Using Silicore SLC1657 Soft Processor

Last update : 2010-08-25

Serge Moutou's project report 2009/2010 (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, PIC16C57

Introduction

The goal of this project is to construct **a pong game on VGA screen** in a FPGA. The pong game on VGA is a classical subject and you can find a lot of examples in the Internet. But what we want to do is to use a free 8 bits soft processor interacting with external logic described in VHDL. The soft core which we refer to as **SLC1657** in this document, is compatible with the microchip PIC®16C57. 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/).

- MPLAB (http://www.microchip.com/stellent/idcplg?

IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en019469) for developing 8-bit PIC1657 applications in Windows NT/2000/XP/Vista/7 environments and using assembler or Hi-Tech C (http://www.htsoft.com/). Hi-Tech C is free for a lite version when installing MPLAB.

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

The SLC1657 is a free soft core (or soft processor) able to run the same instructions set as the microchip PIC® 16C57. It could be found in the internet : http://www.embeddedtronics.com/public/misc/slc1657.zip

It was developped by SiliCore (last version was update in September 2003) :

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

We first present the hardware core starting from Silicore documentation and that of Microchip.

The PIC® 16F57 Architecture

It is a 8-bit processor with a 12-bit coded instruction set. Then its architecture is smaller than the famous 16F84 with 14-bit instructions set.

An other limitation is its stack : only two levels. It's probably very hard to program a C compiler with this architecture.

This processor has no interrupt.

We begin with the more difficult part : the Register File. I use the word difficult only because of the memory banking a microchip particularity even with more recent microcontroller (till 18FXXX series) which only disapears with the new 16/32-bit architectures (24FXXX and others)

The register File

The register file is broken up into four banks as presented below.
The register bank is selected by modifying the two bank selection bits RB0 and RB1 (5 and 6) in the INDEX/FSR register. The lower sixteen registers in each bank all map back to BANK 0.

General banked memory starts at 0x10 address.

	Banque 0	Banque 1	Banque 2	Banque 3	
00h	Indirect addr.				60h
01h	TMR0				
02h	PCL				

03h	STATUS				
04h	FSR				
05h	PORTA				
06h	PORTB				
07h	PORTC				67h
08h	8 Registres	28h	48h	68h	
	généraux	identiques	sur les 4	banques	
0Fh		2Fh	4Fh	6Fh	
10h	16 registres généraux	30h		70h	
		16 registres	16 registres	16 registres	
		généraux	généraux	généraux	
1Fh		3Fh	5Fh	7Fh	

The free size of memory is then :

4 x 16 + 1 x 8 = 72 bytes.

As can be seen it's a very little piece of memory.

Instructions Set

The 32 instructions of PIC® 16F57 are now presented.

Operands :

- f : register file address from 00 to 7F
- W : Working register (accumulator)
- d : destination selection : d=0 store result in W , d=1 store result in f
- bbb : Bit address within an 8-bit file register (3 bits)
- k : Literal field, constant data or label (8, or 9 bits)
- PC program counter
- TO Time Out bit
- PD Power Down bit

BYTE-ORIENTED FILE REGISTER OPERATIONS					
Mnémonic, Operands	Description	Cycles	12 bits Opcode	status affected	notes
ADDWF f[,d]	Add W and f	1	0001 11df ffff	C,DC,Z	1,2,4
ANDWF f[,d]	AND W with f	1	0001 01df ffff	N,Z	2,4
CLRF f	Clear f	1	0000 011f ffff	Z	2
CLRW	Clear W	1	0000 0100 0000	Z	2
COMF f,[d]	Complement f	1	0010 01df ffff	Z	
DECF f[,d]	Decrement f	1	0000 11df ffff	Z	2,4
DECFSZ f[,d]	Decrement f (skip if 0)	1,(2)	0010 11df ffff		2,4
INCF f[,d]	Increment f	1	0010 10df ffff	Z	2,4
INCFSZ f[,d]	Increment f (skip if 0)	1,(2)	0011 11df ffff		2,4
IORWF f[,d]	Inclusive OR W with f	1	0001 00df ffff	Z	2,4
MOVF f[,d]	Move f	1	0010 00df ffff	Z	2,4
MOVWF f	Move W to f	1	0010 000f ffff		1,4
NOP -	No operation	1	0000 0000 0000		
RLF f[,d]	Rotate left f through Carry	1	0011 01df ffff	C	2,4
RRF f[,d]	Rotate right f through Carry	1	0011 00df ffff	C	2,4
SUBWF f[,d]	subtract W from f	1	0000 10df ffff	C,DC,Z	1,2,4
SWAPW f[,d]	Swap f	1	0011 10df ffff		2,4
XORWF f[,d]	Inclusive OR W with f	1	0001 010df ffff	Z	2,4

BIT-ORIENTED FILE REGISTER OPERATIONS					
Mnémonic, Opérands	Description	Cycles	12 bits Opcode	status affected	notes
BCF f,b	Bit Clear f	1	0100 bbbf ffff		2,4
BSF f,b	Bit Set f	1	0101 bbbf ffff		2,4
BTFSC f,b	Bit Test f, Skip if Clear	1,(2)	0110 bbbf ffff		
BTFSS f,b	Bit Test f, Skip if Set	1,(2)	0111 bbbf ffff		

LITERAL AND CONTROL OPERATIONS					
Mnémonic, Operands	Description	Cycles	12 bits Opcode	status affected	notes
ANDLW k	AND literal with W	1	1110 kkkk kkkk	Z	
CALL k	Call subroutine	2	1001 kkkk kkkk		1
CLRWDT -	Clear Watchdog Timer	1	0000 0000 0100	/TO,/PD	
GOTO k	Unconditional branch	2	101k kkkk kkkk		
IORLW k	inclusive OR literal with W		1101 kkkk kkkk	Z	
MOVLW k	Move Literal to W	1	1100 kkkk kkkk		
OPTION k	Load OPTION register	1	0000 0000 0010		
RETLW k	Return, place Literal in W	2	1000 kkkk kkkk		
SLEEP	Go into standby mode	1	0000 0000 0011	/TO,/PD	
TRIS f	Charge le registr TRIS	1	0000 0000 0fff		3
XORLW k	exclusive OR literal with W	1	1111 kkkk kkkk	Z	

Note 1: The 9th bit of the program counter will be forced to a '0' by any instruction that writes to the PC except for GOTO (see Section 6.5 for more on program counter).

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

3: The instruction TRIS f, where f = 5, 6 or 7 causes the contents of the W register to be written to the tristate latches of PORTA, B or C respectively. A '1' forces the pin to a hi-impedance state and disables the output buffers.

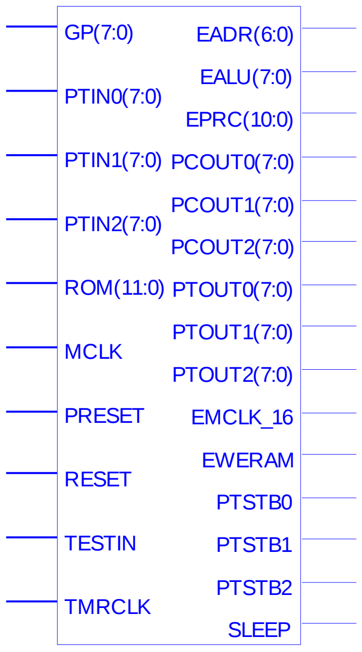
4: If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared (if assigned to TMR0).

Create your own microcontroller with the SLC1657.

The Silicore© SLC1657 is delivered as a VHDL soft core module, and is intended for use in both ASIC and FPGA type devices.

The Core

The core is presented as a schematic and as a VHDL program.



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

For this core, EPRC is the ROM address bus, while ROM is the corresponding data bus of this ROM. The RAM is managed with EALU as adress bus and GP as data bus.

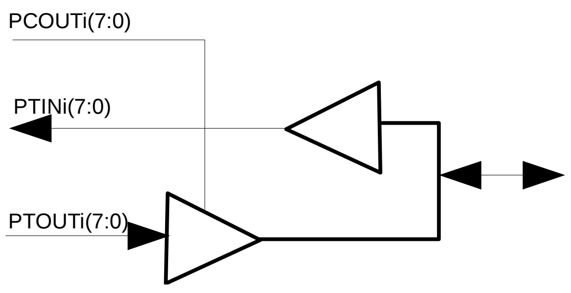
Please note that in the contrary of PIC® 16F57 the SLC1657 use a dedicated entry TMRCLK for timer0 clock (for PIC® it's the b_4 bit T0CKI of PORTA).

Before going further with peripherals we will explore how the I/O ports are working.

The Three SLC1657 Ports

Normally ports are bidirectional entities in a microcontroller. How to synthetize a bidirectional port is FPGA dependant. Then, no bidirectional ports can be found in the SLC1657 core.

In principle, a dedicated register manage bidirectionality. The corresponding registers are present in the core with the names **PCOUT0**, **PCOUT1** and **PCOUT2**. They are all 8 bits in the contrary of PIC® 16F57 where TRISA is a 4 bits register. The I/O ports are in fact in ports named **PTIN0**, **PTIN1** and **PTIN2** and out ports named **PTOUT0**, **PTOUT1** and **PTOUT2**.



In the opposite Figure, you see on the left three ports (of the SLC1657) and on the right one bidirectional port. You can use the three ports in this way or keep the ports without any change : we will keep the three ports in our project.

Input/Output bidirectional PORT

Remark : for compatibility reasons we write in the PCOUTi port in a particular way. If i=0, writing in PCOUT0 is done with the C assertion :

```
// C language
TRISA=0xFF;
```

and in assembly language we could write :

```
; assembly language
movlw  255      ;1:input
TRIS   PORTA  ;PORTA in input
```

Don't read the comments in this program because they are only appropriate for a bidirectinal port that we don't use in this project.

RAM and ROM are necessary peripherals to which we now turn.

The RAM

RAM has already been described in the point of view of its capacity. For SLC1657 core this memory has to be synchronous write but asynchronous read. In the following, we list the program of the implementation :

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

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

ROM : Program Memory

The ROM contains the program to be executed. Because every compiler or assembler generate a 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 "HEX2VHDL.c" given with the original core (slightly modified). Compile this program to generate the binary executable. When done, launch

```
HEX2VHDL  demo
```

which starts from demo.hex and generates file named demo.vhd, used in the project. As can be shown below the entity name is PIC_ROM.

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

entity PIC_ROM is
  port (
            Addr    : in   std_logic_vector(10 downto 0);
            Data    : out  std_logic_vector(11 downto 0));
end PIC_ROM;


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

```
        "101000000100" When to_integer(unsigned(Addr)) = 0006 Else
        "101000000000" When to_integer(unsigned(Addr)) = 2047 Else
        "000000000000";
    end first;
```

This example shows a content but every C programs will give a different content.

Let's turn to VGA hardware.

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 below Figure.

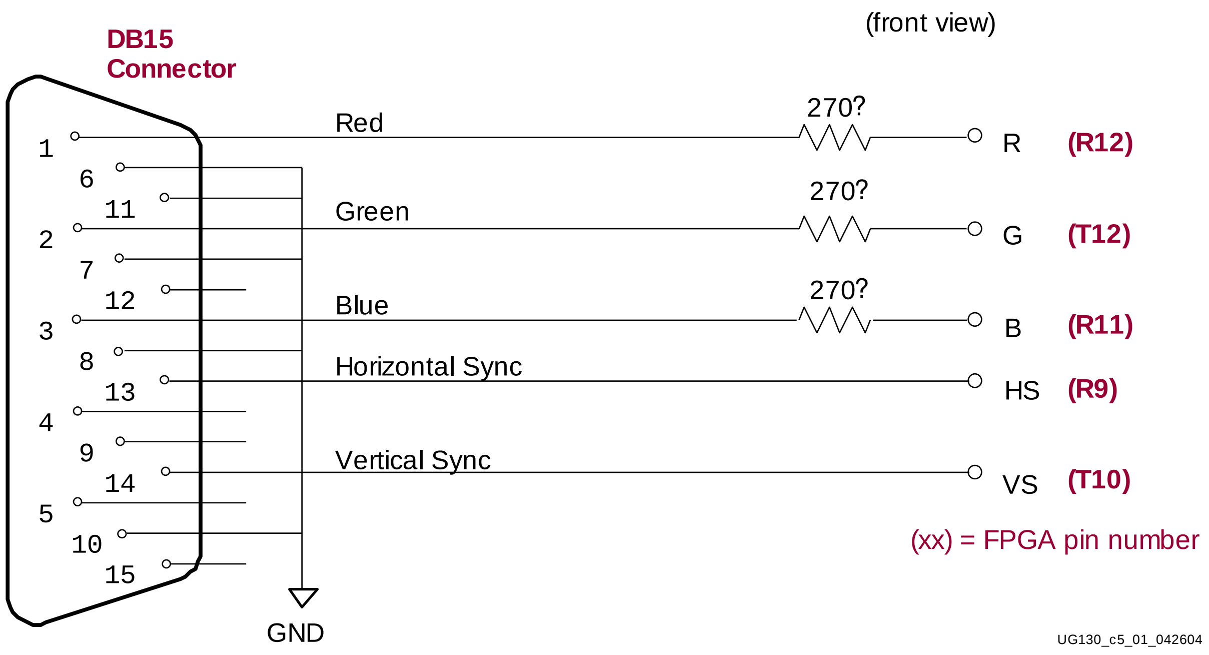
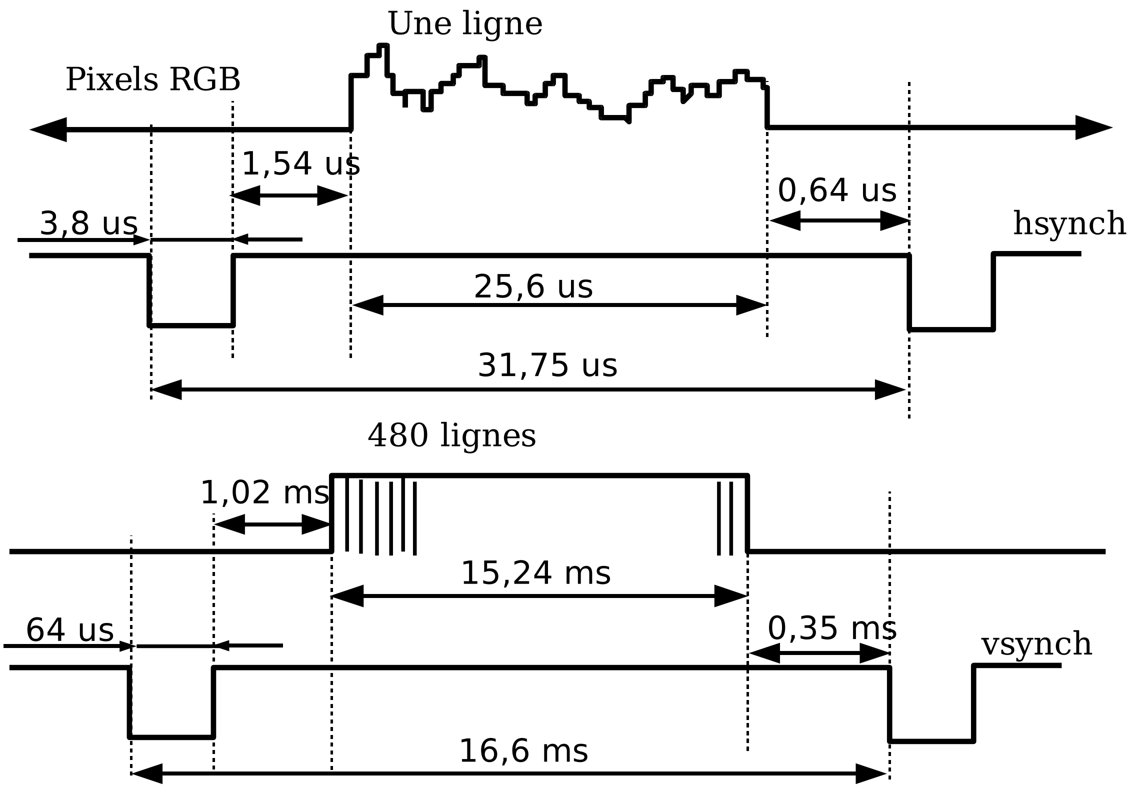


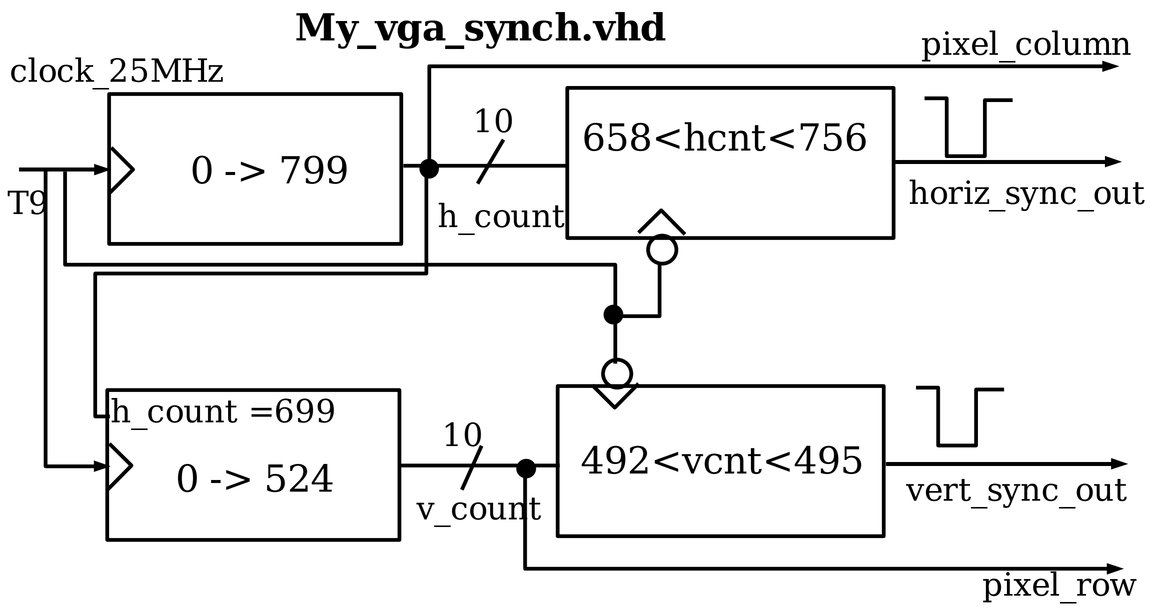
Figure 5-1: **VGA Connections from Spartan-3 Starter Kit Board**

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.



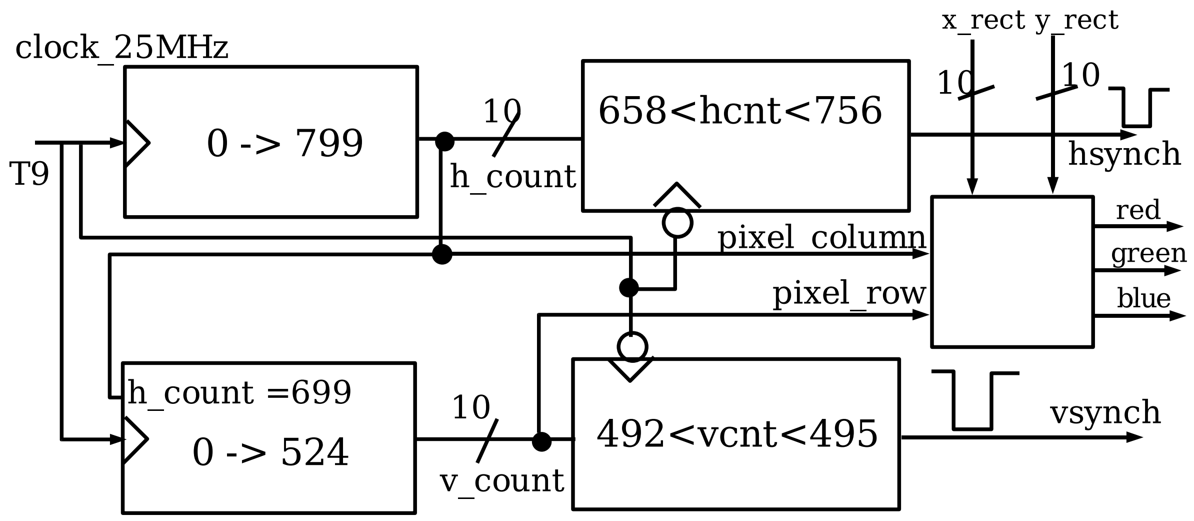
Here is how things are working with two counters (it's the "My_vga_synch.vhd" File content) :



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 at the figure below 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.

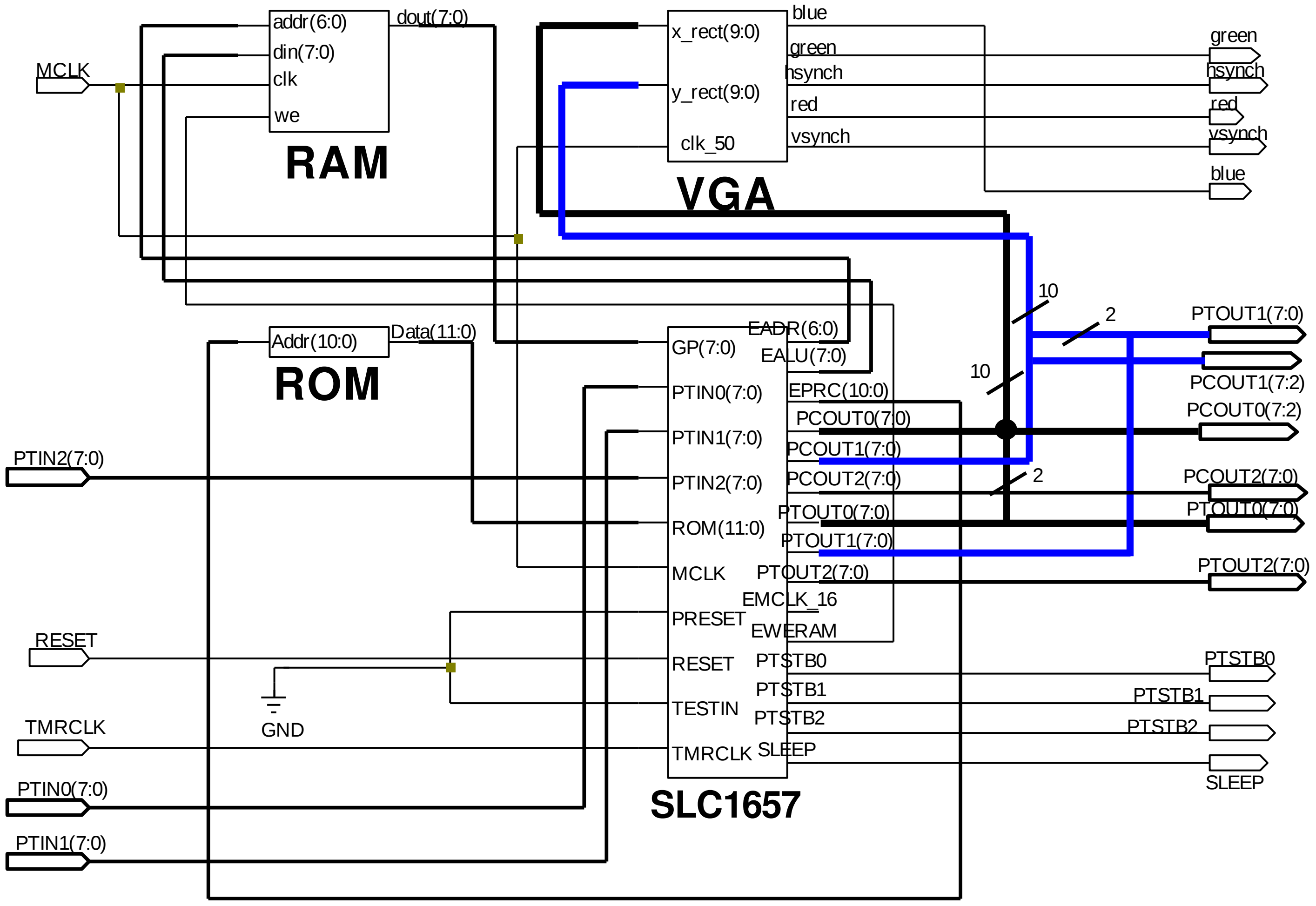
Assembly of Core, Memories and VGA Controller

As shown in the above figure we need 2x10=20 bits to modify our ball/rectangle position. Because ports are 8 bits, we need four ports only to manage the X and Y positions of the ball.

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

Please note we have already explained why TRISA (and TRISB) are true ports (in the contrary of the corresponding PIC®16C57 architecture).

At this stage, a block diagram shows the different parts and their connections.



The Core Programming

We use MPLAB for developping and compiling C programs. MPLAB is a free intergated development environment (IDE) which allows an use of assembler and the Hi-Tech C compiler. Because we are interested in using free tools, Hi-Tech C compiler is OK for us even if this free compiler doesn't support optimizations.

Now we trun first to assembly language.

Assemby Language Programming

The aim of this project was to avoid assembly language for C language. Hereafter, we restrict our attention mainly to few PIC16F5x instructions.

We list in the following a program which doesn't manage our VGA hardware but shows again how to write in TRIS register.

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

        R10     equ    10

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

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

end
```

Here is an other example which allows us a check of the hardware.

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

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

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

The **PORTA** is connected to the switchs of the board.

C Language Programming

C programming is in general more easy. But with a so little processor it could be interesting to have skills on inserting assembly language in a C program.

C with Assembly Language

Let's begin with a C program with assembly.

```
#include <pic16F5x.h>
void main(){
```

```
#asm
Start:    movf      _PORTA, W
                    ;movlw  01        ;0:output
                    TRIS     _PORTA            ;PORTA en entree
                    movwf   _PORTA
                    movlw   1                  ;1:input
                    TRIS     _PORTB            ;PORTB en sortie
                    movf     _PORTA, W
                    movwf   _PORTB
                    goto Start
#endasm
}
```

This program is given as a demonstration. It does'nt do anything interesting and particularly not what stands in comments.

If you want to use PORTA instead of _PORTA you have to add an inclusion directive after #asm directive as shown below :

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

Setting the ball in a precise position is done with two subroutines setX and setY. Because both are very similar we only give setX subprogram.

The "setX" Subprogram

First a C function

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

The harware completely determines the content of this function. The same with assembly language in our C function is shown beow :

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

A horizontal Bouncing Ball

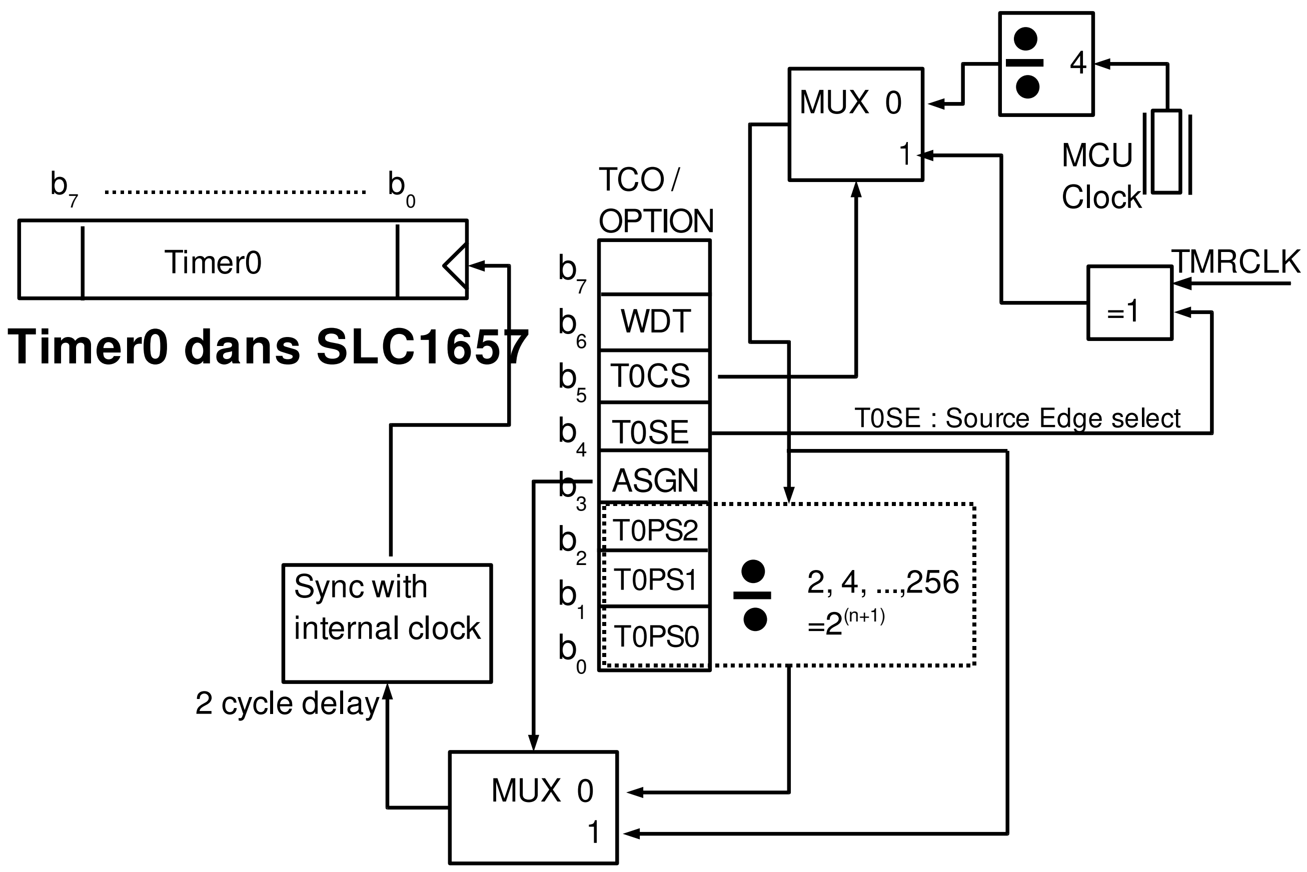
Here is our first program which shows a continuously horizontal bouncing ball.

```
#include <pic16F5x.h>

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

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

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

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

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

As can be seen this program waits with a loop. In fact this program spend a lot of time to wait. Is it possible to wait with timer0 ?

Using Timer0 with C language

The previous program use two wait function calls. Is it possibble to do the same with the only one timer we have : timer0. We first present a schematic which shows how Timer0 works :



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

The corresponding programming example is shown :

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

This new "wait" subprogram is called with a 250 value and is a little quicker than a call of

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

with a 30000 value.

Complete Program managing the Ball

Here is a simple program managing a XY boucing ball :

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

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

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

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

The drawback of this program is the little number of trajectories of the ball. It's not great for a game.

Adding Borders and rackets

Because a racket is only moving in the Y direction it uses only 10 bits (in fact two 8-bit ports). Then a ball and two rackets uses 8 output ports and we have only 6 output ports. SiliCore1657 core has only 3 ports (**PORTA**, **PORTB** and **PORTC**) but we can also use **TRISA**, **TRISB** and **TRISC**. To solve this issue, students choose to manage rackets only with 8 bits and then use only 6 ports.

Simple Solution without Border

At this step we have to connect 6 ports with the VGA hardware. The table below summarize how it's done

X ball position	x_rect<9:8>	TRISA<1:0>
	x_rect<7:0>	PORTA<7:0>
Y ball position	y_rect<9:8>	TRISB<1:0>
	y_rect<7:0>	PORTB<7:0>
left racket	y_raqG<7:0>	PORTC<1:0>
right racket	y_raqD<7:0>	TRISC<7:0>

The size of the ball is not the same of the size of the rackets. If we want to manage such a situation we have to realize rectangle component as shown below :

```
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;
```

Instanciating rectangles for ball and rackets is done as follow :

```
balle:rect port map(row=>srow, col=>scol,r ed1=>sred, green1=>sgreen, blue1=>sblue,
colorRGB=>"111", delta_x=>"0000001010", delta_y=>"0000001100",
              x_rec => x_rect, y_rec => y_rect);
raquetteG:rect port map(row=>srow, col=>scol, red1=>sred1, green1=>sgreen1,
         blue1=>sblue1, colorRGB=>"100", delta_x=>"0000001010",
         delta_y=>"0000111010",       x_rec => "0000010110",
         y_rec(8 downto 1) => y_raquG, y_rec(9)=>'0',y_rec(0)=>'0');
raquetteD:rect port map(row=>srow, col=>scol, red1=>sred2, green1=>sgreen2,
          blue1=>sblue2,colorRGB=>"100", delta_x=>"0000001010",
          delta_y=>"0000111010", x_rec => "1001001000",
          y_rec(8 downto 1) => y_raquD,y_rec(9)=>'0',y_rec(0)=>'0');

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

Signals declarations are omitted. The complete program which manages bouncing on the borders and the rackets is presented now.

```
#include <pic16F5x.h> // Programme pour Hitech C dans MPLAB
void setX(unsigned int x);
void setY(unsigned int y);
void wait(unsigned char tempo);
unsigned int posRaqu_16;
void main(){
int posX,posY;
unsigned char raqD_y=0,raqG_y=0;
signed char deltaX=1,deltaY=1;
```

```
            while(1){
                        posX=113;
                        posY=101;
                        setX(posX);
                        setY(posY);
                        while(RC2==0); // attente départ
                        while( (posX>30) && (posX<580)){
                                    posRaqu_16=raqD_y<<1;
                                    if ((posX>=574) && (posY<posRaqu_16+58) &&
                                         (posY>posRaqu_16-10) && (deltaX>0)) deltaX= -deltaX;
                                    posRaqu_16=raqG_y<<1;
                                    if ((posX<=32) && (posY<posRaqu_16+58) &&
                                        (posY>posRaqu_16-10) && (deltaX<0)) deltaX= -deltaX;
                                    posX=posX+deltaX;
                                    setX(posX);
                                    if ((posY>=460) && (deltaY>0)) deltaY= -deltaY;
                                    if ((posY<=10) && (deltaY<0)) deltaY= -deltaY;
                                    posY=posY+deltaY;
                                    setY(posY);
// gestion des raquettes 2bits PORTC/raquette
                                    if (RC0) if (raqG_y<215) raqG_y++;
                                    if (RC1) if (raqG_y>0) raqG_y--;
                                    PORTC=raqG_y;
                                    if (RC6) if (raqD_y<215) raqD_y++;
                                    if (RC7) if (raqD_y>0) raqD_y--;
                                    TRISC=raqD_y;
                                    wait(250);
                                    wait(250);
                        }
            }
}
```

As can be easily seen, we choose sw0,sw1,sw6 and sw7 switchs of the board connected to input **PORTC** to move up and down the rackets and sw2 (RC2) for the start.

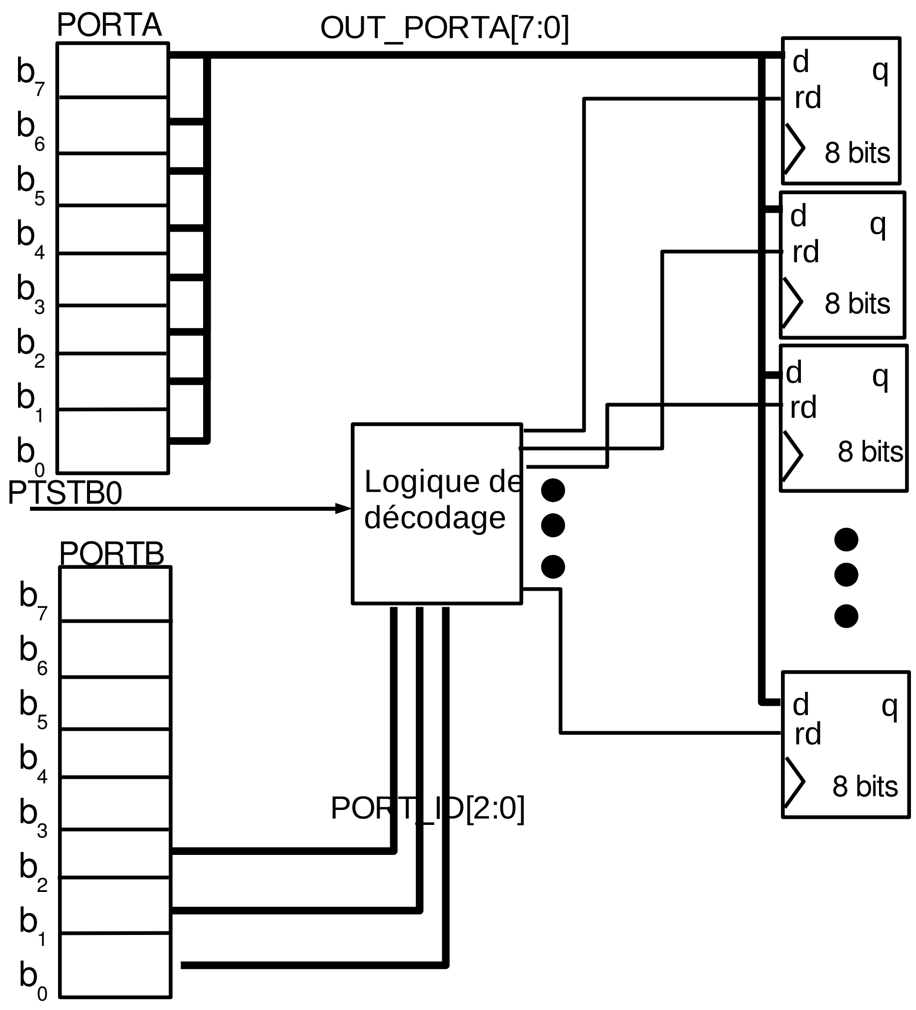
You can also see the global variable `unsigned int posRaqu_16;` (which cannot be declared as a local) which manage the 9 bits of the real position of rackets.

The last problem that needs to be adressed when dealing with harware is the issue of limited number of ports. Strobe signals are very interesting for such a situation. PTSTB0, PTSTB1, and PTSTB2 are the corresponding strobe signals.

Managing eight ports with only two (not checked)

(Since this project was teminated in April 2010, I have successfully checked this hardware but with an other core).

We aim to use **PORTA** and **PORTB** in the following way : the 3 LSB bits of **PORTB** selects one PORT among eight. The strobe signal PTSTB0 is then connected through a demultiplexer till this PORT, allowing a writing in **PORTA** ending in the selected PORT. It's probably more easy to understand with a circuit diagram :



Only four of the eight ports are sketched on the right. After we have to modify all the connections and the corresponding subprograms.

Connections(not checked)

If all the eight ports are numeroted from 0 to 7, we choose

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

y_raq2<7:0>	PORT6<7:0>

C Subprograms (not checked)

The subroutines after the new connections are presented now.

The new "setX"subroutine (not checked)

The new C program listing is given below

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

We stop here the study of the C functions hoping it is enough for a reader to help him to start.

Future Perspectives

Explore if it is possible to avoid the complete compilation of the project every time you change the program (on my eight years old computer it takes 10 minutes to compile all the project). If not, realize a new ROM component which is able to download a program through a serial or parallel port. The initial project contains an example of ROM to complete and to check (see the file SourceCode/xilinx/VHDL_source/Rev2.0/semrmint.vhd). This is a hard work because the corresponding parallel connection is not documented. We only have a DOS executable and its source code (in SourceCode/Download/DOWNLOAD.C).

Manage scores in text mode on the lower part of the VGA screen.

Conclusion

Our undergraduate students encounter few difficulties to realize this project. The bouncing ball was working at half of the project (after 40 hours) for different reasons :

- The project was not prepared by the tutor (author of this report). A problem with the RAM takes more than 20 hours of hard work to the tutor before to be convinced that the problem was effectively with this RAM. How many hours take an undergraduate sudent to solve such a problem ? Probably too many and probably more than 80 hours, the total time of the project.

- This project needs skills on compilation and processor hardware. Fortunatly, today our students still have skills in hardware, but unfortunatly have no skills on compilers. They learn C and Java programming but have no knowledge on where a local variable lie in memory, or how passing parameters works.

I want to mention again that the choice of the softcore core has been done only because of the number of pages of the documentation (194 pages). This was very useful for a beginner like me.

We had to face rapidly the little size of the RAM memory. Even the comparisons in C programming to check if the ball is hitting a racket was not working with the first "wait" subprogram (a simple loop) : not enough RAM. Because we found a solution with the timer we finally didn't make this comparison in the hardware. I was not able to understand exactly how the Hitech C manages local variable, parameters and so on and then how to reduce the size of the RAM used by a program.

When realizing this project, I was obsessed to make a C program running in a FPGA. It was the first time for me. But with hindsight I think it's probably better to program this core with assembly language.

Comparison of the SiliCore1657 with the Picoblaze®

	SiliCore1657	PicoBlaze ®
Licence	Free	IP Core Xilinx (free downloadable)
Source Code	Available (see in the begining of this report)	Available when downloaded by Xilinx but optimized and then not understandable
Size in the FPGA	30% of spartan 3 200k	4% of spartan 3 200k
RAM	72 bytes	64 bytes
Registers	7 spécialisés	16 for general use
PORTS	3 mais peut être étendu	peut être étendu jusqu'à 256
ROM	2048 x 12 bits	1024 x 18 bits
Pile	2 niveaux	31 niveaux
Integrated Development Environment	MPLAB free and well known which works also under Linux	PBlazeIDE (mediatronix) and KPicosim under Linux
Languages	C, assembly language	assembly language
Interruptions	No	you have to manage them yourself. See an interrupt

		manager at opencores.org
Documentation	Very good documented	Very good documented

Both architecture have the same area of applications : little programs and big hardware around.

I intend to use a 16F84 compatible soft processor next year (2010/2011). The begining of this project is already downloadable in French : "http://perso.wanadoo.fr/moutou/ER2/Core16F84.pdf"

and probably very soon available in English :

"http://perso.wanadoo.fr/moutou/ER2/Core16F84_en.pdf"

An other soft core is an ATMEL available at "http://opencores.org/project,cpu_lecture"

The English pre-report is available :

"http://perso.wanadoo.fr/moutou/ER2/CoreAtMega8_en.pdf"

Acknowledgements

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

Writen with OpenOffice 2.4.1 under Linux

APPENDIX I (transforming HEX file into VHDL)

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

FILE *fpi, *fpo;

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

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

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

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

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

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

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

char *MakeBinaryString (unsigned int data)
{
		static char szBinary[20];
		int  i;
		for (i = 0; i < 16; i++) {
```

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

<u>Attention</u> : The VHDL file generated with this program include sometimes an error in the first when : two times a `When to_integer(unsigned(Addr)) = 0000` . It's the case in the only example I gave in this report. I have not check the importance of this issue but I never encounter ROM problems during this project.

Appendix II VHDL module to manage VGA

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

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

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



library IEEE;
use  IEEE.STD_LOGIC_1164.all;
ENTITY VGAtop IS
  PORT (clk_50 : in STD_LOGIC;
              x_rect, y_rect: IN STD_LOGIC_VECTOR(9 DOWNTO 0);
              y_raquG, y_raquD: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
              hsynch,vsynch,red,green,blue : out STD_LOGIC);
END VGAtop;
ARCHITECTURE atop of VGAtop is
COMPONENT VGA_SYNC IS
              PORT(   clock_25Mhz      : IN       STD_LOGIC;
                 horiz_sync_out, vert_sync_out   : OUT    STD_LOGIC;
                 pixel_row, pixel_column: OUT STD_LOGIC_VECTOR(9 DOWNTO 0));
END COMPONENT;
COMPONENT rect IS PORT(
  row,col,x_rec,y_rec,delta_x,delta_y :in STD_LOGIC_VECTOR(9 DOWNTO 0);
  colorRGB : in  STD_LOGIC_VECTOR(2 DOWNTO 0);
  red1,green1,blue1 : out std_logic);
END component;
signal clk_25,sred,sgreen,sblue,sred1,sgreen1,sblue1,sred2,sgreen2,sblue2 : std_logic;
signal srow,scol : STD_LOGIC_VECTOR(9 DOWNTO 0);
begin
  process(clk_50) begin
    if clk_50'event and clk_50='1' then
               clk_25 <= not clk_25;
             end if;
  end process;
  i1:vga_sync port map(clock_25Mhz =>clk_25, horiz_sync_out=>hsynch,
               vert_sync_out=>vsynch, pixel_row=>srow, pixel_column=>scol);
  balle:rect port map(row=>srow, col=>scol, red1=>sred, green1=>sgreen,
                blue1=>sblue,colorRGB=>"111",
                delta_x=>"0000001010",delta_y=>"0000001100",
                x_rec => x_rect, y_rec => y_rect);
  raquetteG:rect port map(row=>srow, col=>scol, red1=>sred1, green1=>sgreen1,
               blue1=>sblue1,colorRGB=>"100",
              delta_x=>"0000001010",delta_y=>"0000111010",
              x_rec => "0000010110", y_rec(8 downto 1) => y_raquG,
            y_rec(9)=>'0',y_rec(0)=>'0');
  raquetteD:rect port map(row=>srow, col=>scol, red1=>sred2, green1=>sgreen2,
                  blue1=>sblue2, colorRGB=>"100",
                  delta_x=>"0000001010",delta_y=>"0000111010",
                  x_rec => "1001001000", y_rec(8 downto 1) => y_raquD,
                  y_rec(9)=>'0',y_rec(0)=>'0');
  red <= sred or sred1 or sred2;
  green <= sgreen or sgreen1 or sgreen2;
  blue <= sblue or sblue1 or sblue2;
end atop;

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

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

APPENDIX III (ucf File)

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

APPENDIX IV (toptoplogic.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;

ENTITY TopTopLogic IS
port (

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


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

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

component xilinx_one_port_ram_sync
   generic(
      ADDR_WIDTH: integer:=7;
      DATA_WIDTH: integer:=8
   );
   port(
      clk: in std_logic;
      we: in std_logic;
      addr: in std_logic_vector(ADDR_WIDTH-1 downto 0);
      din: in std_logic_vector(DATA_WIDTH-1 downto 0);
      dout: out std_logic_vector(DATA_WIDTH-1 downto 0)
    );
end component xilinx_one_port_ram_sync;

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

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

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

  i2:xilinx_one_port_ram_sync PORT MAP(
      clk=>MCLK,
      we=>EWERAM,
      addr=>EADR,
      din=>EALU ,
      dout=>GP    );

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