0x0A(0x2A)	UCSRB	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8

## Les instructions

On donne sans plus d'explications la liste des instructions.

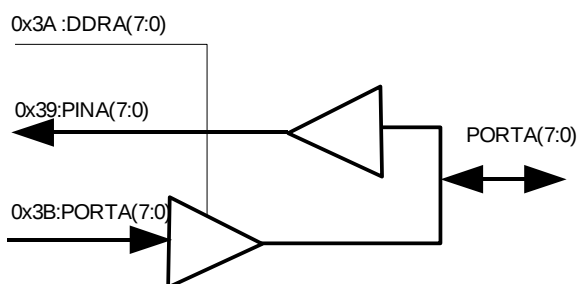
Mnemonics	Operands	Description	Operation	Flags	#Clocks
<b>ARITHMETIC AND LOGIC INSTRUCTIONS</b>					
ADD	Rd, Rr	Add two Registers	$Rd \leftarrow Rd + Rr$	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	$Rd \leftarrow Rd + Rr + C$	Z,C,N,V,H	1
ADIW	Rdl,K	Add Immediate to Word	$Rdh:Rdl \leftarrow Rdh:Rdl + K$	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	$Rd \leftarrow Rd - Rr$	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	$Rd \leftarrow Rd - K$	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	$Rd \leftarrow Rd - Rr - C$	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	$Rd \leftarrow Rd - K - C$	Z,C,N,V,H	1
SBIW	Rdl,K	Subtract Immediate from Word	$Rdh:Rdl \leftarrow Rdh:Rdl - K$	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	$Rd \leftarrow Rd * Rr$	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	$Rd \leftarrow Rd * K$	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	$Rd \leftarrow Rd v Rr$	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	$Rd \leftarrow Rd v K$	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	$Rd \leftarrow Rd \oplus Rr$	Z,N,V	1
COM	Rd	One's Complement	$Rd \leftarrow 0xFF - Rd$	Z,C,N,V	1
NEG	Rd	Two's Complement	$Rd \leftarrow 0x00 - Rd$	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	$Rd \leftarrow Rd v K$	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	$Rd \leftarrow Rd * (0xFF - K)$	Z,N,V	1
INC	Rd	Increment	$Rd \leftarrow Rd + 1$	Z,N,V	1
DEC	Rd	Decrement	$Rd \leftarrow Rd - 1$	Z,N,V	1
TST	Rd	Test for Zero or Minus	$Rd \leftarrow Rd * Rd$	Z,N,V	1
CLR	Rd	Clear Register	$Rd \leftarrow Rd \oplus Rd$	Z,N,V	1
SER	Rd	Set Register	$Rd \leftarrow 0xFF$	None	1
MUL	Rd, Rr	Multiply Unsigned	$R1:R0 \leftarrow Rd * Rr$	Z,C	2
MULS	Rd, Rr	Multiply Signed	$R1:R0 \leftarrow Rd * Rr$	Z,C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	$R1:R0 \leftarrow Rd * Rr$	Z,C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	$R1:R0 \leftarrow (Rd * Rr) \lll 1$	Z,C	2
FMULS	Rd, Rr	Fractional Multiply Signed	$R1:R0 \leftarrow (Rd * Rr) \lll 1$	Z,C	2
FMULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	$R1:R0 \leftarrow (Rd * Rr) \lll 1$	Z,C	2

BRANCH INSTRUCTIONS					
RJMP	k	Relative Jump	$PC \leftarrow PC + k + 1$	None	2
IJMP		Indirect Jump to (Z)	$PC \leftarrow Z$	None	2
RCALL	k	Relative Subroutine Call	$PC \leftarrow PC + k + 1$	None	3
ICALL		Indirect Call to (Z)	$PC \leftarrow Z$ None3		
RET		Subroutine Return	$PC \leftarrow STACK$	None	4
RETI		Interrupt Return	$PC \leftarrow STACK$	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) $PC \leftarrow PC + 2$ or 3	None	1 / 2 / 3
CP	Rd,Rr	Compare	Rd - Rr	Z, N,V,C,H	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z, N,V,C,H	1
CPI	Rd,K	Compare Register with Immediate	Rd - K	Z, N,V,C,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) $PC \leftarrow PC + 2$ or 3	None	1 / 2 / 3
SBRS	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) $PC \leftarrow PC + 2$ or 3	None	1 / 2 / 3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) $PC \leftarrow PC + 2$ or 3	None	1 / 2 / 3
SBSI	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) $PC \leftarrow PC + 2$ or 3	None	1 / 2 / 3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then $PC \leftarrow PC + k + 1$	None	1 / 2
BREQ	k	Branch if Equal	if (Z = 1) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRNE	k	Branch if Not Equal	if (Z = 0) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRCS	k	Branch if Carry Set	if (C = 1) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRCC	k	Branch if Carry Cleared	if (C = 0) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRSH	k	Branch if Same or Higher	if (C = 0) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRLO	k	Branch if Lower	if (C = 1) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRMI	k	Branch if Minus	if (N = 1) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRPL	k	Branch if Plus	if (N = 0) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V = 0) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRLT	k	Branch if Less Than Zero, Signed	if (N ⊕ V = 1) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRTS	k	Branch if T Flag Set	if (T = 1) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then $PC \leftarrow PC + k + 1$	None	1 / 2
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Move Between Registers	$Rd \leftarrow Rr$	None	1
MOVW	Rd, Rr	Copy Register Word	$Rd+1:Rd \leftarrow Rr+1:Rr$	None	1
LDI	Rd, K	Load Immediate	$Rd \leftarrow K$ None1		
LD	Rd, X	Load Indirect	$Rd \leftarrow (X)$	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	$Rd \leftarrow (X), X \leftarrow X + 1$	None	2
LD	Rd, -X	Load Indirect and Pre-Dec.	$X \leftarrow X - 1, Rd \leftarrow (X)$	None	2
LD	Rd, Y	Load Indirect	$Rd \leftarrow (Y)$	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	$Rd \leftarrow (Y), Y \leftarrow Y + 1$	None	2
LD	Rd, -Y	Load Indirect and Pre-Dec.	$Y \leftarrow Y - 1, Rd \leftarrow (Y)$	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	$Rd \leftarrow (Y + q)$	None	2
LD	Rd, Z	Load Indirect	$Rd \leftarrow (Z)$	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	$Rd \leftarrow (Z), Z \leftarrow Z + 1$	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	$Z \leftarrow Z - 1, Rd \leftarrow (Z)$	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	$Rd \leftarrow (Z + q)$	None	2
LDS	Rd, k	Load Direct from SRAM	$Rd \leftarrow (k)$	None	2
ST	X, Rr	Store Indirect	$(X) \leftarrow Rr$	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	$(X) \leftarrow Rr, X \leftarrow X + 1$	None	2
ST	-X, Rr	Store Indirect and Pre-Dec.	$X \leftarrow X - 1, (X) \leftarrow Rr$	None	2
ST	Y, Rr	Store Indirect	$(Y) \leftarrow Rr$	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	$(Y) \leftarrow Rr, Y \leftarrow Y + 1$	None	2
ST	-Y, Rr	Store Indirect and Pre-Dec.	$Y \leftarrow Y - 1, (Y) \leftarrow Rr$	None	2
STD	Y+q,Rr	Store Indirect with Displacement	$(Y + q) \leftarrow Rr$	None	2
ST	Z, Rr	Store Indirect	$(Z) \leftarrow Rr$	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	$(Z) \leftarrow Rr, Z \leftarrow Z + 1$	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	$Z \leftarrow Z - 1, (Z) \leftarrow Rr$	None	2
STD	Z+q,Rr	Store Indirect with Displacement	$(Z + q) \leftarrow Rr$	None	2
STS	k, Rr	Store Direct to SRAM	$(k) \leftarrow Rr$	None	2
LPM		Load Program Memory	$R0 \leftarrow (Z)$	None	3
LPM	Rd, Z	Load Program Memory	$Rd \leftarrow (Z)$	None	3
LPM	Rd, Z+	Load Program Memory and Post-Inc	$Rd \leftarrow (Z), Z \leftarrow Z + 1$	None	3
SPM		Store Program Memory	$(Z) \leftarrow R1:R0$	None	-
IN	Rd, P	In Port	$Rd \leftarrow P$ None1		
OUT	P, Rr	Out Port	$P \leftarrow Rr$	None	1
PUSH	Rr	Push Register on Stack	$STACK \leftarrow Rr$	None	2
POP	Rd	Pop Register from Stack	$Rd \leftarrow STACK$	None	2

BIT AND BIT-TEST INSTRUCTIONS					
SBI	P, b	Set Bit in I/O Register	$I/O(P, b) \leftarrow 1$ None2		
CBI	P, b	Clear Bit in I/O Register	$I/O(P, b) \leftarrow 0$ None2		
LSL	Rd	Logical Shift Left	$Rd(n+1) \leftarrow Rd(n), Rd(0) \leftarrow 0$	Z, C, N, V	1
LSR	Rd	Logical Shift Right	$Rd(n) \leftarrow Rd(n+1), Rd(7) \leftarrow 0$	Z, C, N, V	1
ROL	Rd	Rotate Left Through Carry	$Rd(0) \leftarrow C, Rd(n+1) \leftarrow Rd(n), C \leftarrow Rd(7)$	Z, C, N, V	1
ROR	Rd	Rotate Right Through Carry	$Rd(7) \leftarrow C, Rd(n) \leftarrow Rd(n+1), C \leftarrow Rd(0)$	Z, C, N, V	1
ASR	Rd	Arithmetic Shift Right	$Rd(n) \leftarrow Rd(n+1), n=0..6$	Z, C, N, V	1
SWAP	Rd	Swap Nibbles	$Rd(3..0) \leftarrow Rd(7..4), Rd(7..4) \leftarrow Rd(3..0)$	None	1
BSET	s	Flag Set	$SREG(s) \leftarrow 1$	SREG(s)	1
BCLR	s	Flag Clear	$SREG(s) \leftarrow 0$	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	$T \leftarrow Rr(b)$	T	1
BLD	Rd, b	Bit load from T to Register	$Rd(b) \leftarrow T$ None1		
SEC		Set Carry	$C \leftarrow 1$ C1		
CLC		Clear Carry	$C \leftarrow 0$	C	1
SEN		Set Negative Flag	$N \leftarrow 1$ N1		
CLN		Clear Negative Flag	$N \leftarrow 0$	N	1
SEZ		Set Zero Flag	$Z \leftarrow 1$ Z1		
CLZ		Clear Zero Flag	$Z \leftarrow 0$	Z	1
SEI		Global Interrupt Enable	$I \leftarrow 1$ I1		
CLI		Global Interrupt Disable	$I \leftarrow 0$	I	1
SES		Set Signed Test Flag	$S \leftarrow 1$ S1		
CLS		Clear Signed Test Flag	$S \leftarrow 0$	S	1
SEV		Set Twos Complement Overflow	$V \leftarrow 1$ V1		
CLV		Clear Twos Complement Overflow	$V \leftarrow 0$	V	1
SET		Set T in SREG	$T \leftarrow 1$ T1		
Mnemonics	Operands	Description	Operation	Flags	#Clocks

## Les ports du cœur CoreAtMega8

La gestion des PORTs dans ce cœur ressemble à cette même gestion dans le PicoBlaze. Il est facile d'étendre le nombre de PORTs jusqu'à 64 dans l'espace d'entrées/sorties. Nous nous contenterons cependant de l'existant, à savoir 4 PORTs : **PORTA**, **PORTB**, **PORTC** et **PORTD**. L'architecture est conçue pour une réalisation de PORTs bidirectionnels. Ces PORTs bidirectionnels ne sont pas réalisés dans le cœur car trop dépendants du fabricant du FPGA qui accueillera ce soft processor.



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 souhaitez vous pouvez conserver ces trois ports intacts : c'est ce que l'on fera dans notre projet.

**Remarques** : tous les ports sont accessibles de manière normale. L'écriture dans **DDRA** se fait par l'instruction :

```
// en langage C
DDRA=0xFF; // 1 <=> output
```

tandis qu'en assembleur on écrira :

```
; assembleur
ldi r10,0xFF
out DDRA,r10
```

Ma deuxième remarque sera pour compléter notre description. On doit écrire dans un programme C :

```
PORTA = PINA; // recopie du PORTA dans le PORTA
```

et non pas PORTA=PORTA.

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

On rappelle qu'il s'agit de l'espace à partir de l'adresse 0x60. C'est là que se trouve la mémoire à proprement parler.

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 AASIC Subset Model. Pour la RAM, sa spécificité est qu'il s'agit d'une mémoire à écriture et lecture synchrone. Le cœur CoreAtMega8 propose un exemple de mémoire en VHDL que nous utiliserons sans changement.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity data_mem is
  port ( I_CLK      : in std_logic;

         I_ADR      : in std_logic_vector(10 downto 0);
         I_DIN      : in std_logic_vector(15 downto 0);
         I_WE       : in std_logic_vector( 1 downto 0);

         Q_DOUT     : out std_logic_vector(15 downto 0));
end data_mem;
```

```

architecture Behavioral of data_mem is

    constant zero_256 : bit_vector := X"00000000000000000000000000000000"
        & X"00000000000000000000000000000000";
    constant nine_256 : bit_vector := X"99999999999999999999999999999999"
        & X"99999999999999999999999999999999";

    component RAMB4_S4_S4
        generic(INIT_00 : bit_vector := zero_256;
            INIT_01 : bit_vector := zero_256;
            INIT_02 : bit_vector := zero_256;
            INIT_03 : bit_vector := zero_256;
            INIT_04 : bit_vector := zero_256;
            INIT_05 : bit_vector := zero_256;
            INIT_06 : bit_vector := zero_256;
            INIT_07 : bit_vector := zero_256;
            INIT_08 : bit_vector := zero_256;
            INIT_09 : bit_vector := zero_256;
            INIT_0A : bit_vector := zero_256;
            INIT_0B : bit_vector := zero_256;
            INIT_0C : bit_vector := zero_256;
            INIT_0D : bit_vector := zero_256;
            INIT_0E : bit_vector := zero_256;
            INIT_0F : bit_vector := zero_256);

        port( DOA   : out std_logic_vector(3 downto 0);
            DOB   : out std_logic_vector(3 downto 0);
            ADDR_A : in  std_logic_vector(9 downto 0);
            ADDR_B : in  std_logic_vector(9 downto 0);
            CLKA   : in  std_ulogic;
            CLKB   : in  std_ulogic;
            DIA   : in  std_logic_vector(3 downto 0);
            DIB   : in  std_logic_vector(3 downto 0);
            ENA   : in  std_ulogic;
            ENB   : in  std_ulogic;
            RSTA  : in  std_ulogic;
            RSTB  : in  std_ulogic;
            WEA   : in  std_ulogic;
            WEB   : in  std_ulogic);
    end component;

    signal L_ADR_0   : std_logic;
    signal L_ADR_E   : std_logic_vector(10 downto 1);
    signal L_ADR_O   : std_logic_vector(10 downto 1);
    signal L_DIN_E   : std_logic_vector( 7 downto 0);
    signal L_DIN_O   : std_logic_vector( 7 downto 0);
    signal L_DOUT_E  : std_logic_vector( 7 downto 0);
    signal L_DOUT_O  : std_logic_vector( 7 downto 0);
    signal L_WE_E   : std_logic;
    signal L_WE_O   : std_logic;

begin
    sr_0 : RAMB4_S4_S4 -----
        generic map(INIT_00 => nine_256, INIT_01 => nine_256, INIT_02 => nine_256,
            INIT_03 => nine_256, INIT_04 => nine_256, INIT_05 => nine_256,
            INIT_06 => nine_256, INIT_07 => nine_256, INIT_08 => nine_256,
            INIT_09 => nine_256, INIT_0A => nine_256, INIT_0B => nine_256,

```

```

INIT_0C => nine_256, INIT_0D => nine_256, INIT_0E => nine_256,
INIT_0F => nine_256)

port map(  ADDRA => L_ADR_E,          ADDR_B => "0000000000",
          CLKA => I_CLK,            CLKB => I_CLK,
          DIA => L_DIN_E(3 downto 0), DIB => "0000",
          ENA => '1',              ENB => '0',
          RSTA => '0',             RSTB => '0',
          WEA => L_WE_E,           WEB => '0',
          DOA => L_DOUT_E(3 downto 0), DOB => open);

-- ***** une partie est retirée : voir le fichier original : data_mem.vhd *****

sr_3 : RAMB4_S4_S4 -----
generic map(INIT_00 => nine_256, INIT_01 => nine_256, INIT_02 => nine_256,
          INIT_03 => nine_256, INIT_04 => nine_256, INIT_05 => nine_256,
          INIT_06 => nine_256, INIT_07 => nine_256, INIT_08 => nine_256,
          INIT_09 => nine_256, INIT_0A => nine_256, INIT_0B => nine_256,
          INIT_0C => nine_256, INIT_0D => nine_256, INIT_0E => nine_256,
          INIT_0F => nine_256)

port map(  ADDRA => L_ADR_O,          ADDR_B => "0000000000",
          CLKA => I_CLK,            CLKB => I_CLK,
          DIA => L_DIN_O(7 downto 4), DIB => "0000",
          ENA => '1',              ENB => '0',
          RSTA => '0',             RSTB => '0',
          WEA => L_WE_O,           WEB => '0',
          DOA => L_DOUT_O(7 downto 4), DOB => open);
-- remember ADR(0)
--
adr0: process(I_CLK)
begin
  if (rising_edge(I_CLK)) then
    L_ADR_0 <= I_ADR(0);
  end if;
end process;
-- we use two memory blocks _E and _O (even and odd).
-- This gives us a memory with ADR and ADR + 1 at th same time.
-- The second port is currently unused, but may be used later,
-- e.g. for DMA.
--
L_ADR_O <= I_ADR(10 downto 1);
L_ADR_E <= I_ADR(10 downto 1) + ("000000000" & I_ADR(0));

L_DIN_E <= I_DIN( 7 downto 0) when (I_ADR(0) = '0') else I_DIN(15 downto 8);
L_DIN_O <= I_DIN( 7 downto 0) when (I_ADR(0) = '1') else I_DIN(15 downto 8);

L_WE_E <= I_WE(1) or (I_WE(0) and not I_ADR(0));
L_WE_O <= I_WE(1) or (I_WE(0) and I_ADR(0));

Q_DOUT( 7 downto 0) <= L_DOUT_E when (L_ADR_0 = '0') else L_DOUT_O;
Q_DOUT(15 downto 8) <= L_DOUT_E when (L_ADR_0 = '1') else L_DOUT_O;

end Behavioral;

```

La mémoire est construite à partir de composants Xilinx "RAMB4\_S4\_S4" qui est une

mémoire double ports. Cette façon de faire est peu portable et il vous faudra l'adapter si vous désirez faire tourner ce soft processor dans un FPGA concurrent.

## La ROM

C'est là que se situe le programme à exécuter. Commençons à présenter un exemple de ROM.

```
-- VHDL File : prog_mem.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- the content of the program memory.
--
use work.prog_mem_content.all;

entity prog_mem is
  port ( I_CLK      : in std_logic;

        I_WAIT     : in std_logic;
        I_PC       : in std_logic_vector(15 downto 0); -- word address
        I_PM_ADR   : in std_logic_vector(11 downto 0); -- byte address

        Q_OPC      : out std_logic_vector(31 downto 0);
        Q_PC       : out std_logic_vector(15 downto 0);
        Q_PM_DOUT  : out std_logic_vector( 7 downto 0));
end prog_mem;

architecture Behavioral of prog_mem is

  -- ***** retiré pour question de place *****

end Behavioral;
```

Cet exemple incomplet ne montre aucun contenu. 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 s'appelle "make\_mem.cc" qui est fourni avec le cœur. Compilez donc ce programme pour en faire un exécutable. Une fois l'exécutable réalisé, lancer

```
make_mem demo.hex prog_mem_content.vhd
```

ce qui vous génèrera un package dans un fichier prog\_mem\_content.vhd, qui sera utilisé dans le projet. La mémoire est construite à partir de composants Xilinx "RAMB4\_S4\_S4" qui est une mémoire double ports. Cette façon de faire est peu portable et il vous faudra l'adapter si vous désirez faire tourner ce soft processor dans un FPGA Altera ou ACTEL.

## Mes premiers programmes en C

La programmation du cœur se fait dans l'environnement AVRStudio. Le téléchargement d'AVRStudio nous permet d'utiliser l'assembleur ainsi que le compilateur C GNU (si l'on a installé winAVR). Puisque nous avons décidé pour ce projet de n'utiliser que des outils gratuits, cela nous convient parfaitement.

Mon premier programme en C a été réalisé pour tester le fonctionnement du cœur **CoreAtMega8** sur notre carte d'application Digilent.

Notre architecture **CoreAtMega8** 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 <avr/io.h>
main(void)
{
asm volatile (
debut:  in    r24, 0x16      ; 22 : PINB
        com r24             ; one's complement
        out   0x18, r24     ; 24 : PORTB
        rjmp  debut        ; infinite loop
)
}
```

Ne prenez pas ce programme à la lettre, il est juste donné comme exemple **et n'a pas été testé**.

### *Le premier programme simple en C pur*

Un programme tout simple de test est présenté maintenant : il s'agit tout simplement de recopier le PORTB 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 <avr/io.h>
main(void)
{ // La gestion de DDRB et DDRC est inutile pour ce cœur
  DDRB = 0xFF; // 8 sorties pour B
  DDRC = 0x00; // 8 entrees pour C
  while(1)
  PORTB = PINB; // recopie du PORTB dans le PORTB qui allume les LEDs
}
```

Contrairement au Silicore1657 déjà évoqué (projet de l'année précédente), le cœur **CoreAtMega8** dispose du mécanisme d'interruption que nous allons détailler maintenant.

## Ajoutons une interruption

Cette section devra certainement être déplacée et en tout cas éteffée : la gestion des interruptions semble être prévue dans le cœur, mais aucun mécanisme ne peut la déclencher. C'est à vous de modifier le cœur **CoreAtMega8** pour les gérer (cherchez dans le fichier "io.vhd" et lisez "avr/interrupt.h").

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).

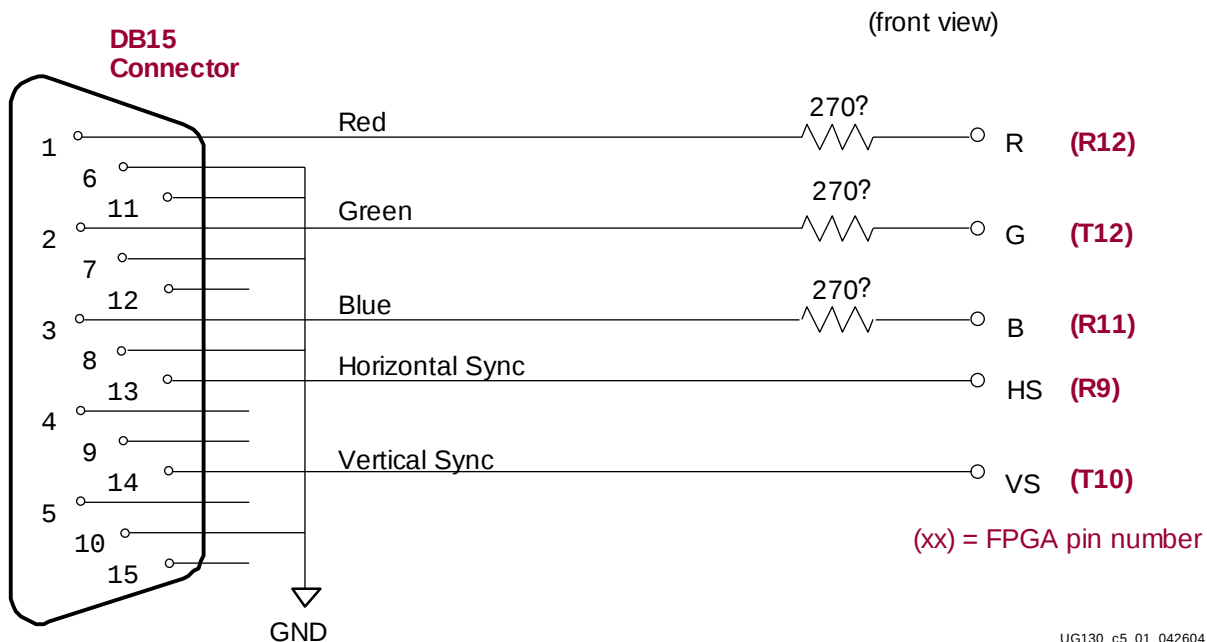
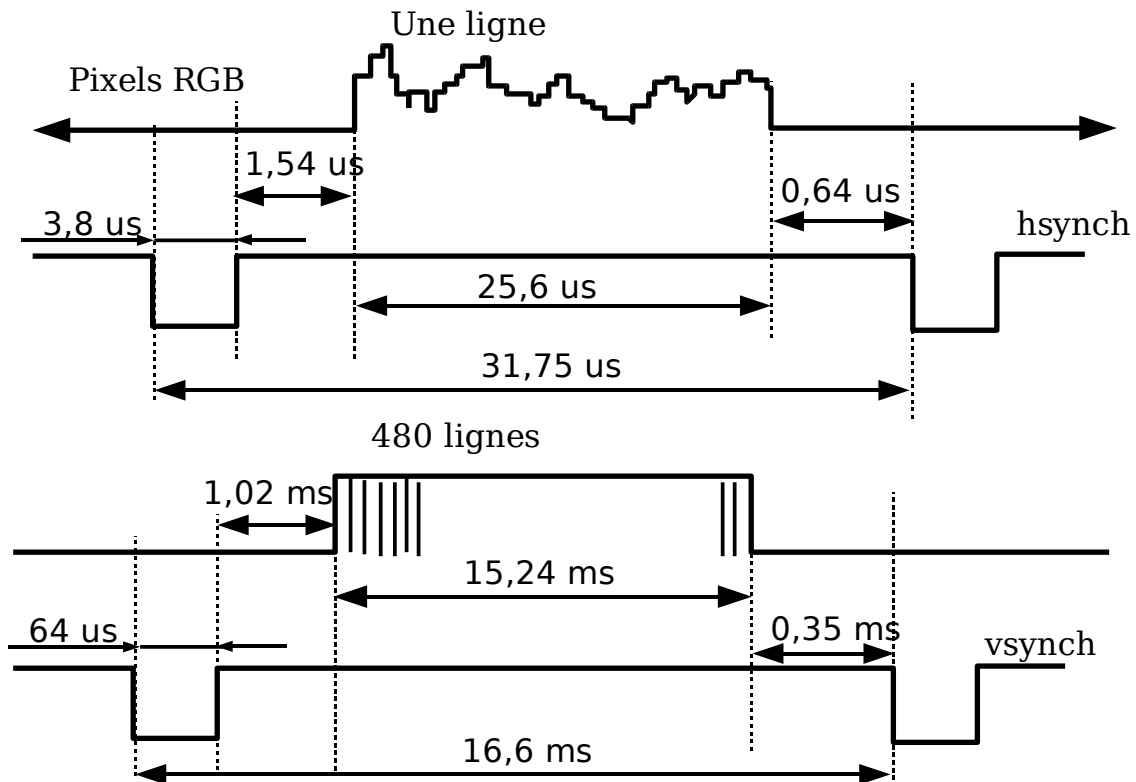


Figure 5-1: VGA Connections from Spartan-3 Starter Kit Board

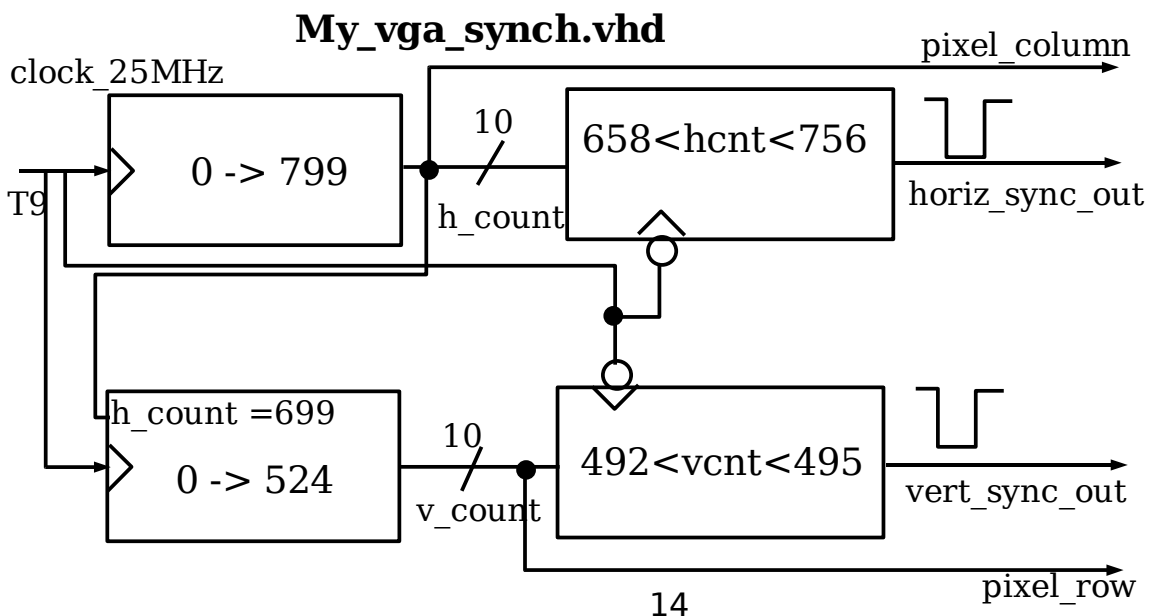
## Les signaux à construire

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



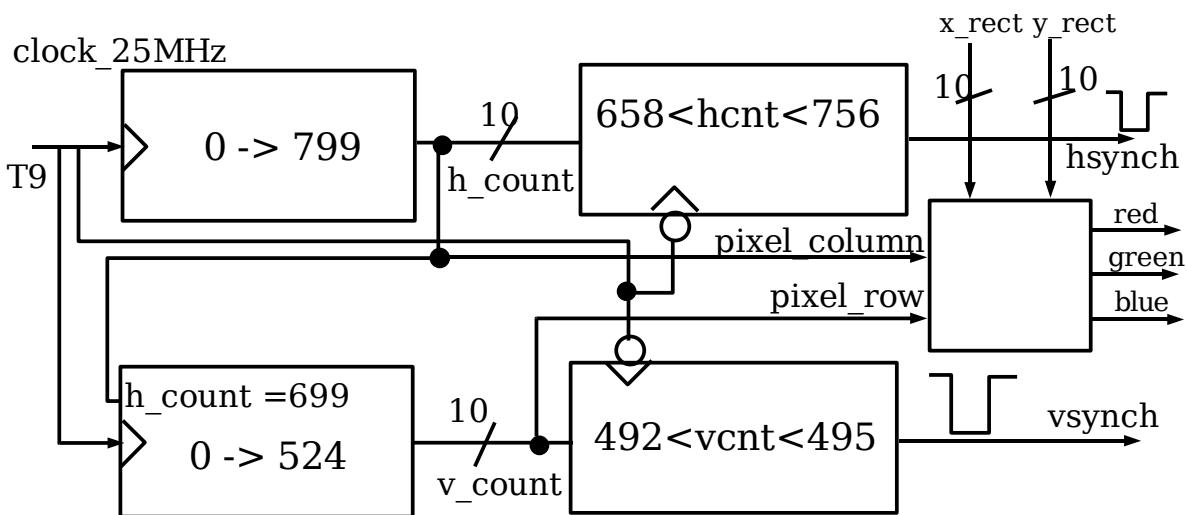
l'architecture ci-dessous qui représente partiellement le contenu du fichier My\_vga\_synch.vhd :



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 (qui deviendra une balle)**

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 pour notre cœur rien que pour gérer les positions X et Y d'une balle sur l'écran.

### **La gestion des PORTs dans le CoreAtMega8**

La gestion des ports du CoreAtMega8 est relativement simple. Il suffit de modifier le fichier `io.vhd` du projet initial. J'ai choisi d'utiliser les PORTs existants (je veux dire connus du compilateur C) pour réaliser mon projet. J'ai choisi aussi de gérer les raquettes sur 8 bits seulement, ce qui me fait en tout 6 PORTs à trouver.

Le fichier VHDL qui fait la gestion s'appelle io2.vhd et remplace le fichier io.vhd du projet initial. Nous en donnons le contenu maintenant. (On a laissé le composant qui gère la communication RS232 du cœur initial, on a ajouté le composant VGA\_Top et modifié le "IO Write" process)

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity io is
  port ( I_CLK      : in  std_logic;
        --> added for VGA synchro 18 June 2010
        I_CLK_50   : in  std_logic;
        --< end added
        I_CLR      : in  std_logic;
        I_ADR_IO   : in  std_logic_vector( 7 downto 0 );
        I_DIN      : in  std_logic_vector( 7 downto 0 );
        I_SWITCH   : in  std_logic_vector( 7 downto 0 );
        I_RD_IO    : in  std_logic;
        I_RX       : in  std_logic;
        I_WE_IO    : in  std_logic;
        Q_7_SEGMENT : out std_logic_vector( 6 downto 0 );
        Q_DOUT     : out std_logic_vector( 7 downto 0 );
        Q_INTVEC   : out std_logic_vector( 5 downto 0 );
        Q_LEDS     : out std_logic_vector( 1 downto 0 );
        Q_TX       : out std_logic;
        hsynch,vsynch,red,green,blue : out STD_LOGIC);
end io;

architecture Behavioral of io is
  component uart
    generic(CLOCK_FREQ : std_logic_vector(31 downto 0);
           BAUD_RATE   : std_logic_vector(27 downto 0));
    port( I_CLK      : in  std_logic;
          I_CLR      : in  std_logic;
          I_RD       : in  std_logic;
          I_WE       : in  std_logic;
          I_RX       : in  std_logic;
          I_TX_DATA  : in  std_logic_vector(7 downto 0);

          Q_RX_DATA  : out std_logic_vector(7 downto 0);
          Q_RX_READY : out std_logic;
          Q_TX       : out std_logic;
          Q_TX_BUSY  : out std_logic);
  end component;

  signal U_RX_READY : std_logic;
  signal U_TX_BUSY  : std_logic;
  signal U_RX_DATA  : std_logic_vector( 7 downto 0 );

  signal L_INTVEC   : std_logic_vector( 5 downto 0 );
  signal L_LEDS     : std_logic;
  signal L_RD_UART  : std_logic;
  signal L_RX_INT_ENABLED : std_logic;
  signal L_TX_INT_ENABLED : std_logic;
  signal L_WE_UART  : std_logic;

  component VGATop
    PORT (clk_50 : in STD_LOGIC;
          x_rect, y_rect: IN STD_LOGIC_VECTOR(9 DOWNT0 0);
          y_raquG, y_raquD: IN STD_LOGIC_VECTOR(7 DOWNT0 0);
          hsynch,vsynch,red,green,blue : out STD_LOGIC);
  END component;

  signal Balle_xLow : STD_LOGIC_VECTOR(7 DOWNT0 0);
  signal Balle_xHigh : STD_LOGIC_VECTOR(7 DOWNT0 0);
  signal Balle_yLow : STD_LOGIC_VECTOR(7 DOWNT0 0);
  signal Balle_yHigh : STD_LOGIC_VECTOR(7 DOWNT0 0);
  signal raqD_y : STD_LOGIC_VECTOR(7 DOWNT0 0);

```

```

signal raqG_y : STD_LOGIC_VECTOR(7 DOWNTO 0);

begin
  urt: uart
    generic map(CLOCK_FREQ => std_logic_vector(conv_unsigned(25000000, 32)),
               BAUD_RATE   => std_logic_vector(conv_unsigned( 38400, 28)))
    port map(
      I_CLK   => I_CLK,
      I_CLR   => I_CLR,
      I_RD    => L_RD_UART,
      I_WE    => L_WE_UART,
      I_TX_DATA => I_DIN(7 downto 0),
      I_RX    => I_RX,

      Q_TX    => Q_TX,
      Q_RX_DATA => U_RX_DATA,
      Q_RX_READY => U_RX_READY,
      Q_TX_BUSY => U_TX_BUSY);
  vga:VGATop
    port map (clk_50 => I_CLK_50,
              x_rect(7 downto 0) => Balle_xLow,
              x_rect(9 downto 8) => Balle_xHigh(1 downto 0),
              y_rect(7 downto 0) => Balle_yLow,
              y_rect(9 downto 8) => Balle_yHigh(1 downto 0),
              y_raqG => raqG_y,
              y_raqD => raqD_y,
              hsynch => hsynch,
              vsynch => vsynch,
              red => red,
              green => green,
              blue => blue);
  -- I0 read process
  --
  iord: process(I_ADR_I0, I_SWITCH,
               U_RX_DATA, U_RX_READY, L_RX_INT_ENABLED,
               U_TX_BUSY, L_TX_INT_ENABLED)
  begin
    -- addresses for mega8 device (use iom8.h or #define __AVR_ATmega8__).
    --
    case I_ADR_I0 is
      when X"2A" => Q_DOUT <=
        L_RX_INT_ENABLED -- UCSRB:
          & L_TX_INT_ENABLED -- Rx complete int enabled.
          & L_TX_INT_ENABLED -- Tx complete int enabled.
          & '1' -- Tx empty int enabled.
          & '1' -- Rx enabled
          & '1' -- Tx enabled
          & '0' -- 8 bits/char
          & '0' -- Rx bit 8
          & '0'; -- Tx bit 8
      when X"2B" => Q_DOUT <=
        U_RX_READY -- UCSRA:
          & not U_TX_BUSY -- Rx complete
          & not U_TX_BUSY -- Tx complete
          & not U_TX_BUSY -- Tx ready
          & '0' -- frame error
          & '0' -- data overrun
          & '0' -- parity error
          & '0' -- double dpeed
          & '0'; -- multiproc mode
      when X"2C" => Q_DOUT <= U_RX_DATA; -- UDR
      when X"40" => Q_DOUT <=
        '1' -- UCSRC
          & '0' -- URSEL
          & "00" -- asynchronous
          & '1' -- no parity
          & "11" -- two stop bits
          & "11" -- 8 bits/char
          & '0'; -- rising clock edge

      when X"36" => Q_DOUT <= I_SWITCH; -- PINB
      when others => Q_DOUT <= X"AA";
    end case;
  end process;
  -- I0 write process
  --

```

```

iowr: process(I_CLK)
begin
  if (rising_edge(I_CLK)) then
    if (I_CLR = '1') then
      L_RX_INT_ENABLED <= '0';
      L_TX_INT_ENABLED <= '0';
    elsif (I_WE_IO = '1') then
      case I_ADR_IO is
        when X"31" => Balle_xLow <= I_DIN; --DDRD
        when X"32" => Balle_xHigh <= I_DIN; --PORTD
        when X"37" => Balle_yLow <= I_DIN; --DDRDB
        when X"38" => Balle_yHigh <= I_DIN; --PORTB
        when X"34" => raqD_y <= I_DIN; --DDRC
        when X"35" => raqG_y <= I_DIN; --PORTC
        when X"40" => -- handled by uart
        when X"41" => -- handled by uart
        when X"43" => L_RX_INT_ENABLED <= I_DIN(0);
                    L_TX_INT_ENABLED <= I_DIN(1);
        when others =>
          end case;
      end if;
    end if;
  end process;

  -- interrupt process
  --
  ioint: process(I_CLK)
  begin
    if (rising_edge(I_CLK)) then
      if (I_CLR = '1') then
        L_INTVEC <= "000000";
      else
        if (L_RX_INT_ENABLED and U_RX_READY) = '1' then
          if (L_INTVEC(5) = '0') then -- no interrupt pending
            L_INTVEC <= "101011"; -- _VECTOR(11)
          end if;
        elsif (L_TX_INT_ENABLED and not U_TX_BUSY) = '1' then
          if (L_INTVEC(5) = '0') then -- no interrupt pending
            L_INTVEC <= "101100"; -- _VECTOR(12)
          end if;
        else
          L_INTVEC <= "000000"; -- no interrupt
        end if;
      end if;
    end if;
  end process;

  L_WE_UART <= I_WE_IO when (I_ADR_IO = X"2C") else '0'; -- write UART UDR
  L_RD_UART <= I_RD_IO when (I_ADR_IO = X"2C") else '0'; -- read UART UDR

  Q_LEDS(1) <= L_LEDS;
  Q_LEDS(0) <= not L_LEDS;
  Q_INTVEC <= L_INTVEC;

end Behavioral;

```

La lecture de ce programme VHDL nous montre immédiatement les choix technologiques effectués pour la connexion des PORTs aux coordonnées des balles et raquettes.

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

raquette gauche	y_raqG<7:0>	PORTC<1:0>
raquette droite	y_raqD<7:0>	DDRC<7:0>

Nous sommes prêt pour la programmation en C de notre ensemble cœur plus gestion écran VGA.

## Programmation du nouveau cœur avec écran VGA

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 pour changer la position en X de la balle.

#### Le sous-programme "setX"

Une version en C pur est montrée ci-dessous :

```
void setX(uint16_t x){
    DDRD=x; //poids faible
    PORTD=x>>8;//poids fort
}
```

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

#### Recopie des interrupteurs pour déplacer la balle

Nous allons présenter maintenant un programme fonctionnel qui déplace la balle dans une position déterminée par les interrupteurs.

```
#include <avr/io.h>
void setX(uint16_t x);
void setY(unsigned int x);
unsigned int posRaqu_16;

void main (void)
{
    unsigned int posX,posY;
    unsigned char raqD_y=0,raqG_y=0;
    signed char deltaX=1,deltaY=1;
    posX=113;
    posY=101;
    setX(posX);
    setY(posY);
}
```

```

        while(1){
            //setY directement
            DDRB=PINB;
            setX(PINB);
        }

void setX(uint16_t x){
    DDRD=x; //poids faible
    PORTD=x>>8;//poids fort
}

void setY(unsigned int y){
    DDRB=y; //poids faible
    PORTB=y>>8;//poids fort
}

```

Remarquez qu'au lieu d'utiliser le sous-programme setY on a réalisé l'affectation directe dans DDRB. Remarquez aussi les deux types distincts pour travailler sur 16 bits: `uint16_t` et `unsigned int`.

## Programme complet de gestion simple de la balle

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

```

#include <avr/io.h>
#include <util/delay.h>
void setX(uint16_t x);
void setY(unsigned int x);
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);
        _delay_loop_2(30000);
    }
}

void setX(uint16_t x){
    DDRD=x; //poids faible
    PORTD=x>>8;//poids fort
}

void setY(unsigned int y){
    DDRB=y; //poids faible
    PORTB=y>>8;//poids fort
}

```

Ce programme réalise une balle rebondissant sur les bords. Le petit nombre des trajectoires gérées, peut devenir lassant pour un jeu. Mais à ce stade personne ne peut jouer car les raquettes sont invisibles.

## Ajouter les bords, et rendre les raquettes mobiles

Nous avons déjà eu l'occasion de présenter les choix technologiques réalisés pour la connexion des deux raquettes. Le hardware doit permettre de voir les raquettes, ce sera le logiciel qui les fera bouger.

### Solution simple sans bord

Nous allons présenter un ensemble fonctionnant mais avec des positions de raquette fixes. Ce sera aux étudiants de les faire bouger.

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

```
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;
```

Les déclarations des signaux dans ce morceau de programme sont omises.

Voici maintenant le programme C permettant le rebond sur les raquettes. Rappelons que les raquettes sont fixes, il vous faudra modifier ce programme pour les faire bouger. Le matériel est lui prévu pour les faire bouger comme on l'a vu dans la section "La gestion des PORTs dans le CoreAtMega8".

```

#include <avr/io.h>
#include "util/delay.h"
/* Port A */
#define PINA _SFR_I08(0x19)
#define DDRA _SFR_I08(0x1A)
#define PORTA _SFR_I08(0x1B)
void setX(uint16_t x);
void setY(unsigned int x);
//void wait(unsigned char tempo);
//void wait(int tempo);
unsigned int posRaqu_16;

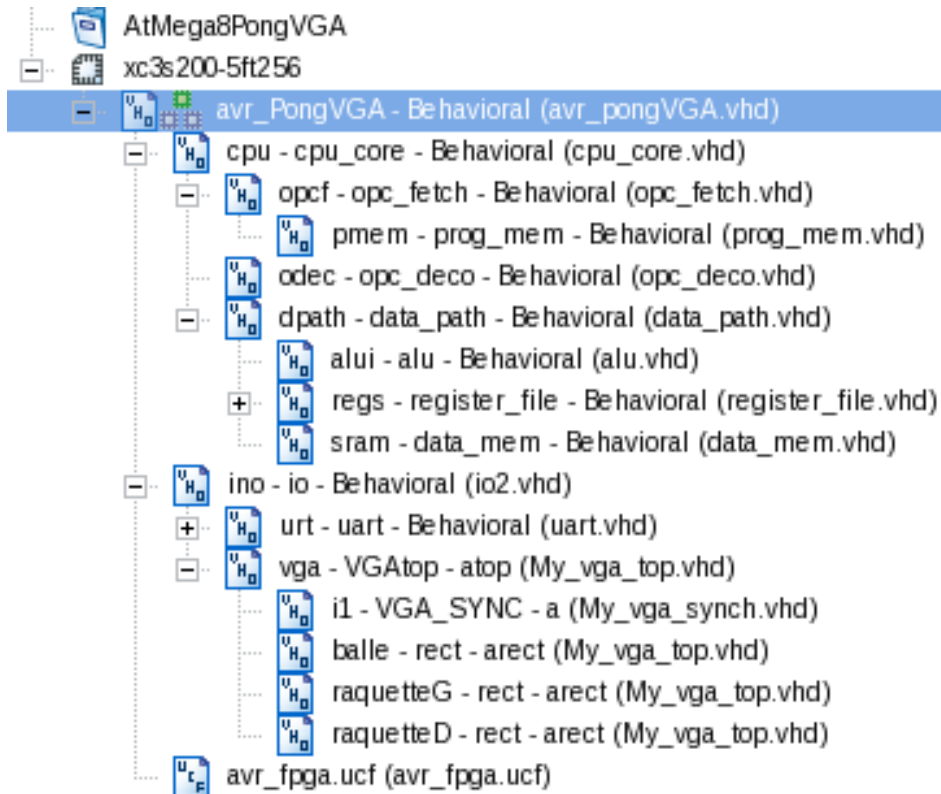
void main (void)
{
    unsigned int posX,posY;
    unsigned char raqD_y=0,raqG_y=0;
    signed char deltaX=1,deltaY=1;
    while(1) {
        posX=130;
        posY=301;
        setX(posX);
        setY(posY);
        while( (posX>30) && (posX<580)){
            posRaqu_16=raqD_y<<1;
            if ((posX>=574) && (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);
            _delay_loop_2(30000);
        }
    }
}

void setX(uint16_t x){
    DDRD=x; //poids faible
    PORTD=x>>8;//poids fort
}

void setY(unsigned int x){
    DDRB=x; //poids faible
    PORTB=x>>8;//poids fort
}

```

Voici la hierarchie des fichiers VHDL dans notre projet à ce point :



Il est facile de construire ce projet, vous avez tous les fichiers nécessaires.

Téléchargement : [http://moutou.pagesperso-orange.fr/ER2/ATMega8\\_pong\\_VGA.zip](http://moutou.pagesperso-orange.fr/ER2/ATMega8_pong_VGA.zip)

## Travail à réaliser

### ***Comprendre la partie matérielle (projet tutoré)***

Vous devez être capable de dessiner les composants et leurs liaisons à partir des fichiers VHDL donnés dans le projet initial.

### ***Traduire et comprendre (projet tutoré)***

Vous allez traduire en français le chapitre 8 sur les Entrées/Sorties du cours (lecture.pdf) et me refaire le tableau des instructions.

### ***Développer en C***

Étendre la partie logicielle pour gérer le déplacement des deux raquettes.

### **Tests de bits des PORTs (pour le déplacement des raquettes)**

Chaque port est défini dans le fichier "avr/io.h". Par exemple PORTB est défini comme `#define PORTB _SFR_I08(0x18)` car il est en position  $0x18=24$  dans

l'espace d'entrées/sorties (soit en adresse 0x38). Chacun des bits des PORTs est aussi prédéfini :

```

/* PORTB */
#define PB7 7
#define PB6 6
#define PB5 5
#define PB4 4
#define PB3 3
#define PB2 2
#define PB1 1
#define PB0 0

/* DDRB */
#define DDB7 7
#define DDB6 6
#define DDB5 5
#define DDB4 4
#define DDB3 3
#define DDB2 2
#define DDB1 1
#define DDB0 0

/* PINB */
#define PINB7 7
#define PINB6 6
#define PINB5 5
#define PINB4 4
#define PINB3 3
#define PINB2 2
#define PINB1 1
#define PINB0 0

```

Muni de ces informations, vous pouvez comprendre que pour tester que le bit PINB1 du port **PINB** est à un il suffit d'écrire en C :

```

/* Port B */
if ((PINB &(1<<PINB1))==(1<<PINB1))
/* ou mieux encore */
if (bit_is_set(PINB, PINB1))

```

Nous avons choisi le PINB connecté aux interrupteurs sw0,sw1,sw6 et sw7 de la carte, pour faire descendre et monter les raquettes mais si vous disposez de joysticks, tout autre choix peut être fait.

### Ajouter un PORT dans notre coeur (inutile si raquettes sur 8 bits)

Par défaut le **PORTA** n'existe pas dans l'ATmega8. Mais pour nous ce n'est pas un problème pour l'ajouter matériellement (dans le fichier io2.vhd) et logiciellement. Il nous suffit d'ajouter :

```

/* Port A */
#define PINA      _SFR_I08(0x19)
#define DDRA     _SFR_I08(0x1A)
#define PORTA    _SFR_I08(0x1B)

```

dans nos programmes en C. Notre fichier "io2.vhd" (voir la section La gestion des PORTs dans le CoreAtMega8) doit être modifié en conséquence (en utilisant les adresses ci-dessus auxquelles on ajoute 0x20 :

```

--***** extraits de io2.vhd *****
case I_ADR_IO is
  when X"3A" => Balle_xLow <= I_DIN; --DDRA
  when X"3B" => Balle_xHigh <= I_DIN; --PORTA
  when X"31" => Balle_xLow <= I_DIN; --DDR0
  when X"32" => Balle_xHigh <= I_DIN; --PORTD
  when X"37" => Balle_yLow <= I_DIN; --DDR1
  when X"38" => Balle_yHigh <= I_DIN; --PORTB
  when X"34" => raqD_y <= I_DIN; --DDRC
  when X"35" => raqG_y <= I_DIN; --PORTC

```

## Tracer de Bresenham

Explorer s'il n'est pas possible d'utiliser l'algorithme de tracé de segment de droite de 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](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.

## Gestion des scores

Gérer un affichage des scores dans le moniteur RS232. Chaque fois qu'un nouveau score est réalisé, il est envoyé sur la liaison série qui est affichée à l'aide d'un hyperterminal. C'est pour répondre à cette question que l'on a laissé la partie concernant la RS232 dans le cœur initial.

## 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 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.

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 amenés à le modifier en partie pour ajouter la gestion des PORTs.

## **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 Starter Board 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 est présenté ci-dessous.

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

L'utilisation est faite par la ligne de commande :

```
make_mem demo1.hex prog_mem_content.vhd
```

si le fichier à convertir s'appelle demo1.hex.

Le fichier généré s'appelle prog\_mem\_content.vhd et se trouve sous la forme d'un package.

```
#include "assert.h"
#include "stdio.h"
#include "stdint.h"
#include "string.h"

uint8_t buffer[0x10000]; // 64 k is max. for Intel hex.
uint8_t slice [0x10000]; // 16 k is max. for Xilinx bram

//-----
//
// get a byte (from cp pointing into Intel hex file).
//
uint32_t
get_byte(const char * cp)
{
    uint32_t value;
    const char cc[3] = { cp[0], cp[1], 0 };
    const int cnt = sscanf(cc, "%X", &value);
    assert(cnt == 1);
    return value;
}
//-----
//
// read an Intel hex file into buffer
void
read_file(FILE * in)
{
    memset(buffer, 0xFF, sizeof(buffer));
    char line[200];
    for (;;)
    {
        const char * s = fgets(line, sizeof(line) - 2, in);
        if (s == 0) return;
        assert(*s++ == ':');
        const uint32_t len    = get_byte(s);
        const uint32_t ah    = get_byte(s + 2);
        const uint32_t al    = get_byte(s + 4);
        const uint32_t rectype = get_byte(s + 6);
        const char * d = s + 8;
        const uint32_t addr = ah << 8 | al;

        uint32_t csum = len + ah + al + rectype;
        assert((addr + len) <= 0x10000);
        for (uint32_t l = 0; l < len; ++l)
        {
            const uint32_t byte = get_byte(d);
```



```

{
    fprintf(out, "constant p%u_%2.2X : BIT_VECTOR := X\"", mem, vec);
    for (uint32_t d = 31; d >= 0; --d)
        fprintf(out, "%2.2X", data[d]);

    fprintf(out, "\";\r\n");
}
//-----
//
// write one memory
//
void
write_mem(FILE * out, uint32_t mem, uint32_t bytes)
{
    fprintf(out, "-- content of p%u -----"
            "-----\r\n", mem);

    const uint8_t * src = slice;
    for (uint32_t v = 0; v < bytes/32; ++v)
        write_vector(out, mem, v, src + 32*v);

    fprintf(out, "\r\n");
}
//-----
//
// write the entire memory_contents file.
//
void
write_file(FILE * out, uint32_t bits)
{
    fprintf(out,
"\r\n"
"library IEEE;\r\n"
"use IEEE.STD_LOGIC_1164.all;\r\n"
"\r\n"
"package prog_mem_content is\r\n"
"\r\n");

    const uint32_t mems = 16/bits;

    for (uint32_t m = 0; m < 2*mems; ++m)
    {
        copy_slice(m, bits, 0x1000);
        write_mem(out, m, 0x200);
    }

    fprintf(out,
"end prog_mem_content;\r\n"
"\r\n");
}
//-----
int
main(int argc, char * argv[])
{
    uint32_t bits = 4;
    const char * prog = *argv++; --argc;

    if (argc && !strcmp(*argv, "-1")) { bits = 1; ++argv; --argc; }
    else if (argc && !strcmp(*argv, "-2")) { bits = 2; ++argv; --argc; }
    else if (argc && !strcmp(*argv, "-4")) { bits = 4; ++argv; --argc; }
    else if (argc && !strcmp(*argv, "-8")) { bits = 8; ++argv; --argc; }
    else if (argc && !strcmp(*argv, "-16")) { bits = 16; ++argv; --argc; }

    const char * hex_file = 0;
    const char * vhd1_file = 0;

    if (argc) { hex_file = *argv++; --argc; }
    if (argc) { vhd1_file = *argv++; --argc; }
    assert(argc == 0);

    FILE * in = stdin;
    if (hex_file) in = fopen(hex_file, "r");
    assert(in);
}

```

```
read_file(in);
fclose(in);

FILE * out = stdout;
if (vhdl_file) out = fopen(vhdl_file, "w");
write_file(out, bits);
assert(out);
}
//-----
```

## ANNEXE II (gestion du VGA)

Nous présentons le programme complet capable de gérer une synchronisation VGA avec une balle et deux raquettes.

```
-- ***** 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;
```

```

        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=>"000001010",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 I_CLK_50 PERIOD = 20 ns;
NET L_CLK PERIOD = 35 ns;

NET I_CLK_50 TNM_NET = I_CLK_100;
NET L_CLK TNM_NET = L_CLK;

NET "I_CLK_50" LOC = "T9";
#NET I_RX LOC = M3;
#NET Q_TX LOC = M4;

# 7 segment LED display
#
#NET Q_7_SEGMENT<0> LOC = "E14";
#NET Q_7_SEGMENT<1> LOC = "G13";
#NET Q_7_SEGMENT<2> LOC = "N15";
#NET Q_7_SEGMENT<3> LOC = "P15";
#NET Q_7_SEGMENT<4> LOC = "R16";
#NET Q_7_SEGMENT<5> LOC = "F13";
#NET Q_7_SEGMENT<6> LOC = "N16";

# single LEDs
#
NET Q_LEDS<0> LOC = "K12";
NET Q_LEDS<1> LOC = "P14";
NET Q_LEDS<2> LOC = "L12";
NET Q_LEDS<3> LOC = "N14";

# DIP switch(0 ... 7) and two pushbuttons (8, 9)
#
NET I_SWITCH<0> LOC = "f12";
NET I_SWITCH<1> LOC = "g12";
NET I_SWITCH<2> LOC = "h14";
NET I_SWITCH<3> LOC = "h13";
NET I_SWITCH<4> LOC = "j14";
NET I_SWITCH<5> LOC = "j13";
NET I_SWITCH<6> LOC = "k14";
NET I_SWITCH<7> LOC = "k13";

NET I_Reset<0> LOC = "L13";
NET I_Reset<1> LOC = "L14";

#NET I_SWITCH<*> PULLUP;

#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 qui décrit donc l'assemblage des composants :

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity avr_PongVGA is
  port (
    I_CLK_50      : in  std_logic;
    I_SWITCH      : in  std_logic_vector(7 downto 0);
    I_RX          : in  std_logic;
    I_Reset       : in  std_logic_vector(1 downto 0);
    Q_LEDS        : out std_logic_vector(3 downto 0);
    Q_TX          : out std_logic;
    hsynch, vsynch, red, green, blue : out STD_LOGIC
  );
end avr_PongVGA;

architecture Behavioral of avr_PongVGA is

  component cpu_core
    port (
      I_CLK      : in  std_logic;
      I_CLR      : in  std_logic;
      I_INTVEC   : in  std_logic_vector( 5 downto 0);
      I_DIN      : in  std_logic_vector( 7 downto 0);

      Q_OPC      : out std_logic_vector(15 downto 0);
      Q_PC       : out std_logic_vector(15 downto 0);
      Q_DOUT     : out std_logic_vector( 7 downto 0);
      Q_ADR_IO   : out std_logic_vector( 7 downto 0);
      Q_RD_IO    : out std_logic;
      Q_WE_IO    : out std_logic);
  end component;

  signal C_PC      : std_logic_vector(15 downto 0);
  signal C_OPC     : std_logic_vector(15 downto 0);
  signal C_ADR_IO  : std_logic_vector( 7 downto 0);
  signal C_DOUT    : std_logic_vector( 7 downto 0);
  signal C_RD_IO   : std_logic;
  signal C_WE_IO   : std_logic;

  component io
    port (
      I_CLK      : in  std_logic;
      --> added for VGA synchro 18th June 2010
      I_CLK_50   : in  std_logic;
      --< end added
      I_CLR      : in  std_logic;
      I_ADR_IO   : in  std_logic_vector( 7 downto 0);
      I_DIN      : in  std_logic_vector( 7 downto 0);
      I_RD_IO    : in  std_logic;
      I_WE_IO    : in  std_logic;
      I_SWITCH   : in  std_logic_vector( 7 downto 0);
      I_RX       : in  std_logic;

      Q_7_SEGMENT : out std_logic_vector( 6 downto 0);
      Q_DOUT      : out std_logic_vector( 7 downto 0);
      Q_INTVEC    : out std_logic_vector(5 downto 0);
      Q_LEDS      : out std_logic_vector( 1 downto 0);
      Q_TX        : out std_logic;
      hsynch, vsynch, red, green, blue : out STD_LOGIC);
  end component;

  signal N_INTVEC  : std_logic_vector( 5 downto 0);
  signal N_DOUT    : std_logic_vector( 7 downto 0);
  signal N_TX      : std_logic;

```

```

signal L_CLK          : std_logic := '0';
signal L_CLK_CNT      : std_logic_vector( 2 downto 0) := "000";
signal L_CLR          : std_logic;           -- reset, active low
signal L_CLR_N        : std_logic := '0';   -- reset, active low
signal L_C1_N         : std_logic := '0';   -- switch debounce, active low
signal L_C2_N         : std_logic := '0';   -- switch debounce, active low

begin
  cpu : cpu_core
  port map(
    I_CLK      => L_CLK,
    I_CLR      => L_CLR,
    I_DIN      => N_DOUT,
    I_INTVEC   => N_INTVEC,

    Q_ADR_IO   => C_ADR_IO,
    Q_DOUT     => C_DOUT,
    Q_OPC      => C_OPC,
    Q_PC       => C_PC,
    Q_RD_IO    => C_RD_IO,
    Q_WE_IO    => C_WE_IO);

  ino : io
  port map(
    I_CLK      => L_CLK,
    I_CLK_50   => I_CLK_50,
    I_CLR      => L_CLR,
    I_ADR_IO   => C_ADR_IO,
    I_DIN      => C_DOUT,
    I_RD_IO    => C_RD_IO,
    I_RX       => I_RX,
    I_SWITCH   => I_SWITCH(7 downto 0),
    I_WE_IO    => C_WE_IO,
    Q_DOUT     => N_DOUT,
    Q_INTVEC   => N_INTVEC,
    Q_LEDS     => Q_LEDS(1 downto 0),
    Q_TX       => N_TX,

    hsynch => hsynch,
    vsynch => vsynch,
    red    => red,
    blue   => blue,
    green  => green);

  -- input clock scaler
  --
  clk_div : process(I_CLK_50)
  begin
    if (rising_edge(I_CLK_50)) then
      L_CLK_CNT <= L_CLK_CNT + "001";
      if (L_CLK_CNT = "001") then
        L_CLK_CNT <= "000";
        L_CLK <= not L_CLK;
      end if;
    end if;
  end process;

  -- reset button debounce process
  --
  deb : process(L_CLK)
  begin
    if (rising_edge(L_CLK)) then
      -- switch debounce
      if ((I_Reset(0) = '0') or (I_Reset(1) = '0')) then -- pushed
        L_CLR_N <= '0';
        L_C2_N <= '0';
        L_C1_N <= '0';
      else -- released
        L_CLR_N <= L_C2_N;
        L_C2_N <= L_C1_N;
        L_C1_N <= '1';
      end if;
    end if;
  end process;

  --> changed 12th June 2010 Serge Moutou for Digilent Spartan 3 starter Board
  --L_CLR <= not L_CLR_N;

```

```
        L_CLR <= L_CLR_N;  
--<  
    Q_LEDS(2) <= I_RX;  
    Q_LEDS(3) <= N_TX;  
    Q_TX <= N_TX;  
end Behavioral;
```

## ANNEXE V : Le GNU C

<code>_BV(x)</code>	positionne un bit spécifique  <u>exemple</u> : <code>char result = _BV(PINA6); // met 64 dans result</code>
<code>void sbi (uint8_t port, uint8_t bit)</code>	positionne un bit spécifique à 1  <u>exemples</u> : <code>sbi (PORTB, 3); //define PORTB 0x18</code> <code>sbi (PORTB, PINA3);</code>
<code>void cbi (uint8_t port, uint8_t bit)</code>	positionne un bit spécifique à 0  <u>exemples</u> : <code>cbi (PORTB, 3); //define PORTB 0x18</code> <code>cbi (PORTB, PINA3);</code>
<code>uint8_t bit_is_set (uint8_t port, uint8_t bit);</code>	teste si un bit spécifique est à 1  <u>exemple</u> : <code>//define PINB 0x16</code> <code>uint8_t result = bit_is_set (PINB, PINB3);</code>
<code>uint8_t bit_is_clear (uint8_t port, uint8_t bit);</code>	teste si un bit spécifique est à 0  <u>exemple</u> : <code>//define PINB 0x16</code> <code>uint8_t result = bit_is_clear (PINB, PINB3);</code>
<code>uint8_t inp (uint8_t port);</code>	lit un registre spécifique et retourne le résultat  <u>exemple</u> : <code>//define SREG 0x3F</code> <code>uint8_t res = inp (SREG);</code>
<code>uint16_t __inw (uint8_t port);</code>	lit un registre spécifique 16 bits et retourne le résultat  <u>exemple</u> : <code>//define TCNT1 0x2C</code> <code>uint16_t res = __inw (TCNT1);</code>
<code>uint16_t __inw_atomic (uint8_t port);</code>	lit un registre spécifique 16 bits et retourne le résultat sans interruption possible  <u>exemple</u> : <code>//define TCNT1 0x2C</code> <code>uint16_t res = __inw (TCNT1);</code>
<code>outp (uint8_t val, uint8_t port);</code>	sort une valeur spécifique sur un port spécifique  <u>exemple</u> : <code>//define PORTB 0x18</code> <code>outp(0xFF, PORTB);</code>
<code>__outw (uint16_t val, uint8_t port);</code>	sort une valeur spécifique 16 bits sur un port spécifique  <u>exemple</u> : <code>//define OCR1A 0x2A</code>

	<code>__outw(0xAAAA, OCR1A);</code>
<code>__outw_atomic (uint16_t val, uint8_t port);</code>	sort une valeur spécifique 16 bits sur un port spécifique sans interruption
<code>    exemple :</code>	<code>    //#define OCR1A 0x0A     __outw_atomic(0xAAAA, OCR1A);</code>

## UART (Universal Asynchronous Receiver Transmitter)

On vous propose deux sous-programmes donnés avec le cœur original. Vous pouvez noter un fonctionnement sans interruption alors que la partie matérielle peut déclencher des interruption pour la RS232.

```
uint8_t uart_putc(uint8_t cc) {
    while ((UCSRA & (1 << UDRE)) == 0);
    UDR = cc;
    return 1;
}

uint16_t uart_puts(const char * s) {
    const char * from = s;
    uint8_t cc;
    while ((cc = pgm_read_byte(s++)) != 0)    uart_putc(cc);
    return s - from - 1;
}
```