

Using a "CoreAtMega8" Core

(Last update : 2012-04-18)

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)

Key words : softcore processor, SoC (System On Chip), C programming language and FPGA, FPGA Spartan3, Digilent Starter Board, VGA screen, ATMEGA8®.

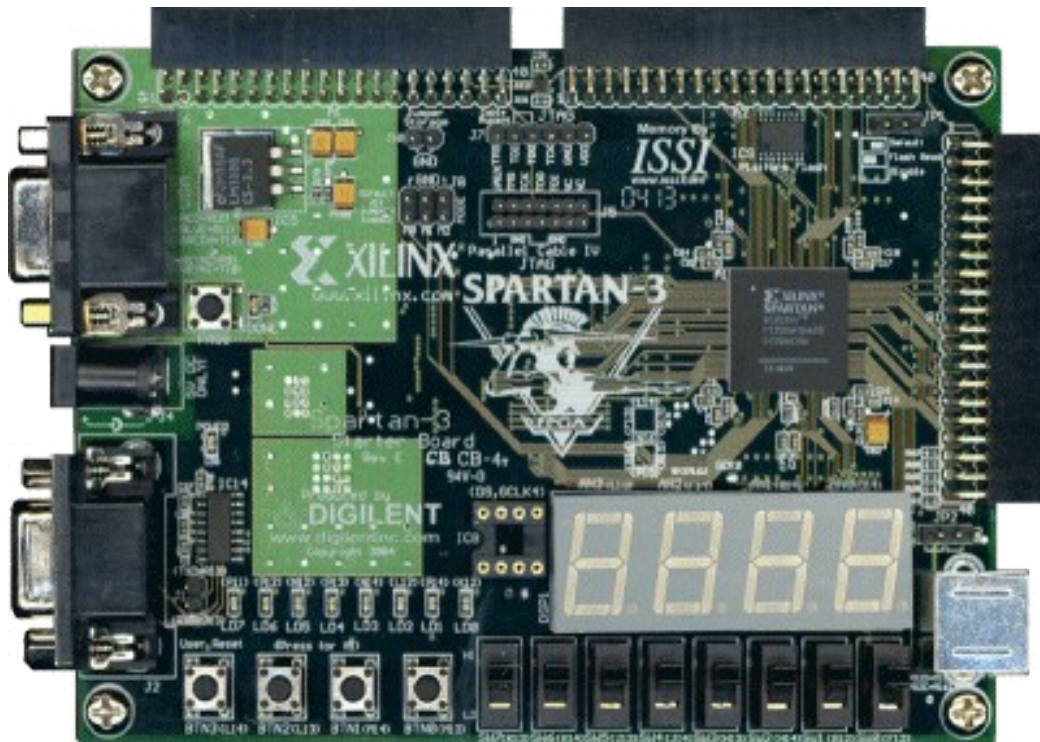
The starting VHDL core and its last modifications are on :
"http://moutou.pagesperso-orange.fr/ER2/ATMega8_pong_VGA.zip"
and the complete solution is in CorrProjet2010 directory.

Introduction

The goal of this project is to construct **a pong game interfaced to a VGA monitor** with FPGA as a target chip. The pong game on VGA and on a FPGA is a classical student project and you can find a lot of examples on the Internet (googlize "pong game VGA" for instance). But what we want to do is to use a free 8-bit soft processor interacting with external logic described in VHDL. The soft core which we refer to as **CoreAtMega8** in this document, is compatible with the Atmel ATMEGA8®. We expect to program this soft core with C language.

The target board is a digilent Board with a Xilinx FPGA, more exactly the **Spartan-3 Starter Board** see :

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



Integrated Development Environment (IDE) used are then :

- ISE Xilinx for VHDL programs or the free WebPack (<http://www.xilinx.com/support/download/>).
- AVRStudio (http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2725) for developing 8-bit AVR applications in Windows NT/2000/XP/Vista/7 environments and using assembler or GNU C (avr-gcc).

This project is intended for three undergraduate students in third semester during 80 hours.

Core Choice

You can find many soft core compatible with the Atmel RISC on the Internet. What determines my choices is often the number of pages of their documentation. This was already the case with a previous soft processor PIC16C57 last academic year (the corresponding report can be downloaded on

["http://moutou.pagesperso-orange.fr/ER2/SiliCore1657_en.pdf"](http://moutou.pagesperso-orange.fr/ER2/SiliCore1657_en.pdf) for the academic year 2009/2010).

Even if you can find other concurrent cores : http://opencores.org/project,avr_core or an other Atmel AVR ATtiny261/461/861 :

<http://opencores.org/project,avrtinyx61core>, I choose this core because it is part of a complete lecture. The **CoreAtMega8** is a soft processor compatible with the Atmel ATMEGA8®. It was developed by Dr. Juergen Sauermann (Germany) and published in January 2010 in OpenCore as a lecture to learn to construct a "soft processor" in VHDL (http://opencores.org/project,cpu_lecture).

A portable soft processor core has to be complete but without ports, RAM and ROM because they are very dependent of the FPGA target. This core is provided with RAM and ROM implementations as Xilinx components. This means if you want it to put in other FPGA, a lot of work wait for you.

Before diving into the VHDL core, we explain the hardware core using Atmel documentation.

Atmel ATMEGA8® Programming Model

ATMEGA8 is a 8-bit microcontroller with instructions coded with 16 bits or 32 bits.

This processor manages interrupts. (See http://en.wikipedia.org/wiki/Atmel_AVR)

Register File Architecture

The data address space is divided in 3 distinct spaces : the register file , I/O registers, and SRAM.

- The 32 [single-byte](#) registers are mapped in as the first 32 addresses. These registers are connected with ALU and data computations use them (addition, subtraction, multiplication, logic operations, logic tests).
- The IO space from adress 0x20 to 0x5F. In this space we find registers which manage hardware (ports, Timers, UART,...).
- SRAM starts from address 0x60. It's where you can save data and the corresponding size is processor dependant : ATMEGA16 has 1ko of RAM).

Most important registers in register set

We give a partial list of registers using the avr-gcc naming convention. For instance, in the official documentation, the b₀ bit of PORTB is called PORTB0 instead of PB0 in the include file of GNU-C. **Grayed registers are present in our core** and others are free.

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	Most significant Byte ofTimer1							
0x2C(0x4C)	TCTN1L	Last significant Byte ofTimer1							

0x23(0x43)	OCR2	Timer/Counter2 output compare register							
0x21(0x41)	WDTCR	-	-	-	WDCE	WDE	WDP2	WDP1	WDPO
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

Instruction Set

We give without explanation all the instructions.

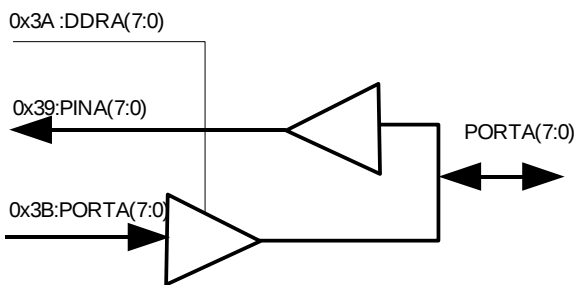
Mnemonics	Operands	Description	Operation	Flags	#Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	$Rd \leftarrow Rd + Rr$	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	$Rd \leftarrow Rd + Rr + C$	Z,C,N,V,H	1
ADIW	RdI,K	Add Immediate to Word	$Rdh:RdI \leftarrow Rdh:RdI + K$	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	RdI,K	Subtract Immediate from Word	$Rdh:RdI \leftarrow Rdh:RdI - K$	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	$Rd \leftarrow Rd \vee Rr$	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	$Rd \leftarrow Rd \vee K$	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	$Rd \leftarrow Rd \vee K$	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

Ports of CoreAtMega8

The way of managing PORTs in this core is very similar as in the Xilinx PicoBlaze core. It is very easy to extend the number of PORTs up to 64 in IO space. The 4 existing PORTs : **PORTA**, **PORTB**, **PORTC** et **PORTD** are not enough for us. The core architecture is for bidirectional PORTs. These bidirectional PORTs are not realized in the core because they are too dependent of FPGA manufacturer.



bidirectional input/output (optional)

You have in fact three ports allowing the realization of one bidirectional PORT as explained in the opposite figure. The three left ports belong to the core while the bidirectional port is external.

Comments : all ports are accessible as a variable. To write in **DDRA** you use the instruction :

```
// C language
DDRA=0xFF; // 1 <=> output in ATmega8
```

while in assembly code we could write :

```
; assembly code
ldi r10,0xFF
out DDRA,r10
```

My second comment is to complete our description. We have to write in C language :

```
PORTA = PINA; // copy PORTA in PORTA
```

instead of PORTA=PORTA.

The PORTs description in this section isn't complete but will be described deeper in a next section when explaining how to use our core with external logic.

It's time to have a look at RAM and ROM.

RAM

RAM is in a space beginning at 0x60 address.

FASM: FPGA and ASIC Subset Model. The FASM model describes a set of available resources that are common to most FPGA and ASIC target devices. This simplifies the task of creating portable IP cores. We don't use FASM model with this core because write cycle has to be synchronous (asynchronous in FASM Model).

The **CoreAtMega8** core gives a VHDL example of memory given below.

```
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);
      ADDRA : in  std_logic_vector(9 downto 0);
      ADDRB : 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,          ADDRB => "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 => "000000000",
  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;

```

As can be easily seen this RAM is constructed with Xilinx "RAMB4_S4_S4" components, a dual port memory. This way of constructing RAM is Xilinx specific. This means, if you want to use ALTERA, ACTEL or other FPGA, you have to work around this VHDL description and modify it to meet new constraints.

ROM

The program to be executed is in this ROM. Here is a partial ROM example.

```

-- 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

-- ***** removed for size *****

end Behavioral;

```

This example is incomplete because we have removed the architecture part. Every new program will give a different ROM. Because every compiler is able to generate hex file, we have to transform this hex file in a VHDL file. This task is performed with a C++ program described further in appendix 1. Its name is "make_mem.cc" given with the original core. Compile this program to generate the binary executable. When done, launch

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

which generates a package in a file named prog_mem_content.vhd, used in the project. This memory is again constructed with Xilinx "RAMB4_S4_S4" components. See the previous discussion in the RAM section for portability.

My first C programs

We use AVRStudio for developing and compiling C programs. The download of AVRStudio allows an use of assembler and the C GNU compiler (with winAVR). Because we are interested in using free tools, GNU compiler is OK for us.

My first C program was for checking the **CoreAtMega8** core on our Digilent Board. Because our **CoreAtMega8** core is a 8 bits architecture with a little SRAM size, it would be interesting to begin with a C program with assembly code.

First C program with assembly code(not checked)

The C program below shows how to include assembly code in a C program.

```
#include <avr/io.h>
main(void)
{
asm volatile (
debut:  in    r24, 0x16      ; 22 : PINB
        com r24            ; complement a un
        out   0x18, r24    ; 24 : PORTB
        rjmp  debut      ; boucle infinie
)
}
```

This program is only given as an example and hasn't been checked.

Only C in a simple Program

A very short program is presented now. Its goal is to recopy PORTB in PORTB. It's probably worthwhile to mention again that physical PORTB input is not physical PORTB output.

```
#include <avr/io.h>
main(void)
{ // managing DDRB and DDRC is not necessary for this core
  DDRB = 0xFF; // 8 outputs in PORTB
  DDRC = 0x00; // 8 inputs in PORTC
  while(1)
  PORTB = PINB; // recopy of PORTB in PORTB which switch on the LEDs
}
```

In the contrary of Silicore1657 already mentioned (last academic year project), the **CoreAtMega8** core has an interrupt service mechanism.

The Interrupt Service Mechanism

This is not a good place for this section and by the way I have to go further in explaining Interrupts : an interrupts manager seems to be present in this core but only RS232 is able to throw them. You have to modify the **CoreAtMega8** core for managing new one (have a look in "io.vhd" file and read "avr/interrupt.h").

It's time to return to the original problem : interfacing a VGA monitor with our core.

Interfacing a VGA Monitor

The Digilent S3 Board has (as many other boards) a VGA port with 5 signals : three for colors (Red, Green and Blue) and horizontal and vertical synchronizations. This is shown in the Figure below.

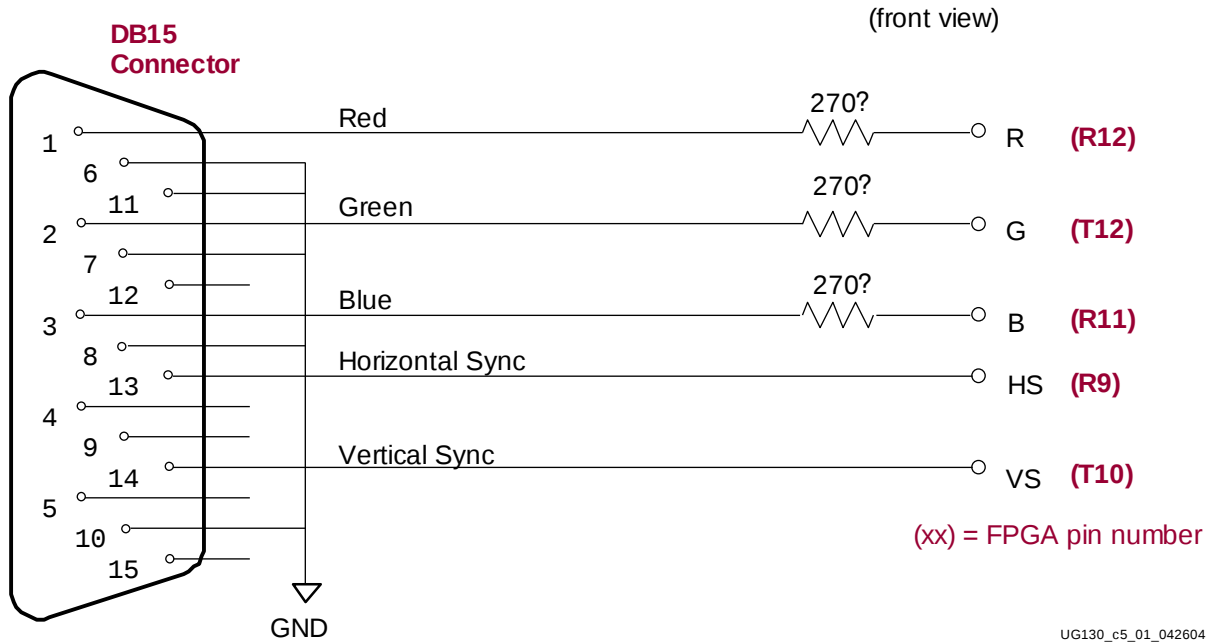
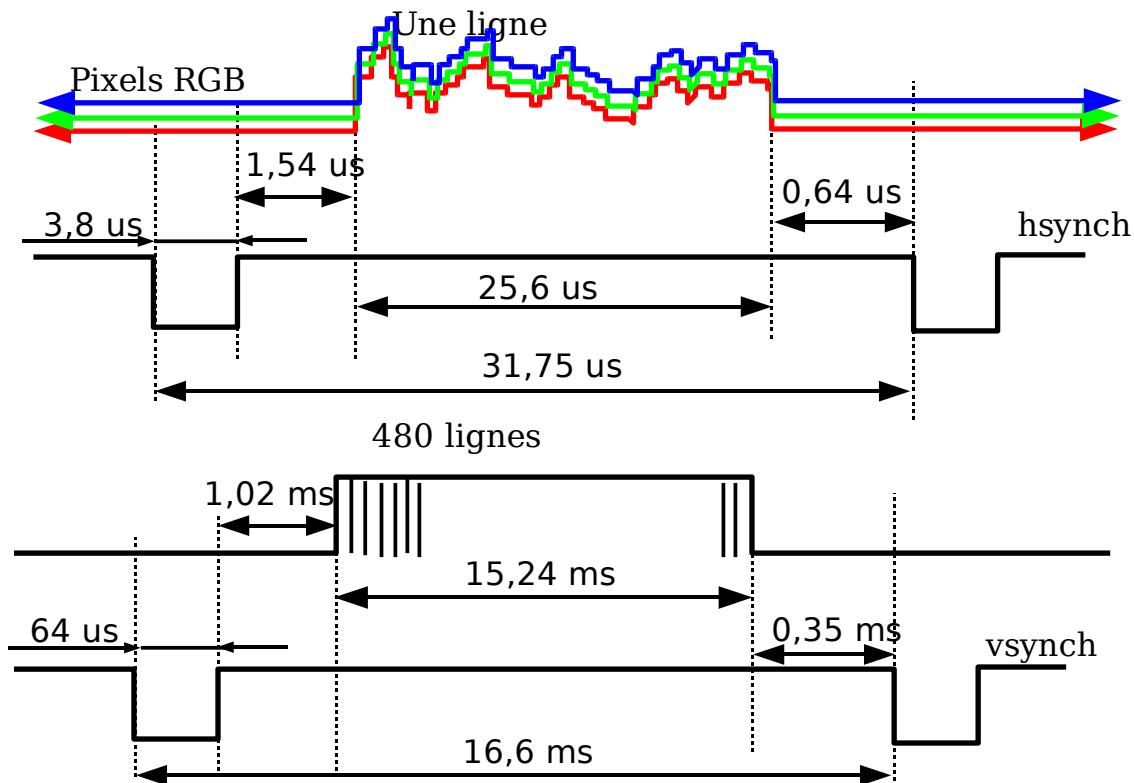


Figure 5-1: VGA Connections from Spartan-3 Starter Kit Board

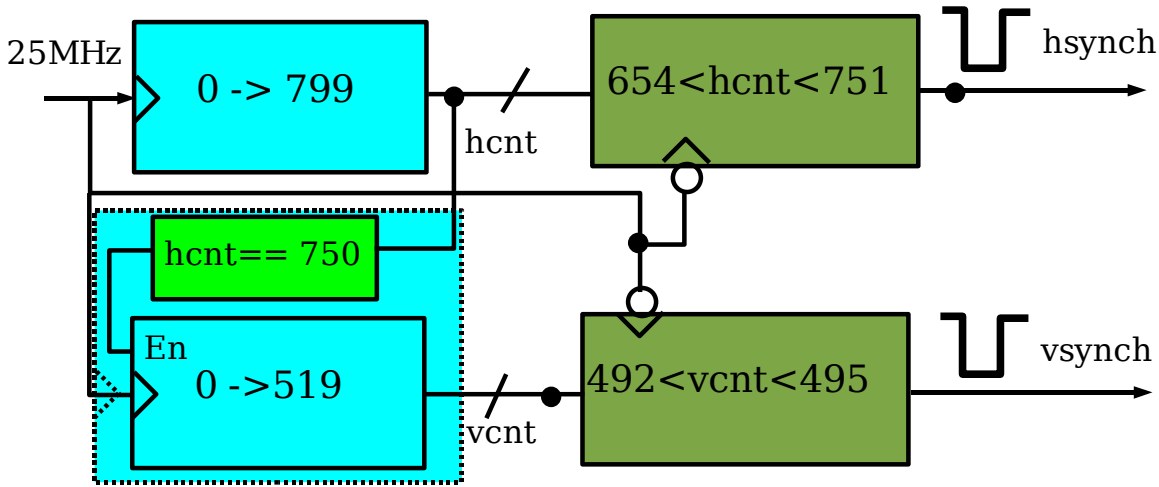
Horizontal and Vertical synchronization

Horizontal and vertical synchronization timings are presented in the Figure below for a 640 by 480 resolution. To meet these times, we only use two counters.



Here is how things are working with two counters (it's the "My_vga_synch.vhd" File

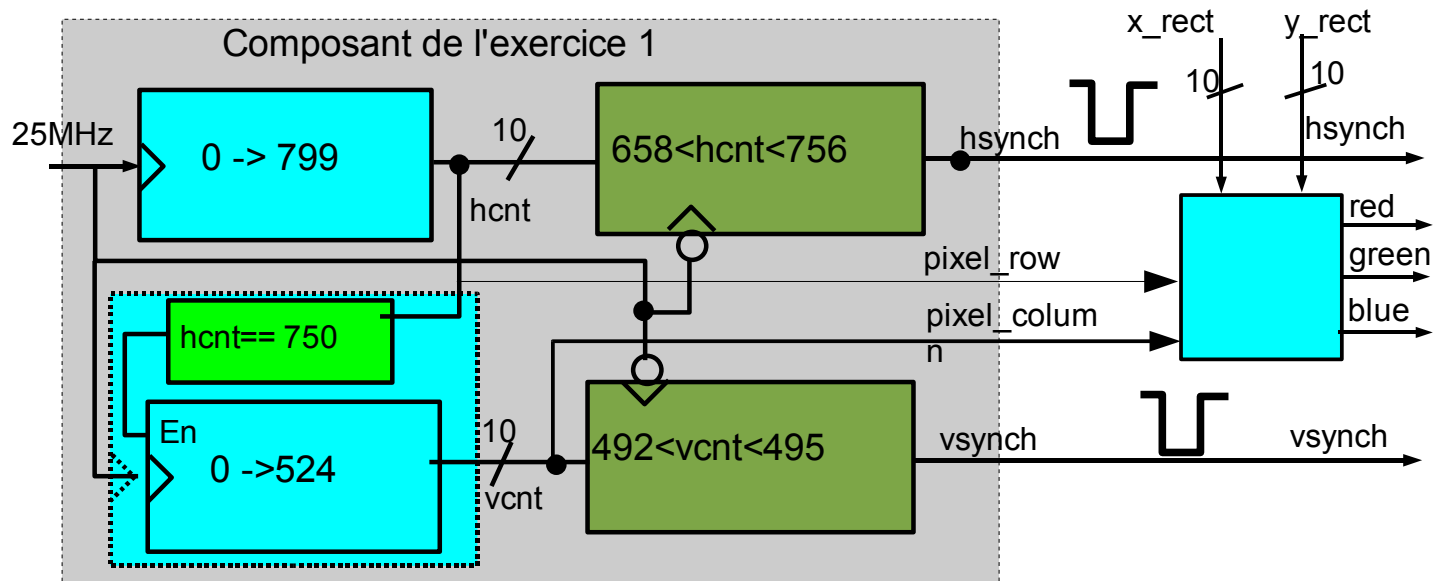
content) :



Comparators are combinational circuits in principle. But, as can be easily seen, they are synchronized on falling edge of the main 25 MHz clock. The corresponding VHDL program able to manage synchronization is given in appendix II. The corresponding VHDL code is only able to synchronize but draw nothing on the screen. Let's see how to draw a rectangle.

Drawing a Rectangle (which will become later a Ball)

In order to draw a rectangle with a position set by an external program we have to change our VGA controller. The rectangle size is fixed, only its position can be changed. Have a look in the below figure and you will see a combinatory part added, in which only rectangle 10 bits positions (here x_rect and y_rect) are present. Every drawing in the screen will be in this component. The x_rect and y_rect values are connected to core ports.



Before coming back into programming subjects we present the top architecture with all the previous components connected together.

Assembly of Core, Memories and VGA Controller

As shown in the above figure we need $2 \times 10 = 20$ bits to modify our ball/rectangle position. Because ports are 8 bits wide, we need four ports only to manage the X and Y positions of the ball.

Managing PORTs in the CoreAtMega8 Soft Processor

The ports management of CoreAtMege8 is relatively simple. We have to modify io.vhd file of the initial project. I make the choice to use existing PORTs (I mean known by C compiler) to realize the project. I have also choosed to manage the rackets positions only with 8 bits. Then I have 6 PORTs to find.

The VHDL file which manage ports is io2.vhd. We give its content now. (We left the RS232 component of the initial core and add the component VGA_Top and modify the "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 DOWNTO 0);
        y_raquG, y_raquD: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        hsynch,vsynch,red,green,blue : out STD_LOGIC);
END component;

signal Balle_xLow : STD_LOGIC_VECTOR(7 DOWNTO 0);
signal Balle_xHigh : STD_LOGIC_VECTOR(7 DOWNTO 0);
signal Balle_yLow : STD_LOGIC_VECTOR(7 DOWNTO 0);
signal Balle_yHigh : STD_LOGIC_VECTOR(7 DOWNTO 0);
signal raqD_y : STD_LOGIC_VECTOR(7 DOWNTO 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_raquG => raqG_y,
           y_raquD => raqD_y,
           hsynch => hsynch,
           vsynch => vsynch,
           red => red,
           green => green,
           blue => blue);

  -- IO read process
  --
  iord: process(I_ADR_IO, 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_IO 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
        & '0' -- 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
        -- Tx complete
    end case;
  end process;
end;

```



```

        & 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
    & '0';         -- 8 bits/char
                -- rising clock edge

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

-- IO 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; --DDR0
                when X"32" => Balle_xHigh <= I_DIN; --PORTD
                when X"37" => Balle_yLow <= I_DIN; --DDR0
                when X"38" => Balle_yHigh <= I_DIN; --PORTB
                when X"34" => raqD_y <= I_DIN; --DDR0
                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 -- no interrupt
                L_INTVEC <= "000000";
            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;

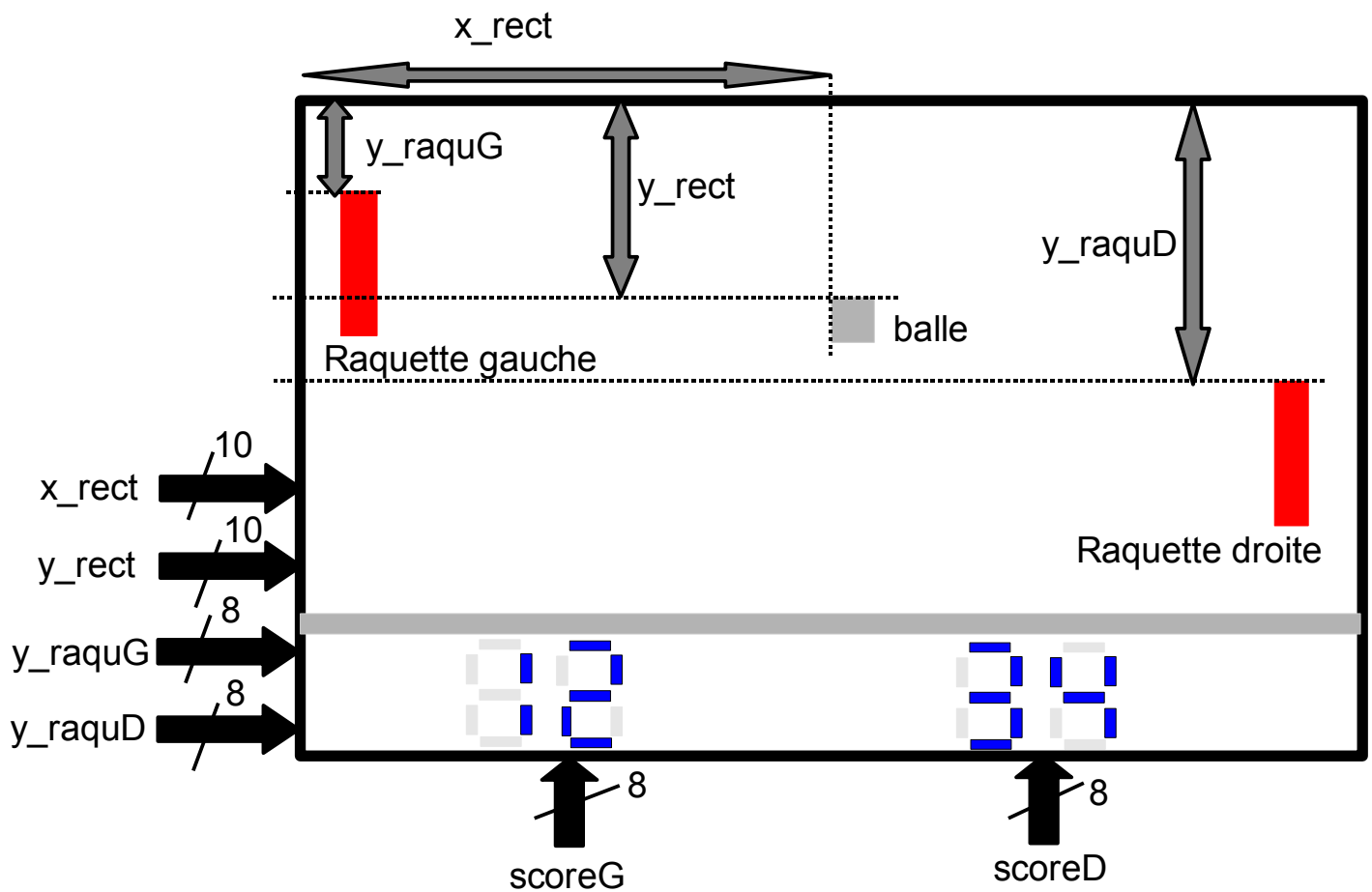
end Behavioral;

```

This VHDL file shows us the connections between ball and rackets coordinates and PORTs.

X position of the ball	x_rect<9:8>	PORTD<1:0>
	x_rect<7:0>	DDRD<7:0>
Y position of the ball	y_rect<9:8>	PORTB<1:0>
	y_rect<7:0>	DDRB<7:0>
left racket	y_raquG<7:0>	PORTC<1:0>
right racket	y_raquD<7:0>	DDRC<7:0>

Here is a schematic to explain the complete interface and the corresponding VGA screen :



We are ready to develop C programs for this hardware.

Programming the Soft Processor with VGA Controller

Previously few C programs have been shown but managing all the new PORTs require any changes.

C programming

We begin with showing a function to write 16 bits in two PORTs.

The "setX" Function

If x is a 16 bits variable (of course the x coordinate of the ball), the function to set it in the corresponding ports is shown below :

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

The content of this function is hardware dependent.

Using the Board Switchs to move the Ball

A complete working example program is shown. Needless to say it's a good thing to have a look in this snippet.

```
#include <avr/io.h>
void setX(uint16_t x);
void setY(unsigned int y);
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){
        //directly setY
        DDRB=PINB;
        setX(PINB);
    }
}

void setX(uint16_t x){
    DDRD=x; //Least Significant Byte
```

```

    PORTD=x>>8;//Most Significant Byte
}

void setY(unsigned int y){
    DDRB=y; // Least Significant Byte
    PORTB=y>>8;//Most Significant Byte
}

```

You can see we don't use the setY function in this program but set directly the corresponding ports. We also show the two ways to use a 16 bits variable : "uint16_t" or "unsigned int" types.

A complete Program to manage the Ball and the Borders

You want to see a bouncing ball on the screen, here is the way :

```

#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; // Least Significant Byte
    PORTD=x>>8;//Most Significant Byte
}

void setY(unsigned int y){
    DDRB=y; // Least Significant Byte
    PORTB=y>>8;//Most Significant Byte
}

```

The most important problem with this kind of program is the limited number of predictable trajectories you have. A game player will quickly be boring. But wait. At this step nobody can play. Rackets are not shown.

Adding motion for rackets

At this step moving the rackets is only done with a C program because we have already connected both Y coordinates with both ports. Hardware has to show the rackets and software has to move them.

Simple Solution without borders

We present now a working example with motionless rackets. Moving the rackets is left to the students as an exercise.

Because rackets have not the same size of the ball and not the same color it's worthwhile to complicate a little the rectangle component. here is our new rectangle component :

```
COMPONENT rect IS PORT(
  row,col,x_rec,y_rec,delta_x,delta_y :in STD_LOGIC_VECTOR(9 DOWNT0 0);
  colorRGB : in STD_LOGIC_VECTOR(2 DOWNT0 0);
  red1,green1,blue1 : out std_logic);
END component;
```

Instanciating this rectangle for the ball and the rackets is shown below :

```
balle:rect port map(row=>srow, col=>scol, red1=>sred, green1=>sgreen, blue1=>sblue,
  colorRGB=>"111", delta_x=>"000001010", delta_y=>"000001100",
  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=>"000001010",
  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;
```

Signals declarations are ommitted in this snippet for simplicity.

Here is a C program which allows bouncing off the moveless rackets. If you want to move the rackets please modify this snipet. You have to modify software but not hardware.

```
#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;
```

```

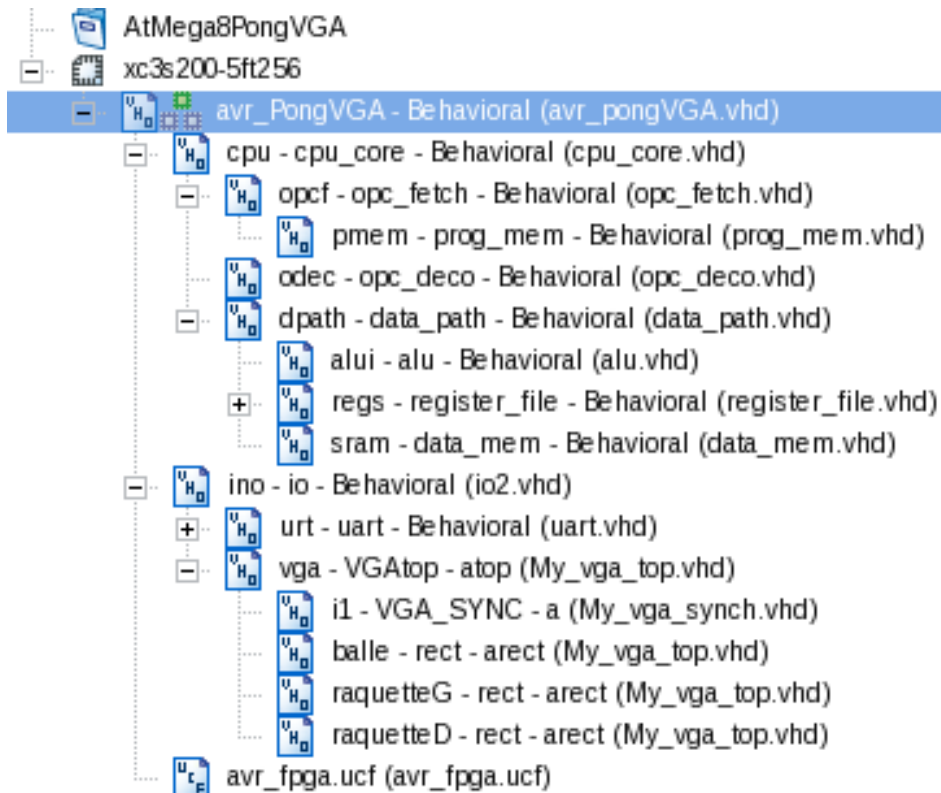
posY=301;
setX(posX);
setY(posY);
while( (posX>30) && (posX<580)){
    posRaqu_16=raqD_y<<1;
    if ((posX>=574) && (posY<posRaqu_16+58) &&
        (posY>posRaqu_16-10) && (deltaX>0)) deltaX= -deltaX;
    posRaqu_16=raqG_y<<1;
    if ((posX<=32) && (posY<posRaqu_16+58) &&
        (posY>posRaqu_16-10) && (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; //Least Significant Byte
    PORTD=x>>8;//Most Significant Byte
}

void setY(unsigned int y){
    DDRB=y; // Least Significant Byte
    PORTB=y>>8;//Most Significant Byte
}

```

The project hierarchy at this point is shown below :



It's easy to construct this project, because I give you all the necessary files.

Download : http://moutou.pagesperso-orange.fr/ER2/ATmega8_pong_VGA.zip

What your Work consists of ?

The complete understanding of hardware (project preparation)

You have to be able to draw all the components and their connexions starting from the given VHDL files.

Translate and understand (project preparation)

You have to translate in French the entire chapter 8 on Inputs/Outputs of the original lecture (lecture.pdf) and to and make a true OpenOffice table for all the AVR instructions.

Developping in C language

Design a C program for managing rackets move with first the switchs and second the joysticks.

Bits check in a PORT (to move the rackets)

Every port is defined in "avr/io.h" file. For instance PORTB is defined with

`#define PORTB _SFR_I08(0x18)` because it is in address $0x18=24$ in Inputs/Outputs memory space (in fact in adress $0x38$). Every PORT bit is also defined :

```

/* 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

```

With these informations, you should understand to check bit PINB1 of port **PINB** you simply write in C :

```
/* Port B */
if ((PINB &(1<<PINB1))== (1<<PINB1))
/* or better */
if (bit_is_set(PINB, PINB1))
```

We choose to connect PINB with switches sw0,sw1,sw6 and sw7 of the board (throw ucf file) , to move rackets but if you have joysticks, all other choice is welcome.

Adding a PORT in our core (needless if rackets use only 8 bits)

By default **PORTA** doesn't exist in the ATmega8. But adding a Port in a soft core in a FPGA is not really a problem (hardware in io2.vhd file). The corresponding software consists only to add :

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

in our C programs. Our "io2.vhd" file (see [Managing PORTs in the CoreAtMega8](#) section) is conversely modified (using the address below with adding 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;  --DDRD
  when X"32" => Balle_xHigh <= I_DIN; --PORTD
  when X"37" => Balle_yLow <= I_DIN;  --DDRB
  when X"38" => Balle_yHigh <= I_DIN; --PORTB
  when X"34" => raqD_y <= I_DIN;     --DDRC
  when X"35" => raqG_y <= I_DIN;     --PORTC
```

Bresenham Line Algorithm

Explore if it is possible to use Bresenham's line algorithm for trajectories and so enlarge considerably the number of them. This algorithm is explain in WIKI : http://en.wikipedia.org/wiki/Bresenham's_line_algorithm

You can directly find a C version in the Internet : google [Bresenham in C](#).

Managing Scores

Every time a ball is lost a new score is send with RS232 to an hyperterminal. It's the main reason we let the RS232 controller of the initial core.

The complete solution

The complete solution is present in the Corrprojet2010 directory where we have explored a Bresenham like algorithm. The only great difference is with score management. It doesn't use RS232 but is directly displayed on the screen.

Please note there was a bug in the original core. This bug is corrected in the correction found in

["http://moutou.pagesperso-orange.fr/ER2/ATMega8_pong_VGA.zip"](http://moutou.pagesperso-orange.fr/ER2/ATMega8_pong_VGA.zip)

in **CorrProjet2010** directory. You can find a C file (demo1.c) which manage Bresenham algorithm for the ball's trajectory.

Acknowledgements

Thanks to Xilinx University Program for giving us gracefully five Digilent S3 Starter Board used with our students projects and practices.

Written with OpenOffice 2.4.1

Using a "CoreAtMega8" Core (2)

(Last update : 2012-04-18)

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)

Key words : softcore processor, SoC (System On Chip), C programming language and FPGA, FPGA Spartan3, Digilent Starter Board, VGA screen, ATMEGA8®.

The starting VHDL core and its last modifications are on :
"http://moutou.pagesperso-orange.fr/ER2/ATMega8_pong_VGA.zip"
and the complete solution is in CorrProjet2011 directory.

A second project in the next academic year (2011/2012) uses again this **coreAtMega8**. The new goal was to construct a breakout game. I was also tired to compile all the project when I only want to change a line in the C program, I decided then to explore new areas and if possible to separate completely the hardware part and the software part. Student are working only on the hardware for the half of project (almost 40 hours) and program all the hardware for the rest of the project.

Hardware Part

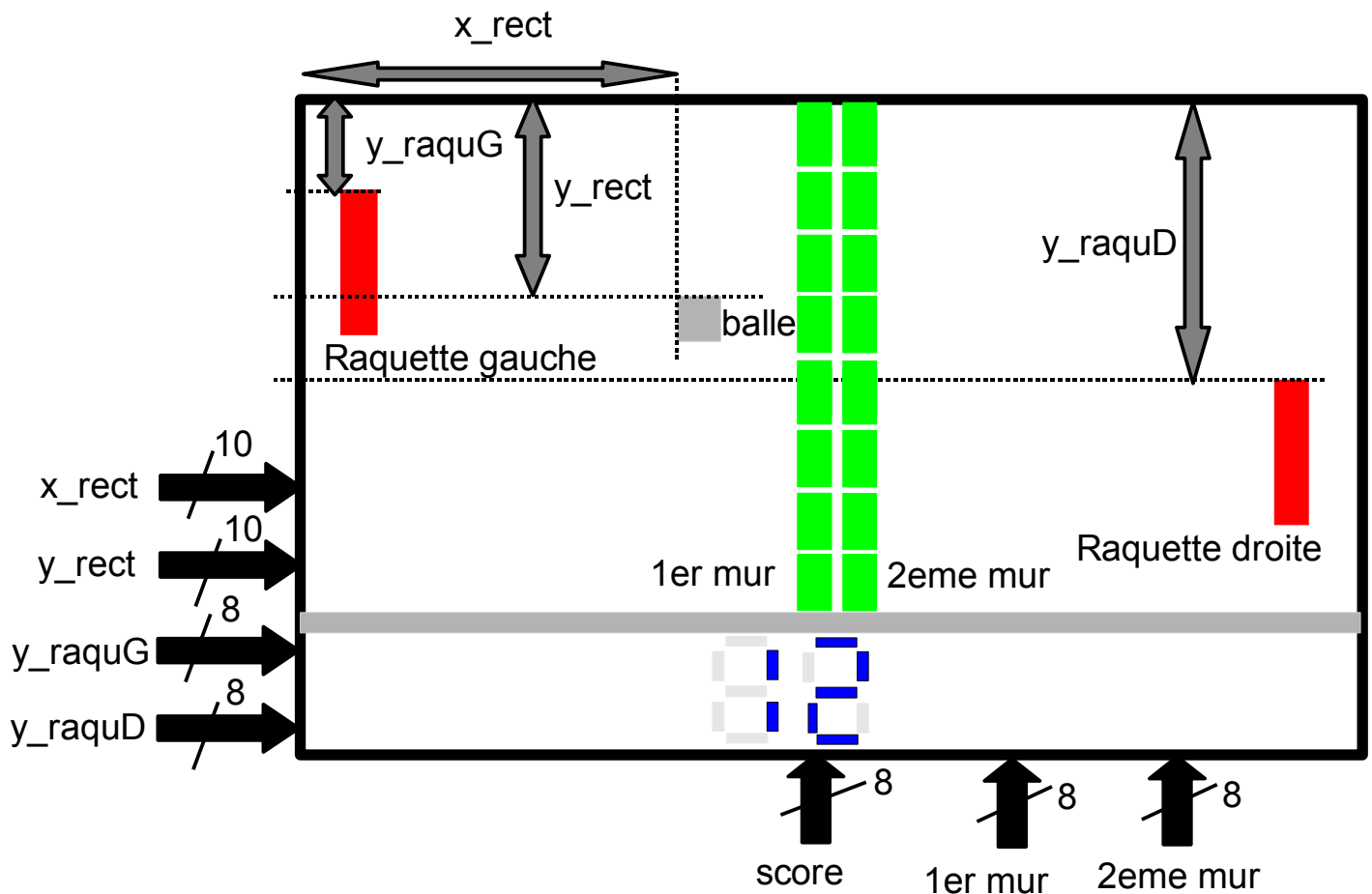
Simplifying considerably what we are doing, we can say we are making a primitive graphic card associated with a processor. We will then solve this project in three stages :

- making a simple graphic card,
- making the interface between the core and the graphic card
- programming all the hardware part.

Making a simple graphic card

Your goal is to take the hardware part of the previous academic year and to transform it like the schematic below. This means you have to transform two seven segments displays into two break walls. You have also to add a corresponding interface.

This is not always the case but here we can check with the switch of the board all the PORTs one by one and see what happens when modifying a switch.

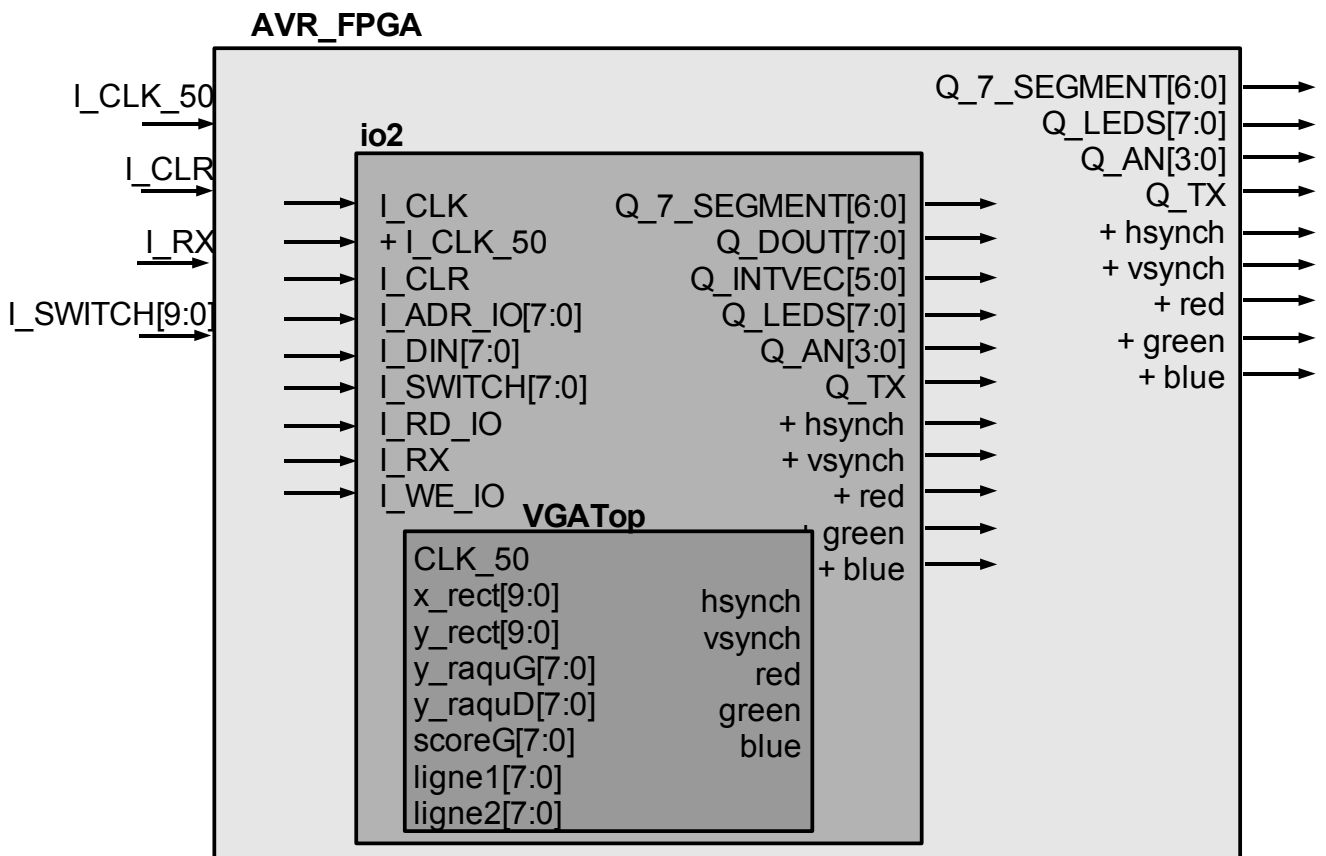


Interfacing the graphic card

This interfacing problem has already been tackled in the previous project but we give here an other point of view.

Please have a look at the following schematic. We have added a "+" before every inputs or output we have added. I mean we start with the original core ("avr_fpga.vhd") where you can already find the "io.vhd" file. See how they are represented in the figure and that io.vhd is copied and renamed as "io2.vhd". In this component the "VGATop" component is seen. It is simply the previous graphic card.

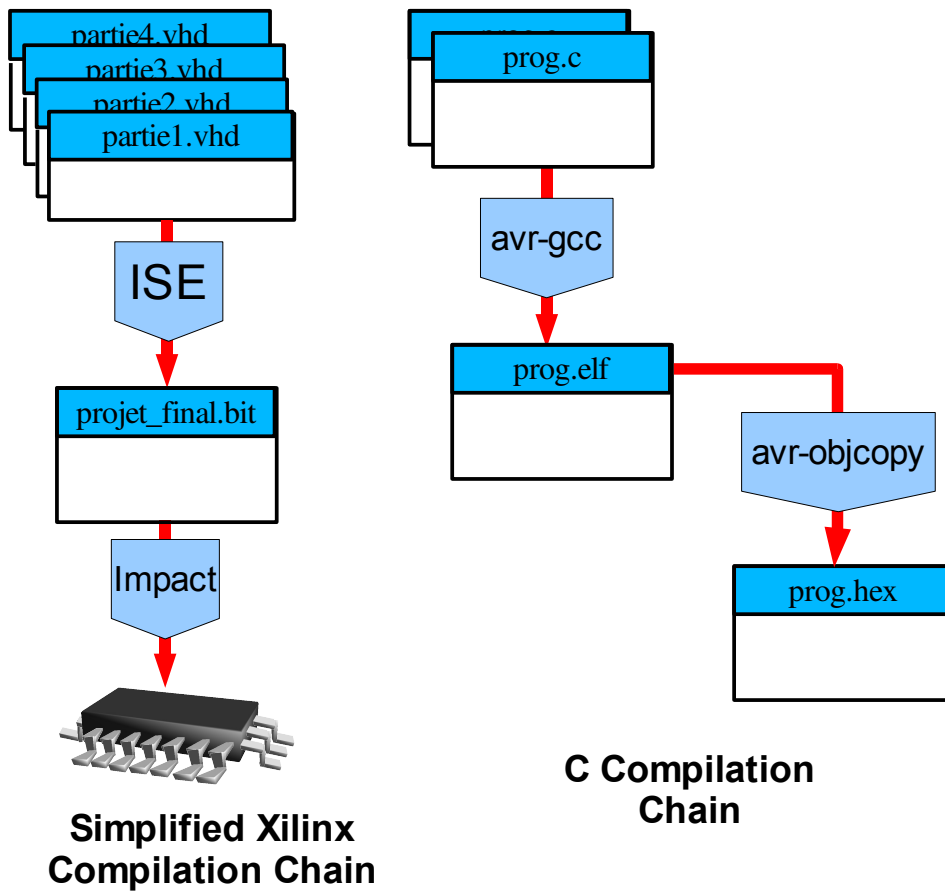
The VHDL counterpart has been tackled in the previous project (in section **Managing PORTs in the CoreAtMega8 Soft Processor**).



Inserting a C Program in a ROM

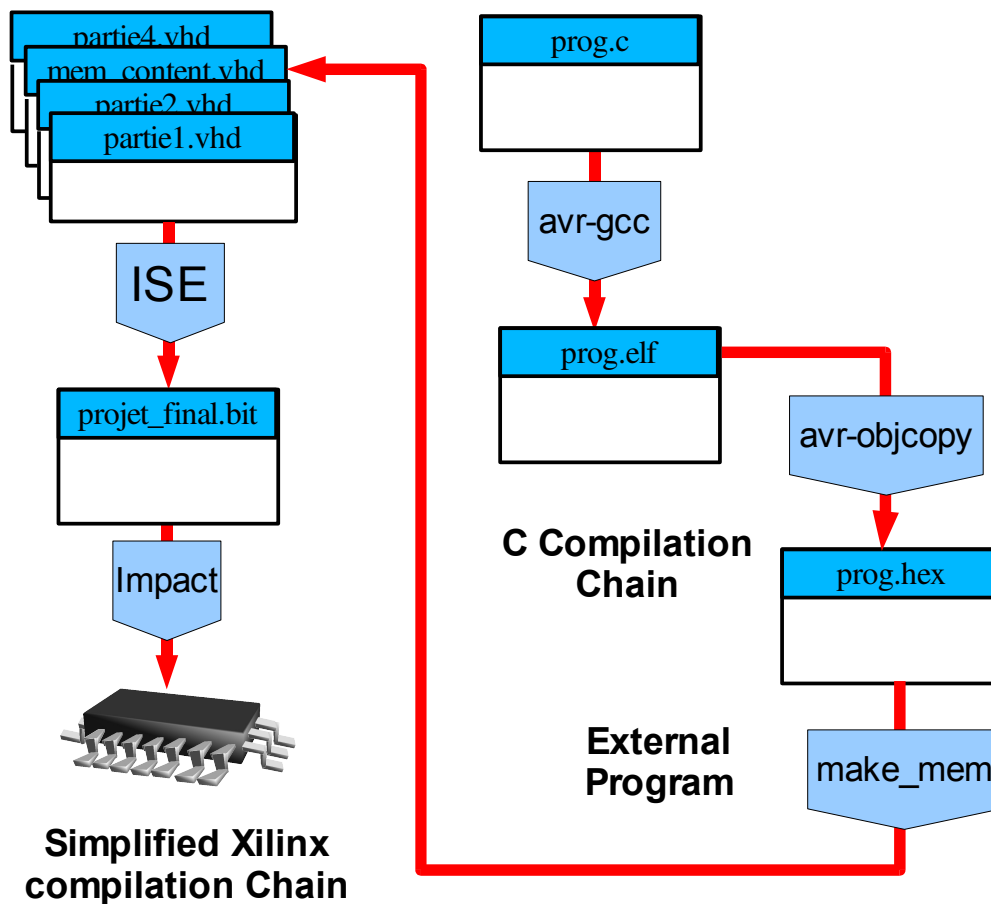
Here we are presenting again the problem with a figure. In the figure below we can see two different compilation chains : Xilinx VHDL compilation chain and C compilation chain. Both chains are using many files together but are disconnected.

Making a program working in a FPGA consists simply in connecting both chains with any way you can find. Because we have already make working programs during the previous academic year, that means we have connected them. But if the corresponding way is the simplest it is also the longest in compilation time, because compiling VHDL files is always a very long process.



The simplest connecting way

This way was used last academic year : it consists of using an external program `make_mem`. This program comes with the entire project with its source c++ code. What is important to note is you make a vhd file describing a ROM starting from a hex file. But what is not apparent in the figure is that after this operation you have to recompile all the vhd files using Xilinx compiling chain.



We have now to investigate if there exist an other quicker process.

Inserting binary programs in a ROM could be done with many ways. Here we put two of them and examine after the second one :

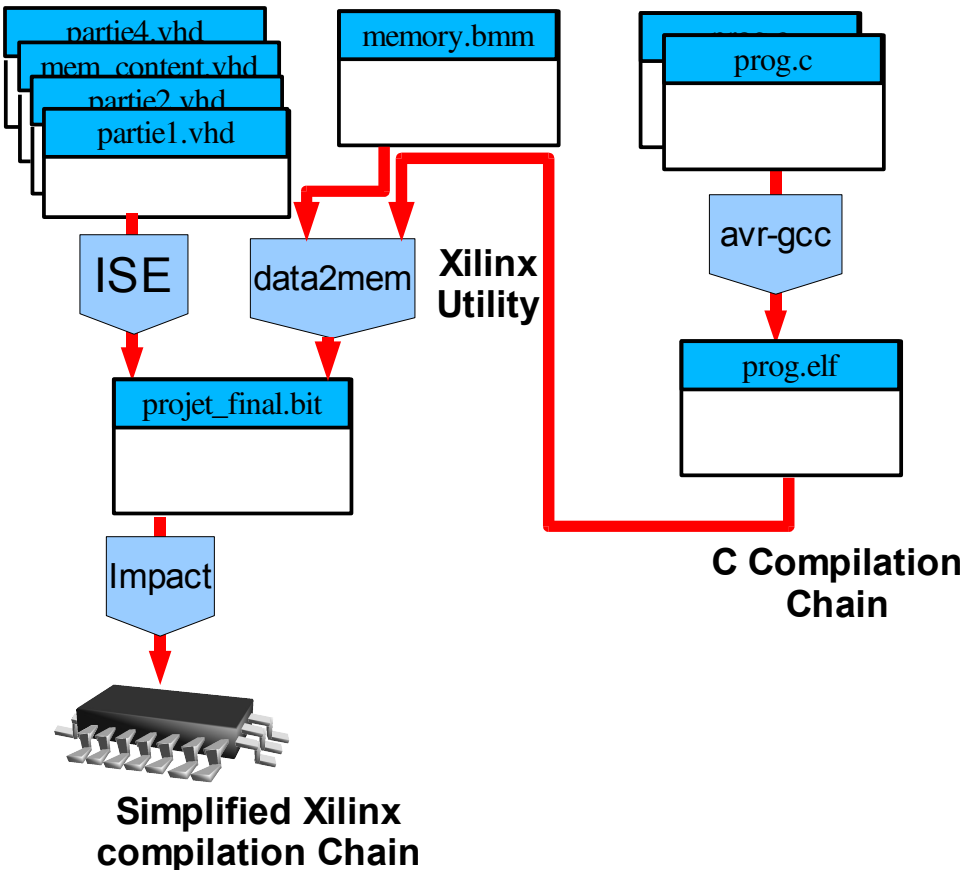
- using a bootloader with RS232
- using a specific utility

Using data2mem Xilinx specific utility

Again and again a Figure is given as explanation. You see a Xilinx utility named data2mem which is able to take an elf file and put it directly in the bit file. But as you can also discover with the figure a new file is needed with the extension BMM. This file describes how the BRAM are connected to form the single program ROM. This program is present in the CorrProjet2011 directory of

["http://moutou.pagesperso-orange.fr/ER2/ATmega8_pong_VGA.zip"](http://moutou.pagesperso-orange.fr/ER2/ATmega8_pong_VGA.zip)

Please note that such a file need a corresponding constraint file also present.



Appendix I (transforming a HEX into VHDL)

In order to transform a hex file into a VHDL file we use the C++ program presented below.

When compiled the binary executable name is make_mem.exe (under windows) or make_mem under Linux.

The transform is done with :

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

if the file to convert is demo1.hex.

The corresponding VHDL file is prog_mem_content.vhd as a 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);
            d += 2;
            buffer[addr + l] = byte;
            csum += byte;
        }
    }
}
```



```

        csum = 0xFF & -csum;
        const uint32_t sum = get_byte(d);
        assert(sum == csum);
    }
}
//-----
//
// copy a slice from buffer into slice.
// buffer is organized as 32-bit x items.
// slice is organized as bits x items.
//
void copy_slice(uint32_t slice_num, uint32_t port_bits, uint32_t mem_bits)
{
    assert(mem_bits == 0x1000 || mem_bits == 0x4000);

    const uint32_t items = mem_bits/port_bits;
    const uint32_t mask = (1 << port_bits) - 1;
    const uint8_t * src = buffer;

    memset(slice, 0, sizeof(slice));

    for (uint32_t i = 0; i < items; ++i)
    {
        // read one 32-bit value;
        const uint32_t v0 = *src++;
        const uint32_t v1 = *src++;
        const uint32_t v2 = *src++;
        const uint32_t v3 = *src++;
        const uint32_t v = (v3 << 24 |
                           v2 << 16 |
                           v1 << 8 |
                           v0) >> (slice_num*port_bits) & mask;

        if (port_bits == 16)
        {
            assert(v < 0x10000);
            slice[2*i] = v;
            slice[2*i + 1] = v >> 8;
        }
        else if (port_bits == 8)
        {
            assert(v < 0x100);
            slice[i] = v;
        }
        else if (port_bits == 4)
        {
            assert(v < 0x10);
            slice[i >> 1] |= v << (4*(i & 1));
        }
        else if (port_bits == 2)
        {
            assert(v < 0x04);
            slice[i >> 2] |= v << (2*(i & 3));
        }
        else if (port_bits == 1)
        {
            assert(v < 0x02);
            slice[i >> 3] |= v << ((i & 7));
        }
        else assert(0 && "Bad aspect ratio.");
    }
}
//-----
//
// write one initialization vector
//
void
write_vector(FILE * out, uint32_t mem, uint32_t vec, const uint8_t * data)
{
    fprintf(out, "constant p%u_%2.2X : BIT_VECTOR := X\\", mem, vec);
    for (int32_t d = 31; d >= 0; --d)
        fprintf(out, "%2.2X", data[d]);
}

```

```

    fprintf(out, "\n";\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 * vhdl_file = 0;

    if (argc) { hex_file = *argv++; --argc; }
    if (argc) { vhdl_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);  
}  
//-----
```

Appendix II VHDL module to manage VGA

Here is a complete VHDL module to manage completely VGA synchronization with a ball and two rackets.

```
-- ***** My_vga_sync.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;

```

Appendix III (ucf File)

```

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";

```

Appendix IV (pong.vhd File)

Here is the VHDL file at the top of the hierarchy describing then how components are connected :

```

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;
```

Appendix V : GNU C

BV(x)	set a specific bit
<u>example</u> :	<code>char result = BV(PINA6); // set 64 in result</code>
void sbi (uint8_t port, uint8_t bit)	set a specific bit to 1
<u>examples</u> :	<code>sbi (PORTB, 3); //define PORTB 0x18 sbi (PORTB, PINA3);</code>
void cbi (uint8_t port, uint8_t bit)	set a specific bit to 0
<u>examples</u> :	<code>cbi (PORTB, 3); //define PORTB 0x18 cbi (PORTB, PINA3);</code>
uint8_t bit_is_set (uint8_t port, uint8_t bit);	check if a specific bit is set to 1
<u>example</u> :	<code>//define PINB 0x16 uint8_t result = bit_is_set (PINB, PINB3);</code>
uint8_t bit_is_clear (uint8_t port, uint8_t bit);	check if a specific bit is set to 0
<u>example</u> :	<code>//define PINB 0x16 uint8_t result = bit_is_clear (PINB, PINB3);</code>
uint8_t inp (uint8_t port);	read a specific 8 bits register
<u>example</u> :	<code>//define SREG 0x3F uint8_t res = inp (SREG);</code>
uint16_t __inw (uint8_t port);	read a specific 16 bits register
<u>example</u> :	<code>//define TCNT1 0x2C uint16_t res = __inw (TCNT1);</code>
uint16_t __inw_atomic (uint8_t port);	read a specific 16 bits register without interruption
<u>example</u> :	<code>//define TCNT1 0x2C uint16_t res = __inw (TCNT1);</code>
outp (uint8_t val, uint8_t port);	set a value into a specified port
<u>example</u> :	<code>//define PORTB 0x18 outp(0xFF, PORTB);</code>
__outw (uint16_t val, uint8_t port);	set a 16 bits value in a specific port
<u>example</u> :	<code>//define OCR1A 0x2A __outw(0xAAAA, OCR1A);</code>

```
__outw_atomic (uint16_t val,  
uint8_t port);
```

exemple :

```
//#define OCR1A 0x0A  
__outw_atomic(0xAAAA, OCR1A);
```

set a 16 bits value in a specific port without interruption