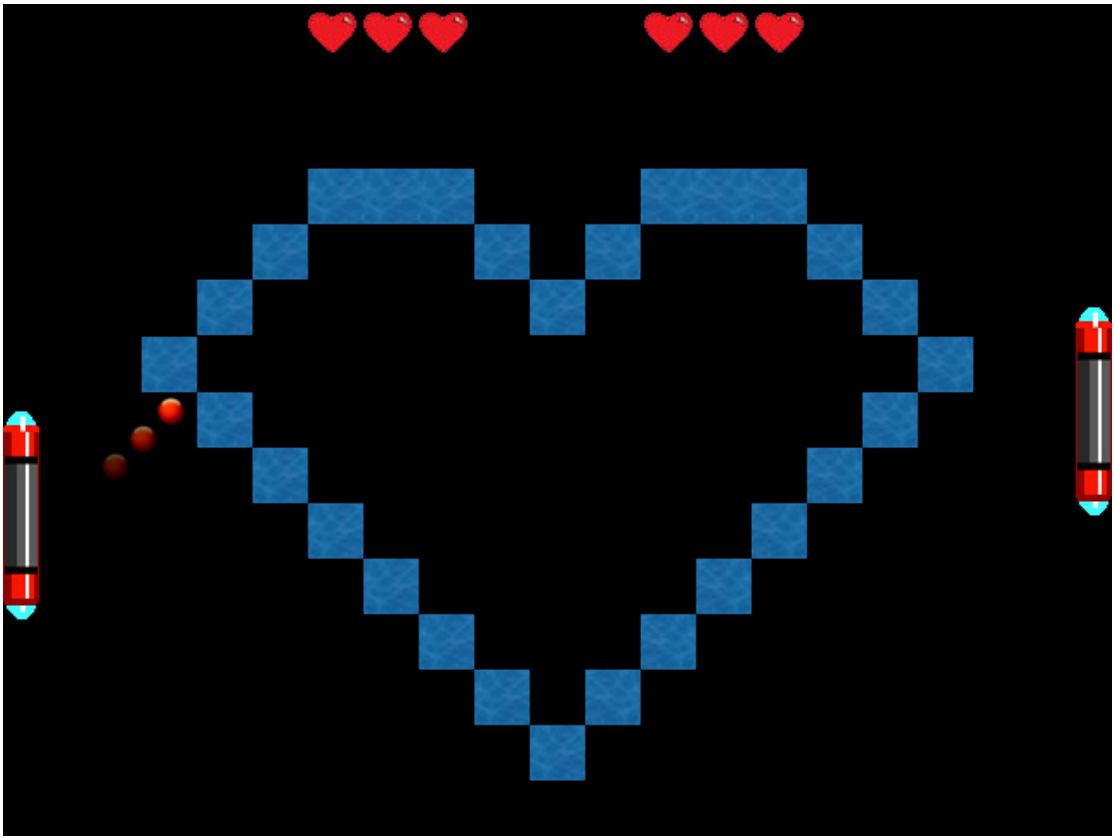
Pong

CSEE 4840 Spring 2012 Project

Bo Li (bl2438) Jihong Zou (jz2428)

Cong Liu (cl2991) Zuyang Cao (zc2220)

May 9, 2011



1 Introduction

Our project goal is to design a Pong video game which is a networked multiplayer version of the 1970s classic Atari video game Pong. It is an entertaining game that involves two players, each with a paddle that hits a ball back and forth around the screen. A player wins a point when s/he manages to put the ball beyond the opponent's paddle.

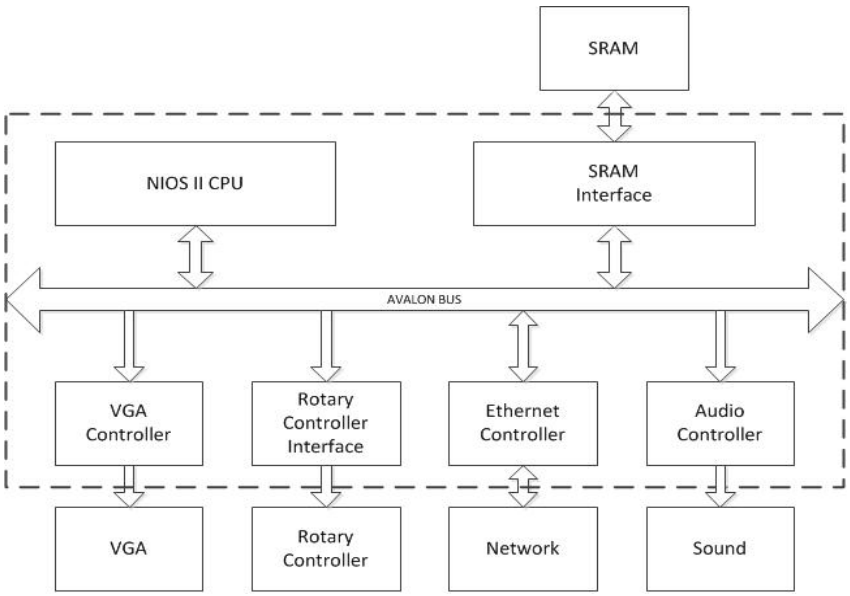
The rule in our design is easy: Both of the players try to kick the moving ball with a paddle, the winner is the one who makes the other lose the ball. Our design is based on the Altera FPGA Board.

In our design, the game always keeps track of the position of the paddles and the game ball. Players move their paddles using a keyboard or rotary controller. One player also can use rotary controller to play with AI. If a ball hits a paddle, it bounces back to the other side of the screen at an angle equal to that at which it hit. Each time the ball hits the paddles or boundaries, there will be clear sound. When the ball is moving, there will be a shadow-like vision effect which makes the game more living. The game will restart until one player loses all 3 life signs. As the time is going, the ball is moving faster so as to increase the difficult of the game.

To increase the animation movement of ball, we add the shadow which is somewhat like the tail of ball on the moving track of the ball. Besides, VGA interrupt is used so as to lower the refreshment rate and thus reduce the breakdown probability of CPU.

2 Overall Architecture

The system is based on the Altera NIOS II CPU and implemented as a series of peripherals, connected using Altera's proprietary Avalon bus and built with the assistance of the SOPC Builder tool. The following is the block diagram of the system.



A brief description of each module is given below:
1 CPU: loads instructions stored in the SRAM and executes them one by one.
2 Audio Controller: receives audio commands from the BUS and translates it to Audio Driver so as to make a sound.
3 Audio Driver: drives the audio decoder with the preloaded sounds.
4 SRAM: stores data and instructions of the NIOS II needs.
5 Ethernet Controller: receives data packages sent from keyboard connected through Ethernet and sends data to the BUS.
6 VGA Controller: receives display data from the BUS and feeds them to the driver.
7 VGA Driver: drives the VGA monitor with the preloaded images.
8 Rotary Controller: produces control signal and send it to the BUS.
In the next few sections, we will discuss the rotary controller, Ethernet, VGA display, audio control and keyboard in detail.

3 Rotary Controller

Rotary controller is the controller of the paddle which can realize the control through different input sequences. When the rotary controller is rotated by one gear, A and B, as two input signals, will produce a sequence listed below. By judging the order of the sequence, we know whether it is clockwise rotation or not. Furthermore, we can correspond to the movement of paddle. The input signals A, B can be connected to DE2 board through GPIO interface. The following table is the sequence of the binary code produced by the rotation of rotary controller.

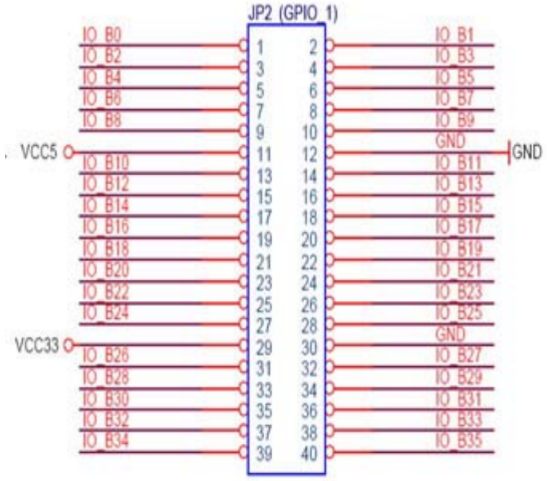
Binary Code for Clockwise Rotation			Binary Code for Counter-clockwise Rotation	
phase	A	B	A	B
1	0	1	1	0
2	1	1	1	1
3	1	0	0	1
Null	0	0	0	0

(Where Null represents the state when no rotation happens)

3.1 Interface Connection

The DE2 Board provides two 40-pin expansion headers. Each header connects directly to 36 pins on the Cyclone II FPGA, and also provides DC +5V (VCC5), DC +3.3V (VCC33), and two GND pins. In this design, GPIO_1[24] and GPIO_1[25] as the input interface of rotary controller, VCC33 as the DC source and GND as the ground base.

The following is the pin assignments of GPIO_1 on DE2 board.

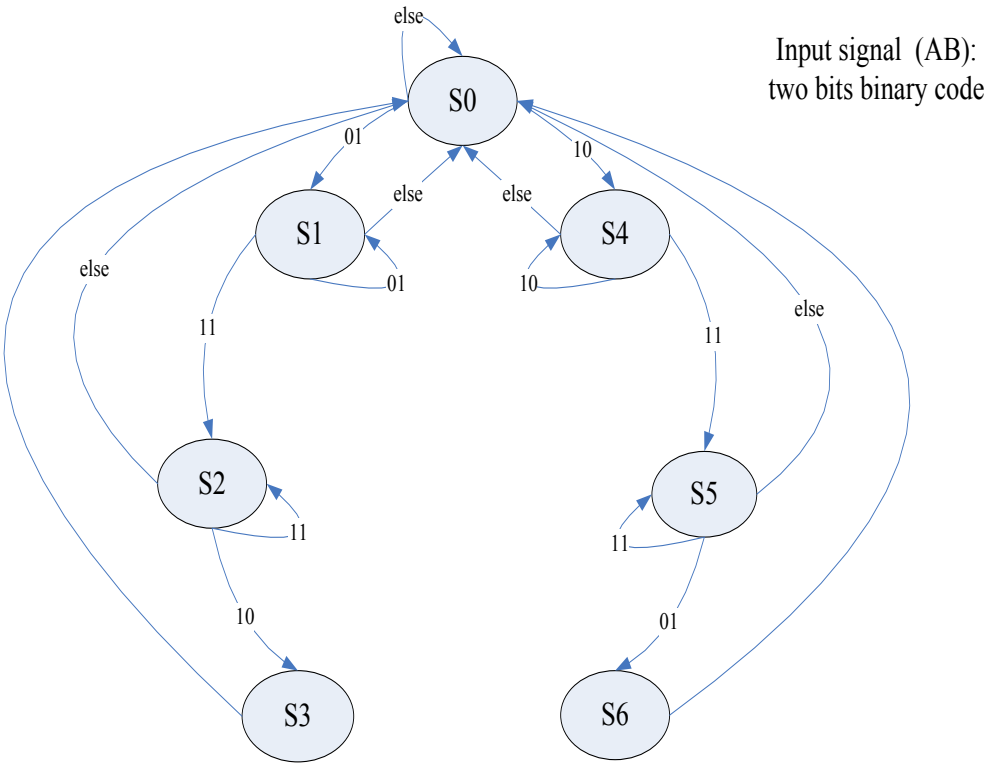


The schematic diagram of the expansion headers is as follows and we only choose the ports we need in this design.

Signal Name	FPGA pin Number	Description
GPIO_1[24]	PIN_U23	GPIO Connection 1[24]
GPIO_1[25]	PIN_U24	GPIO Connection 1[25]

3.2 State Machine

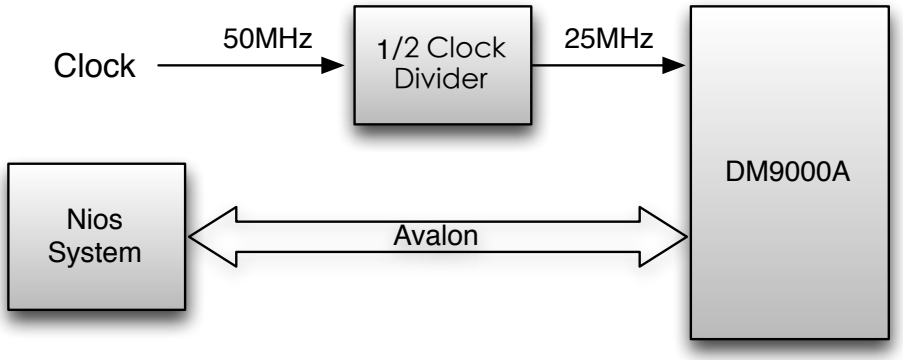
In the part of hardware design for the rotary controller, we use Moore state machine as the basic structure to detect the input signal and send the control command so as to move the paddle up and down. There are 7 states in the state machine marked by S0, S1, S2, S3, S4, S5, S6. When no rotation happens, state machine will stay at state S0. Only when particular sequence appears, i.e. , from S0 to S3 or from S0 to S6, one rotation can then be detected and send the command signal to the BUS. The state machine graph is as follows.



State Machine of Rotary Controller

4 Ethernet

As in Lab2, Altera DE2 board's DM9000A chip which contains a fast 10/100 Mbps transceiver is employed to communication under CSMA/CD protocol. The architecture is shown below.



The Ethernet peripheral is a straightforward bridge between the Avalon bus and the DM9000A's interface, with a few notable features:

1. The address/data bus is not used in a conventional way to access DM9000A configuration registers. The address space is only two words, in fact. The first accesses the configuration register address, the second accesses the register referenced in the first.

2. The Altera DE2 board does not provide a crystal oscillator for the controller. The signal must be generated by the FPGA. Fortunately, it is an even multiple of the clock

frequency so this is not difficult to implement once its necessity is discovered.

3. A power-on reset controller is important for repeatable performance. The DM9000A RST_N line does not map sensibly to an Avalon control signal so it was initially tied high to keep the device out of reset. This led to erratic functionality and eventually a power-on reset delay counter was implemented to keep the device in reset until shortly after system initialization (and, presumably, power supplies have stabilized).

4. The SOPC configuration for this peripheral is non-standard and very important to set correctly. The interface is unclocked, with 20ns setup/hold/read/write latency times specified.

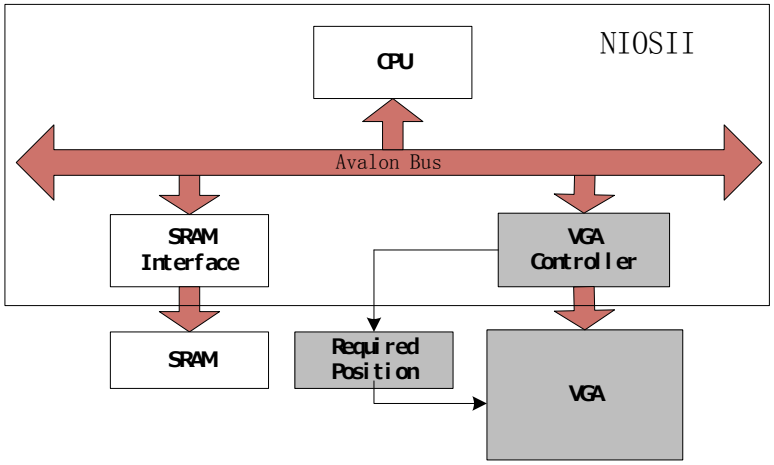
5 VGA Display

The VGA Controller is based on the de2_vga_raster.vhd file provided in Lab 3. The clock which is originally set to 50MHz will be reset to 25MHz for the VGA protocol. Vertical and horizontal blank signals tell the VGA whether or not to draw the default background and sync signals tell the VGA_Raster to draw something besides the default background. Because the background is a static image, the VGA Controller will always show the background in hardware without any communication with the software. The software merely tells the VGA controller where to draw the ball and the paddle.

At the preparation stage, we decide to use 8 bit RGB data of the image, with matlab coding, the RGB data is achieved from the image. Then we rewrite the data into 10 bit form since the RGB data storage in FPGA is 10 bit data.

There are several tricky parts we focus to handle for the display. Since the RGB data of the image is too large to restore as one dimension data array, we choose to restore them as two-dimension data array. Also, we arrange the sprites display in a reasonable order. Moreover, by separating the whole window into 20*15 tiles, we are able to make full use of the resource of the background image. As a result, our game can performance a fancy game scene just with a few small image data array, which realizes the benefit of both the player and the designer. Finally, it is the turn to fix the refresh of the display. We add a port in VGA file for the interrupt signal, and it can slow down the speed of the data writing from the Nios system to the hardware, with this signal the problem of flicker has been fixed.

In conclusion, for the display part, we have met the basic requirement in our proposal, and fix some display bugs. The display can keep in a good and stable state as we expected.



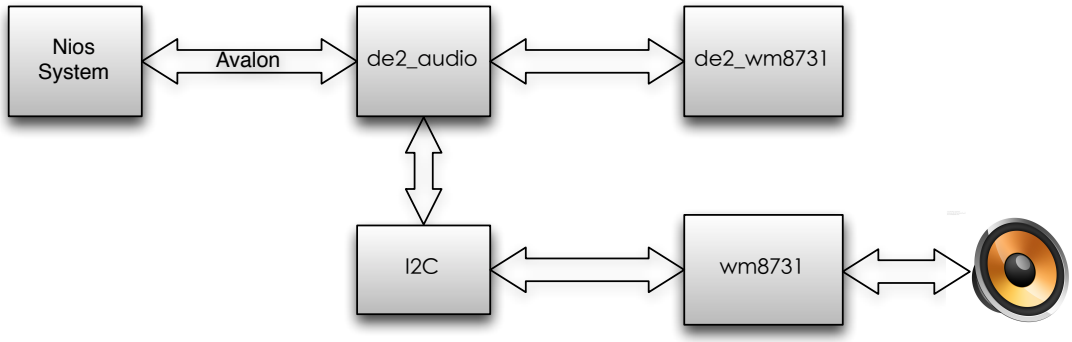
VGA Structure

6 Audio Controller

A sound data, which contains the amplitude information of a record of sound, will be saved into audio controller. Once the ball touches the wall or paddle, NIOS II system will generate a control signal to audio controller. And audio controller will send data to the ADC and generate a sound effect.

The Pong sound file of .wav type was found on the Internet. Then the sound .wav file can be imported into Matlab. After resampling and normalizing by Matlab, sound data can be transferred to the format fit for VHDL code and exported to a .VHD file.

One important thing to have correct sound is to make the sampling rate of sound data and audio controller same. The sampling rate of audio controller can be changed by choose different value of clock_divider.



Audio Block Diagram

7 Keyboard

The keyboard is one of the most important modules in this design. It provides another way to play the game, i.e., one user with rotary controller while another using keyboard. The keyboard is connected to DE2 board through Ethernet wire. In this design, following keys are used and their functions are described below:

key	function
w	Move paddle up
s	Move paddle down
k	Two-players play mode
l	One-Player and AI play mode

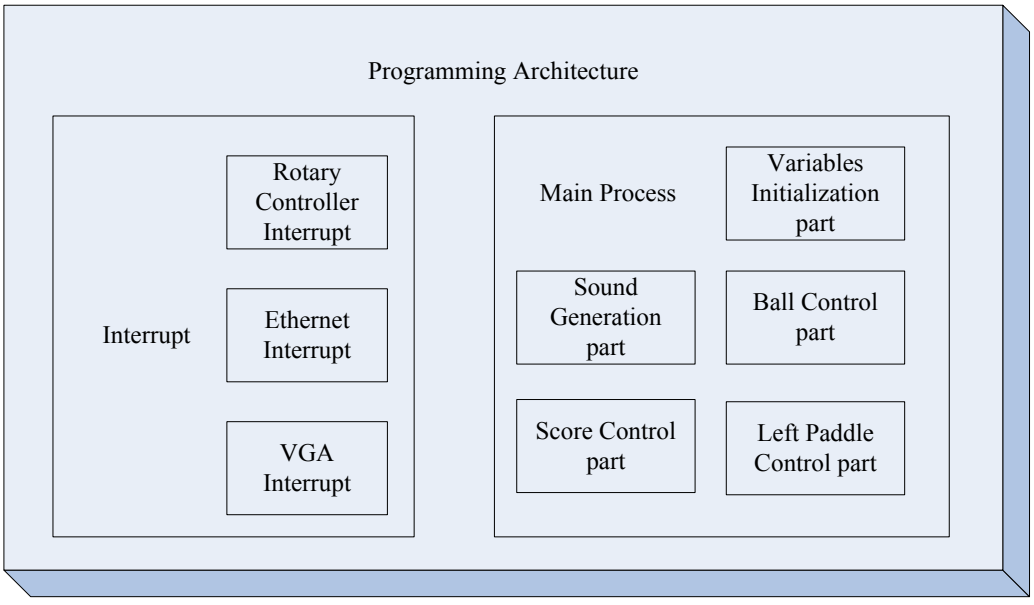
In our design, we add a small display window on the player's computer to denote the mode of the game. When the key is continuously pressed, the corresponding function will be performed repeatedly.

8 Software Design

Software is definitely the most significant part of this design, since all hardware components are functioning according to the commands received from the software program. In our design, software has the following functions:

1. Handles the keyboard input interrupt (Ethernet Interrupt) and translate it into the corresponding movement of the paddle or different mode.
2. Handles the rotary controller input interrupt and translate it into up or down movement of the paddle.
3. Handles the VGA interrupt and refreshes positions of balls and paddles on the VGA display screen.
4. Generates position of ball and paddles.
5. Records the game process and change the difficulty of the game as the game is going on.
6. Generates sound when the ball hits paddles or boundaries.

The programming architecture is shown as follows.



As we can see, in this design, we use two relatively separate parts to build the software system. The interrupt part provides three different entrances when the main process is interrupted by one of three events. Interrupts can do the relevant operations more efficient because It will not take up CPU resources when no events are activated. Particularly for the VGA Interrupt, it not only reduce the refresh rate of the VGA display screen, but also provide an accuracy clock which was originally produced by for loop expressions and wasted lots of CPU time.

9 Design Challenge

The following is a list of some of the significant challenges faced in implementing this design. For a particular challenge we have faced, a summary of solution then presented to aid future development:

1. Ethernet initialization failures.

Solution: Fixed by adding power-on reset controller

2. Ethernet controller unresponsive on bus.

Solution: Fixed by adding clock

3. Breakdown of state machine.

Solution: Fixed by setting unexpected state back to a default state no matter what the input signal is.

4. Breakdown of VGA screen refresh.

Solution: Fixed by putting screen refresh process in a CPU interrupt caused when the whole VGA screen has been scanned one time so as to reduce the refresh rate.

10 Lessons Learned

In this project, we had learned a lot on how to work as a team and how to make a good and clean design. First of all, appropriate design partition is a key for us to work as a team. If the design is not partitioned properly, some team members may be idling while the others are busy working. Working in parallel is important for a design team to make progress. Second, when dealing with hardware design, it is best to spend a significantly longer period of time before attempting to write any code, to plan out every detail of the design which will contribute to later on debug processing. Third, good software data structure is important for implementing complex functions. Poor data structure may be easy to quickly make the program run, but will definitely cause a lot of extra work when implementing complex functions. Fourth, backup source files regularly with a modification date associated will greatly help when they are needed. Fourth, online files sharing tools, like “Dropbox”, help us to organize and share the latest work easily.

11 Contributions

Zuyang Cao is in charge of Audio, Ethernet and keyboard hardware design. Also he finishes software and hardware interface such as building the NIOS system in SOPC builder.

Jihong Zou and Cong Liu work mainly together on the VGA display hardware design, which is the most complex part of this project. Besides, they make contributions on other parts of hardware design.

Bo Li is in charge of hardware design of rotary controller, and also the principal designer of software design.

12 Source Code

```
/*************************************************/
// hellow_world.c
/*************************************************/
#include <io.h>
#include <system.h>
#include <stdio.h>
#include "DM9000A.h"

#define IOWR_VGA_DATA(base, offset, data) \
      IOWR_16DIRECT(base, (offset) * 2, data)
#define IORD_VGA_DATA(base, offset) \
      IORD_16DIRECT(base, (offset) * 2)
#define ROUND_NUMBER 3
#define MAX_MSG_LENGTH 128

// Ethernet MAC address.    Choose the last three bytes yourself
unsigned char mac_address[6] = { 0x01, 0x60, 0x6E, 0x11, 0x02, 0x0A    };
unsigned int    receive_buffer_length;
unsigned char receive_buffer[1600];

int counter=0;
float x_ball=312;// default x location of ball
float y_ball=232;// default y location of ball
int x_ball1=60;// x location of ball_1
int y_ball1=60;// y location of ball_1
int x_ball2=55;// x location of ball_2
int y_ball2=55;// y location of ball_2

float delta_x=1;// delta x of ball
```

```
float delta_y=1;// delta y of ball

int l3=2,l2=2,l1=2;//left pad score :from left to right is l1(address is 6), l2(address is 7), and l3(address is 8)
int r3=2,r2=2,r1=2;// right pad score :from right to left is r1(address is 9), r2(address is 10), and r3(address is 11)

int k; //delay counter
int l; //delay counter
int x_lpad=0;//default x location of left paddle
int y_lpad=150;//default y location of left paddle
int t=1;// delta x of left paddle
int o=1;// delta y of left paddle
int x_rpad=619;//x location of right paddle
int y_rpad=150;//y location of right paddle
int deltax_rpad=1;// delta x of right paddle
int deltay_rpad=1;// delta y of right paddle
int score_lpad=3;// score of left paddle
int score_rpad=3;// score of right paddle

int loop_length[]={1200,1000,800};// speed of the ball
int score_rctrl=0; // the score of the rotary controller
int score_ai=0; // the score of AI
int level_stage=2;//default is 1, the total levels will be 3
int winning_flag=0; //default is 0, when 1, rotary controller wins.otherwise, AI wins.
int step_level=3;//default is 1(level 1), total number of levels are 3
int *pt_loop;
int divcounter=0;// change the frequency of the refreshment of left padder
int counter_map[]={64,16,2};
int *pt_cntmap;
int temp_x_lpad;
int temp_y_lpad;
int temp_x_rpad;
int temp_y_rpad;
int update_flag=1;
char ai_control = 1;

//Ethernet Interrupt
static void ethernet_interrupt_handler()
{
   unsigned int receive_status;

   receive_status = ReceivePacket(receive_buffer, &receive_buffer_length);
```

```
    if (receive_status == DMFE_SUCCESS)
    {
        printf("0x%.2X,", receive_buffer[42]);

        //Up
        if(receive_buffer[42] == 0x77)
        {
                if(y_lpad>0)
                {
                        y_lpad=y_lpad-10;
                }
         }

        //Down
        if(receive_buffer[42] == 0x73)
        {
                if(y_lpad < 360)
                {
                        y_lpad=y_lpad+10;
                }
        }

        //AI ON
        if(receive_buffer[42] == 0x6B)
        {
                ai_control = 1;
        }

        //AI OFF
        if(receive_buffer[42] == 0x6C)
        {
                ai_control = 0;
        }
    }
    /* Clear the DM9000A ISR: PRS, PTS, ROS, ROOS 4 bits, by RW/C1 */
    dm9000a_iow(ISR, 0x3F);
    /* Re-enable DM9000A interrupts */
    dm9000a_iow(IMR, INTR_set);
}

//VGA Interrupt
static void interrupt_vga (void * context, alt_u32 id)
{
//IOWR_16DIRECT(VGA_BASE, 0, 0); // reset request
```

```
IOWR_VGA_DATA(VGA_BASE, 0,    x_ball);    // ball h
IOWR_VGA_DATA(VGA_BASE, 1,    y_ball);    // ball v
IOWR_VGA_DATA(VGA_BASE, 12, x_ball1);    // ball1 h
IOWR_VGA_DATA(VGA_BASE, 13, y_ball1);    // ball1 v
IOWR_VGA_DATA(VGA_BASE, 14, x_ball2);    // ball2 h
IOWR_VGA_DATA(VGA_BASE, 15, y_ball2);    // ball2 v
IOWR_VGA_DATA(VGA_BASE, 2, x_lpad);    // plot x_lpad
IOWR_VGA_DATA(VGA_BASE, 3, y_lpad);    // plot y_lpad
IOWR_VGA_DATA(VGA_BASE, 4, x_rpad);    // plot x_rpad
IOWR_VGA_DATA(VGA_BASE, 5, y_rpad);    // plot y_rpad
//Heart Display Control
IOWR_VGA_DATA(VGA_BASE, 6, l1);    // ----left lose one score
IOWR_VGA_DATA(VGA_BASE, 7, l2);    // ----left lose one score
IOWR_VGA_DATA(VGA_BASE, 8, l3);    // ----left lose one score
IOWR_VGA_DATA(VGA_BASE, 9, r1);    // ----right lose one score
IOWR_VGA_DATA(VGA_BASE, 10,r2);    // ----right lose one score
IOWR_VGA_DATA(VGA_BASE, 11,r3);    // ----right lose one score

//Heart Display Control
            if(score_lpad == 0)
            {
                  l1 = 3;l2 = 3;l3 = 3;
            }
            else if(score_lpad == 1)
            {
                  l1 = 3;l2 = 3;l3 = 2;
            }
            else if(score_lpad == 2)
            {
                  l1 = 3;l2 = 2;l3 = 2;
            }
            else
            {
                  l1 = 2;l2 = 2;l3 = 2;
            }

            //Heart Display Control
            if(score_rpad == 0)
            {
                  r1 = 3;r2 = 3;r3 = 3;
            }
            else if(score_rpad == 1)
            {
                  r1 = 3;r2 = 3;r3 = 2;
```

```
                }
                else if(score_rpad == 2)
                {
                        r1 = 3;r2 = 2;r3 = 2;
                }
                else
                {
                        r1 = 2;r2 = 2;r3 = 2;
                }

}

//Rotary Controller Interrupt
static void interrupt_rotary (void * context, alt_u32 id)
{
        int flag_right_left;
        counter ++;
        printf ("%d\n",counter);
        flag_right_left=IORD_16DIRECT(ROTARY_BASE, 0);
        //Rotary Left
        if(flag_right_left==1)
        {
                if(y_rpad>0)
                {
                     y_rpad=y_rpad-10;
                }
                printf ("Left\n");
          }
        //Rotary Right
        if(flag_right_left==2)
        {
                if(y_rpad<360)
                {

                y_rpad=y_rpad+10;
                }
                printf ("Right\n");
        }
        // reset the interrupt request
        IOWR_16DIRECT(ROTARY_BASE, 0, 0);
}

//Make one sound
void sound()
```

```
{
	IOWR_16DIRECT(AUDIO_BASE, 0, 0);
	for(k=0;k<60;k++){}
	IORD_16DIRECT(AUDIO_BASE, 0);
}

int main()
{

	alt_irq_register(VGA_IRQ, NULL, (void*)interrupt_vga); //register the irq
	// Initalize the DM9000 and the Ethernet interrupt handler
	DM9000_init(mac_address);
	alt_irq_register(DM9000A_IRQ, NULL, (void*)ethernet_interrupt_handler);

	pt_loop=loop_length;// default is assigning the address of loop_length[0] to p_loop
	pt_cntmap=counter_map; //default *pt_cnt=counter_map[0]
	alt_irq_register( ROTARY_IRQ, NULL,(void*)interrupt_rotary ); // register IRQSOURCE

	for(;;)
	{
		for(k=0;k<*pt_loop;k++)
		{;}//delay

		temp_y_lpad=y_lpad;
		temp_y_rpad=y_rpad;
		x_ball=x_ball+delta_x;
		y_ball=y_ball+delta_y;

		switch(level_stage)
		{
			case 1:pt_cntmap=counter_map;
				break;

			case 2:pt_cntmap=counter_map+1;
				break;

			case 3:pt_cntmap=counter_map+2;
				break;

			default:
				break;
		}
```

```
/*****************************************************************
**********/
 // Score Control

/*****************************************************************
**********/
        if(score_lpad == 0 | score_rpad == 0)
        {
            score_lpad = 3;
            score_rpad = 3;
            if(level_stage==3)
             {
                level_stage=1;
                pt_loop=loop_length;
            }
            else
            {
                level_stage++;
                pt_loop++;
            }
        }


/*****************************************************************
**********/
 // AI:left padder control part

/*****************************************************************
**********/
        if((divcounter = *pt_cntmap) && (ai_control == 1))
        {
            if(delta_x<0)
            {
                if(y_lpad>=y_ball+8)
                {
                    if(y_lpad<=0)
                    {
                        y_lpad=0;
                    }
                    else if(y_lpad>=360)
                    {
                        y_lpad=360;
                    }
```

```
                        else
                        {
                            y_lpad=y_lpad-step_level;
                        }
                    }
                    else if(y_lpad<=y_ball-120)
                    {
                        if(y_lpad<=0)
                        {
                            y_lpad=0;
                        }
                        else if(y_lpad>=360)
                        {
                            y_lpad=360;
                        }
                        else
                        {
                            y_lpad=y_lpad+step_level;
                        }
                    }
                    else
                        y_lpad=y_lpad;
                    divcounter=0;
                }
            }
            else
            {
                divcounter++;
            }

/*****************************************************************
*****************/
    if( x_ball<=21)
    {
        if(y_ball>=y_lpad & y_ball<=(y_lpad+120))
        {
            if(y_ball >= (y_lpad + 48) & y_ball<= (y_lpad + 72))
            {
                delta_x=1.3;
            }
            else if (y_ball >= (y_lpad + 24) & y_ball<= (y_lpad + 96))
            {
                delta_x=1;
            }
```

```
            else
            {
                  delta_x=0.7;
            }

            sound();
      }
      else
      {
            x_ball=312;
            y_ball=232;
            x_ball1=312;
            y_ball1=232;
            x_ball2=312;
            y_ball2=232;
            delta_x=-1;

            IOWR_VGA_DATA(VGA_BASE, 0, x_ball);    // ball h
            IOWR_VGA_DATA(VGA_BASE, 1, y_ball);    // ball v
            score_lpad--;

            for(k=0;k<5000;k++)
            {
                  for(l=0;l<200;l++)
                  {;}//delay
            }
      }
}

if( x_ball>=603)
{
      if(y_ball>=y_rpad & y_ball<=(y_rpad+120))
      {
            if(y_ball >= (y_rpad + 48) & y_ball<= (y_rpad + 72))
            {
                  delta_x=-1.3;
            }
            else if (y_ball >= (y_rpad + 24) & y_ball<= (y_rpad + 96))
            {
                  delta_x=-1;
            }
            else
            {
                  delta_x=-0.7;
```

```
            }
            sound();
        }
        else
        {
            x_ball=312;
            y_ball=232;
            x_ball1=312;
            y_ball1=232;
            x_ball2=312;
            y_ball2=232;
            delta_x=1;
            IOWR_VGA_DATA(VGA_BASE, 0, x_ball);    // ball h
            IOWR_VGA_DATA(VGA_BASE, 1, y_ball);    // ball v
            score_rpad--;

            for(k=0;k<5000;k++)
            {
                for(l=0;l<200;l++)
                {;}//delay//delay
            }
        }
}

//Up and down edge of Screen
if(y_ball==0)
{
    delta_y=1;
    sound();
}
if(y_ball==464)
{
    delta_y=-1;
    sound();
}

//shadow of ball control
if( delta_x > 0 & delta_y > 0 )
{
    x_ball1=x_ball- 5;// x location of ball_1
    y_ball1=y_ball- 5;// y location of ball_1
    x_ball2=x_ball-10;// x location of ball_2
    y_ball2=y_ball-10;// y location of ball_2
}
```

```
        else if( delta_x > 0 & delta_y < 0 )
        {
              x_ball1=x_ball- 5;// x location of ball_1
              y_ball1=y_ball+ 5;// y location of ball_1
              x_ball2=x_ball-10;// x location of ball_2
              y_ball2=y_ball+10;// y location of ball_2
        }
        else if( delta_x < 0 & delta_y > 0 )
        {
              x_ball1=x_ball+ 5;// x location of ball_1
              y_ball1=y_ball- 5;// y location of ball_1
              x_ball2=x_ball+10;// x location of ball_2
              y_ball2=y_ball-10;// y location of ball_2
        }
        else
        {
              x_ball1=x_ball+ 5;// x location of ball_1
              y_ball1=y_ball+ 5;// y location of ball_1
              x_ball2=x_ball+10;// x location of ball_2
              y_ball2=y_ball+10;// y location of ball_2
        }

  //      printf ("ball h : %d \n", IORD_VGA_DATA(VGA_BASE, 2));
  //      printf ("ball v : %d \n", IORD_VGA_DATA(VGA_BASE, 3));
  //      printf ("score of AI : %d\n ",score_ai);
  //      printf ("score of ROTARY : %d\n", score_rctrl);
  //      printf ("who wins(1 or 0): %d %d\n", winning_flag,level_stage);
  //      printf ("game round number: %d\n", ROUND_NUMBER);
    }
    return 0;
}

/************************************************/
// Project.vhd
/************************************************/
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Project is

    port (
        signal CLOCK_50 : in std_logic;                         -- 50 MHz clock
        signal key : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
```

```
        signal LEDR : out std_logic_vector(17 downto 0); -- Red LEDs
        signal GPIO_1      : inout std_logic_vector(35 downto 0);
        SRAM_DQ : inout std_logic_vector(15 downto 0); -- Data bus 16 Bits
        SRAM_ADDR : out std_logic_vector(17 downto 0); -- Address bus 18 Bits
        SRAM_UB_N,                                              --  High-byte  Data
Mask
        SRAM_LB_N,                                              --  Low-byte  Data
Mask
        SRAM_WE_N,                                             -- Write Enable
        SRAM_CE_N,                                            -- Chip Enable
        SRAM_OE_N : out std_logic;                           -- Output Enable

        VGA_CLK,                                                  -- Clock
        VGA_HS,                                                   -- H_SYNC
        VGA_VS,                                                   -- V_SYNC
        VGA_BLANK,                                                -- BLANK
        VGA_SYNC : out std_logic;                              -- SYNC
        VGA_R,                                                    -- Red[9:0]
        VGA_G,                                                    -- Green[9:0]
        VGA_B : out std_logic_vector(9 downto 0);             -- Blue[9:0]

        AUD_ADCLRCK : inout std_logic;                         -- ADC LR Clock
        AUD_ADCDAT : in std_logic;                             -- ADC Data
        AUD_DACLRCK : inout std_logic;                         -- DAC LR Clock
        AUD_DACDAT : out std_logic;                            -- DAC Data
        AUD_BCLK : inout std_logic;                           -- Bit-Stream Clock
        AUD_XCK : out std_logic;                               -- Chip Clock

        ENET_DATA : inout std_logic_vector(15 downto 0);       -- DATA bus 16Bits
        ENET_CMD,                -- Command/Data Select, 0 = Command, 1 = Data
        ENET_CS_N,                                             -- Chip Select
        ENET_WR_N,                                              -- Write
        ENET_RD_N,                                             -- Read
        ENET_RST_N,                                            -- Reset
        ENET_CLK : out std_logic;                             -- Clock 25 MHz
        ENET_INT : in std_logic;                              -- Interrupt

        I2C_SDAT : inout std_logic; -- I2C Data
        I2C_SCLK : out std_logic      -- I2C Clock
        );

end Project;

architecture rtl of Project is
```

```
    signal counter : unsigned(15 downto 0);
    signal reset_n : std_logic;
    signal clk25 : std_logic;

begin

    process (CLOCK_50)
    begin
        if rising_edge(CLOCK_50) then
            if counter = x"ffff" then
                reset_n <= '1';
            else
                reset_n <= '0';
                counter <= counter + 1;
            end if;
        end if;
    end process;

    process (CLOCK_50)
    begin
      if rising_edge(CLOCK_50) then
            clk25 <= not clk25;
      end if;
    end process;
    ENET_CLK <= clk25;

    nios : entity work.nios_system port map (
        clk                                     => CLOCK_50,
        reset_n                                 => reset_n,

        key_to_the_rotary                       => key,
        GPIO_1_to_and_from_the_rotary    => GPIO_1,
        leds_from_the_rotary                    => LEDR,
        -- the_sram
        SRAM_ADDR_from_the_sram          => SRAM_ADDR,
        SRAM_CE_N_from_the_sram          => SRAM_CE_N,
        SRAM_DQ_to_and_from_the_sram => SRAM_DQ,
        SRAM_LB_N_from_the_sram          => SRAM_LB_N,
        SRAM_OE_N_from_the_sram          => SRAM_OE_N,
        SRAM_UB_N_from_the_sram          => SRAM_UB_N,
        SRAM_WE_N_from_the_sram          => SRAM_WE_N,

        VGA_BLANK_from_the_vga => VGA_BLANK,
```

```
        VGA_B_from_the_vga => VGA_B,
        VGA_CLK_from_the_vga => VGA_CLK,
        VGA_G_from_the_vga => VGA_G,
        VGA_HS_from_the_vga => VGA_HS,
        VGA_R_from_the_vga => VGA_R,
        VGA_SYNC_from_the_vga => VGA_SYNC,
        VGA_VS_from_the_vga => VGA_VS,
        -- the_DM9000A
        ENET_CMD_from_the_DM9000A => ENET_CMD,
        ENET_CS_N_from_the_DM9000A => ENET_CS_N,
        ENET_DATA_to_and_from_the_DM9000A => ENET_DATA,
        ENET_INT_to_the_DM9000A => ENET_INT,
        ENET_RD_N_from_the_DM9000A => ENET_RD_N,
        ENET_RST_N_from_the_DM9000A => ENET_RST_N,
        ENET_WR_N_from_the_DM9000A => ENET_WR_N,
        -- the_audio
       AUD_ADCDAT_to_the_audio => AUD_ADCDAT,
       AUD_ADCLRCK_to_and_from_the_audio => AUD_ADCLRCK,
       AUD_BCLK_to_and_from_the_audio => AUD_BCLK,
       AUD_DACDAT_from_the_audio => AUD_DACDAT,
       AUD_DACLRCK_to_and_from_the_audio => AUD_DACLRCK,
       AUD_XCK_from_the_audio => AUD_XCK,
       I2C_SCLK_from_the_audio => I2C_SCLK,
       I2C_SDAT_to_and_from_the_audio => I2C_SDAT

        );

end rtl;

/*************************************************/
// rotary_top.vhd
/*************************************************/
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity rotary_top is port (
   clk, reset_n, chipselect, read, write, address : in std_logic;
   readdata    : out std_logic_vector(15 downto 0);
   writedata : in    std_logic_vector(15 downto 0);

   irq            : out std_logic;
   GPIO_1         : inout std_logic_vector(35 downto 0);
   key              : in    unsigned( 3 downto 0);
```

```
   leds             : out std_logic_vector(17 downto 0)
   );
end rotary_top;

architecture rtl of rotary_top is

component rotary_controller is
      port(
            CLOCK_50 : in     std_logic;
            GPIO_1      : inout std_logic_vector(35 downto 0);
            ledr         : out std_logic_vector(17 downto 0);
            r_output : out unsigned (1 downto 0)
      );
end component rotary_controller;

   signal ram : unsigned (1 downto 0) := "00";
   signal temp : std_logic := '0';
   signal data        : std_logic_vector(15 downto 0);
   signal counter : unsigned(25 downto 0);

   begin

C1 : rotary_controller
       port map(
            CLOCK_50 => clk,
            GPIO_1      => GPIO_1,
            ledr         => leds,
            r_output => ram
       );

   process (clk)
   begin
     if(ram = "10") then
            readdata <= X"0002";
     elsif(ram = "01") then
            readdata <= X"0001";
     end if;

       if rising_edge(clk) then
            if reset_n = '0' then
                  irq <= '0';
            else
                  if write = '1' and chipselect = '1' then
                  irq <= '0';    -- important to reset the irq
```

```
                elsif ((ram = "10" or ram = "01") and (temp = '0')) then
                irq    <= '1';
                temp <= '1';
                elsif ram = "00" then
                temp <= '0';
                --irq <= '0';
                  end if;
          end if;
     end if;
end process;
end rtl;

/************************************************/
// rotary_controller.vhd
/************************************************/
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity rotary_controller is

     port(
          CLOCK_50 : in     std_logic;
          GPIO_1      : inout std_logic_vector(35 downto 0);
          ledr         : out std_logic_vector(17 downto 0);
          r_output : out unsigned (1 downto 0)
     );

end entity;

architecture testing of rotary_controller is

     type state_type is (s0, s1, s2, s3, s4, s5, s6);

     signal state                : state_type := s0;
     signal temp                   : std_logic_vector(7 downto 0) := "00000000";--value of
clk_count determines the duration of high level voltage
     signal rc_input           : std_logic_vector(1 downto 0);
      signal clk_count         : integer range 0 to 10000:=0;
      signal output_flag      : std_logic_vector(1 downto 0):="00";
      signal output_buffer : unsigned (1 downto 0):="00";

begin
```

```
 rc_input <=not GPIO_1(25 downto 24);
r_output <= output_buffer;
 ledr(7 downto 0) <= temp;

p1: process (CLOCK_50)
    begin
        if (rising_edge(CLOCK_50)) then
            case state is
                when s0=>
                    if rc_input = "01" then
                        state <= s1;
                    elsif rc_input= "10" then
                        state <= s4;
                    else
                         state <= s0;
                    end if;
                when s1=>
                    if rc_input = "11" then
                        state <= s2;
                    elsif rc_input = "01" then
                         state <= s1;
                     else
                        state <= s0;
                    end if;
                when s2=>
                    if rc_input = "10" then
                        state <= s3;
                    elsif rc_input = "11" then
                         state <= s2;
                     else
                        state <= s0;
                    end if;
                when s3 =>
                    --if rc_input = "10" then
                    --    state <= s3;
                        --else
                          state <= s0;
                    --end if;
                when s4 =>
                    if rc_input = "11" then
                        state <= s5;
                    elsif rc_input = "10" then
                         state <= s4;
                    else
```

```
                        state <= s0;
                    end if;
                when s5 =>
                    if rc_input = "01" then
                        state <= s6;

                    elsif rc_input = "11" then
                        state <= s5;
                    else
                        state <= s0;
                    end if;
                when others =>
                        state <= s0;
                end case;
        end if;
    end process p1;

p2:  process (CLOCK_50)
    begin
    if rising_edge(CLOCK_50) then
        case state is
            when s0 =>
                temp <= "00000001";
            when s1 =>
                temp <= "00000010";
            when s2 =>
                temp <= "00000100";
            when s3 =>
                temp <= "00001000";
                output_flag <= "10";        --clockwise rotation
            when s4 =>
                temp <= "00010000";
            when s5 =>
                temp <= "00100000";
            when others =>
                temp <= "01000000";
                output_flag <= "01";        --counter-clockwise rotation
        end case;

          if output_flag ="10" then
          clk_count <= clk_count+1;
          output_buffer <="10";

          elsif output_flag ="01" then
```

```
            clk_count <= clk_count+1;
            output_buffer <= "01";

            elsif output_flag ="00" then
            output_buffer <="00";
            clk_count <=0;

            end if;

            if clk_count=10000 then
               output_flag <="00";
               clk_count<=0;
            end if;
      end if;

end process p2;
end testing;

/*************************************************/
// de2_audio.vhd
/*************************************************/
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity de2_audio is

   port (
clk             : in    std_logic;
reset_n       : in    std_logic;
read            : in    std_logic;
write           : in    std_logic;
chipselect : in    std_logic;
address       : in    unsigned(4 downto 0);
readdata     : out unsigned(15 downto 0);
writedata    : in    unsigned(15 downto 0);


      --CLOCK_50  :in  std_logic;                                          -- 50
MHz

      I2C_SDAT : inout std_logic; -- I2C Data
      I2C_SCLK : out std_logic;      -- I2C Clock
```

```
    AUD_ADCLRCK : inout std_logic;                    -- ADC LR Clock
    AUD_ADCDAT : in std_logic;                       -- ADC Data
    AUD_DACLRCK : inout std_logic;                   -- DAC LR Clock
    AUD_DACDAT : out std_logic;                      -- DAC Data
    AUD_BCLK : inout std_logic;                      -- Bit-Stream Clock
    AUD_XCK : out std_logic                          -- Chip Clock


    );

end de2_audio;

architecture datapath of de2_audio is

  component de2_wm8731_audio is
   port (
    clk : in std_logic;                    --    Audio CODEC Chip Clock AUD_XCK
    reset_n : in std_logic;
    test_mode : in std_logic;              --    Audio CODEC controller test mode
    audio_request : out std_logic;         --    Audio controller request new data
    data : in std_logic_vector(15 downto 0);
    signal_start : in std_logic;

    -- Audio interface signals
    AUD_ADCLRCK   : out std_logic;         --    Audio CODEC ADC LR Clock
    AUD_ADCDAT    : in   std_logic;        --    Audio CODEC ADC Data
    AUD_DACLRCK   : out std_logic;         --    Audio CODEC DAC LR Clock
    AUD_DACDAT    : out std_logic;         --    Audio CODEC DAC Data
    AUD_BCLK      : inout std_logic        --    Audio CODEC Bit-Stream Clock
  );
  end component;

  component de2_i2c_av_config is
  port (
    iCLK : in std_logic;
    iRST_N : in std_logic;
    I2C_SCLK : out std_logic;
    I2C_SDAT : inout std_logic
  );
  end component;

  signal audio_clock : unsigned(1 downto 0) := "00";
  signal audio_request : std_logic;
```

```
    signal temp_signal_start : std_logic;
    signal ram : unsigned(15 downto 0);--useless, just for test

begin

    process (clk)
    begin
        if rising_edge(clk) then
             audio_clock <= audio_clock + "1";
            if write = '1' and chipselect = '1' then
                  temp_signal_start <= '1';
                  ram <= writedata;--useless, just for test
            elsif read = '1' and chipselect = '1' then
                  temp_signal_start <= '0';
                  readdata <= ram;--useless, just for test
             end if;
        end if;
    end process;

    AUD_XCK <= audio_clock(1);

    i2c : de2_i2c_av_config port map (
        iCLK         => clk,
        iRST_n       => '1',
        I2C_SCLK => I2C_SCLK,
        I2C_SDAT => I2C_SDAT
    );

    V1: de2_wm8731_audio port map (
        clk => audio_clock(1),
        reset_n => '1',
        test_mode => '1',                              -- Output a sine wave
        audio_request => audio_request,
        data => "0000000000000000",
        signal_start => temp_signal_start,

        -- Audio interface signals
        AUD_ADCLRCK    => AUD_ADCLRCK,
        AUD_ADCDAT     => AUD_ADCDAT,
        AUD_DACLRCK    => AUD_DACLRCK,
        AUD_DACDAT     => AUD_DACDAT,
        AUD_BCLK       => AUD_BCLK
    );
end datapath;
```

```
/*************************************************/
// de2_audio.vhd
/*************************************************/
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity de2_vga_raster is

   port (
      reset : in std_logic;
      clk : in std_logic;                    -- Should be 25.125 MHz

      VGA_CLK,                                    -- Clock
      VGA_HS,                                     -- H_SYNC
      VGA_VS,                                     -- V_SYNC
      VGA_BLANK,                                  -- BLANK
      VGA_SYNC : out std_logic;              -- SYNC
      VGA_R,                                      -- Red[9:0]
      VGA_G,                                      -- Green[9:0]
      VGA_B : out std_logic_vector(9 downto 0); -- Blue[9:0]
      chipselect: in std_logic;
      write : in std_logic;
      read    : in std_logic;
      address:in unsigned(4 downto 0);
      readdata: out unsigned(15 downto 0);
      writedata:in unsigned(15 downto 0);
      byteenable: in unsigned(1 downto 0);
      irq: out std_logic
);

end de2_vga_raster;

architecture rtl of de2_vga_raster is

   -- Video parameters

   constant HTOTAL          : integer := 800;
   constant HSYNC           : integer := 96;
   constant HBACK_PORCH     : integer := 48;
   constant HACTIVE         : integer := 640;
   constant HFRONT_PORCH : integer := 16;

   constant VTOTAL          : integer := 525;
```

```
    constant VSYNC              : integer := 2;
    constant VBACK_PORCH    : integer := 33;
    constant VACTIVE          : integer := 480;
    constant VFRONT_PORCH : integer := 10;

    -- Signals for the video controller
    signal Hcount : unsigned(9 downto 0);    -- Horizontal position (0-800)
    signal Vcount : unsigned(9 downto 0);    -- Vertical position (0-524)
    signal EndOfLine, EndOfField : std_logic;

    signal
RGB,RGB_G,RGB_B,RGB_r,RGB_G_r,RGB_B_r,RGB_bg,RGB_G_bg,RGB_B_bg           :
std_logic_vector (9 downto 0);

    signal background_x, background_y: integer;--back ground;
    --signal score_x,score_y: integer;

    signal vga_hblank, vga_hsync,
            vga_vblank, vga_vsync : std_logic;    -- Sync. signals
    signal  l1,l2,l3,r1,r2,r3:integer:=2;--l1,l2,l3:left pad lose score; r1,r2,r3: right pad
lose score;

    signal          rectangle_h_ball,          rectangle_v_ball,          rectangle_h_ball1,
rectangle_v_ball1,rectangle_h_ball2,          rectangle_v_ball2,rectangle_h_paddle,
rectangle_v_paddle ,
            rectangle_ball, rectangle_ball1, rectangle_ball2, rectangle_paddle,
            rectangle_h_paddle_2,          rectangle_v_paddle_2,rectangle_paddle_2:
std_logic;    -- rectangle area
    signal clk25 : std_logic:='0';
    signal RECTANGLE_HSTART: unsigned(15 downto 0):=x"0100" ;
    signal RECTANGLE_VSTART : unsigned(15 downto 0):=x"0000" ;
    signal RECTANGLE_HSTART_ball1: unsigned(15 downto 0):=x"0010" ;
    signal RECTANGLE_VSTART_ball1 : unsigned(15 downto 0):=x"0000" ;
    signal RECTANGLE_HSTART_ball2: unsigned(15 downto 0):=x"000F" ;
    signal RECTANGLE_VSTART_ball2 : unsigned(15 downto 0):=x"0000" ;
    signal RECTANGLE_HSTART_paddle: unsigned(15 downto 0):=x"0000" ;
    signal RECTANGLE_VSTART_paddle: unsigned(15 downto 0):=x"0000" ;
    signal RECTANGLE_HSTART_paddle_2: unsigned(15 downto 0):=x"016B" ;
    signal RECTANGLE_VSTART_paddle_2: unsigned(15 downto 0):=x"00a0" ;
    signal num :integer;
    signal flag :std_logic:='0' ;
    signal bg_flag :std_logic:='0' ;

    type ram_type is array(0 to 15, 0 to 15 ) of std_logic_vector(9 downto 0);
```

```
signal ball_r : ram_type:=(
("0000000000","0000000000," … "0000000000")
   );

signal ball_g : ram_type:=(
("0000000000","0000000000", … "0000000000")
   );

signal ball_b : ram_type:=(
("0000000000","0000000000", … ,"0000000000")
   );

signal ball1_r : ram_type:=(
("0000000000","0000000000", … ,"0000000000")
   );

signal ball1_g : ram_type:=(
("0000000000","0000000000", … ,"0000000000")
   );

signal ball1_b : ram_type:=(
("0000000000","0000000000", … ,"0000000000")
   );

signal ball2_r : ram_type:=(
("0000000000","0000000000", … ,"0000000000")
   );

signal ball2_g : ram_type:=(
("0000000000","0000000000", … ,"0000000000")
   );

signal ball2_b : ram_type:=(
("0000000000","0000000000", … ,"0000000000")
   );

type rom_type is array (0 to 119, 0 to 20) of std_logic_vector (9 downto 0);
constant pad_r : rom_type :=

(
("0000000000","0000000000",…
);

type flag1 is array(integer range 0 to 14, integer range 0 to 19 ) of integer;
```

```
signal background: flag1:=(
( 0,0,0,0,0,0,l1,l2,l3,0,0,r3,r2,r1,0,0,0,0,0,0 ), --l3,l2,l1:left pad score; r3,r2,r1:right pad score;
( 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ),
( 0,0,0,0,0,1,1,1,0,0,0,1,1,1,0,0,0,0,0,0 ),
( 0,0,0,0,1,0,0,0,1,0,1,0,0,0,1,0,0,0,0,0 ),
( 0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0 ),
( 0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0 ),
( 0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0 ),
( 0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0 ),
( 0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0 ),
( 0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0 ),
( 0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0 ),
( 0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0 ),
( 0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0 ),
( 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ),
( 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 )
);

begin

--v1: entity work.rom_32_4 port map(
--Clk => clk,
--addr_x => to_integer(Hcount - (HSYNC + HBACK_PORCH + RECTANGLE_HSTART_paddle)-1 ),
--addr_y => to_integer(Vcount -( VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART_paddle + 1)),
--addrg_x => to_integer(Hcount - (HSYNC + HBACK_PORCH + RECTANGLE_HSTART_paddle)-1 ),
--addrg_y => to_integer(Vcount -( VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART_paddle + 1)),
--addrb_x => to_integer(Hcount - (HSYNC + HBACK_PORCH + RECTANGLE_HSTART_paddle)-1 ),
--addrb_y => to_integer(Vcount -( VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART_paddle + 1)),
--addr_r_x =>to_integer(Hcount - (HSYNC + HBACK_PORCH + RECTANGLE_HSTART_paddle_2)-1 ),
--addr_r_y =>to_integer(Vcount -( VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART_paddle_2 + 1)),
--addrg_r_x => to_integer(Hcount - (HSYNC + HBACK_PORCH + RECTANGLE_HSTART_paddle_2)-1 ),
--addrg_r_y => to_integer(Vcount -( VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART_paddle_2 + 1)),
--addrb_r_x => to_integer(Hcount - (HSYNC + HBACK_PORCH +
```

```
RECTANGLE_HSTART_paddle_2)-1 ),
--addrb_r_y => to_integer(Vcount -( VSYNC + VBACK_PORCH - 1 +
RECTANGLE_VSTART_paddle_2 + 1)),
--
--
--data =>RGB,
--datag =>RGB_G,
--datab =>RGB_B,
--data_r =>RGB_r,
--datag_r =>RGB_G_r,
--datab_r =>RGB_B_r
--
--);


   process (clk)
   begin
      if rising_edge(clk) then
         clk25 <= not clk25;
      end if;
   end process;

   -- Horizontal and vertical counters

   -- new plug in

   one : process (clk)
   begin
if rising_edge(clk) then
      if chipselect = '1' then
            if read ='1'then
                  case address(3 downto 0) is
                                                when "0000" => readdata <=
RECTANGLE_HSTART;
                                                when "0001" => readdata <=
RECTANGLE_VSTART;
                                                when "0010" => readdata <=
RECTANGLE_HSTART_paddle;
                                                when "0011" => readdata <=
RECTANGLE_VSTART_paddle;
                                                when "0100" => readdata <=
RECTANGLE_HSTART_paddle_2;
                                                when "0101" => readdata <=
RECTANGLE_VSTART_paddle_2;
```

```
--                                                  when "0110" => readdata <=
to_unsigned(l3);
--                                                  when "0111" => readdata <=
to_unsigned(l2);
--                                                  when "1000" => readdata <=
to_unsigned(l1);
--                                                  when "1001" => readdata <=
to_unsigned(r3);
--                                                  when "1010" => readdata <=
to_unsigned(r2);
--                                                  when "1011" => readdata <=
to_unsigned(r1);
                                                   when others=> readdata <=
X"0000";
                        end case;
                    elsif write='1' then


                        case address(3 downto 0) is
                                                   when "0000" =>
RECTANGLE_HSTART <= writedata;
                                                   when "0001" =>
RECTANGLE_VSTART <= writedata;
                                                   when "0010" =>
RECTANGLE_HSTART_paddle <= writedata;
                                                   when "0011" =>
RECTANGLE_VSTART_paddle <= writedata;
                                                   when "0100" =>
RECTANGLE_HSTART_paddle_2 <= writedata;
                                                   when "0101" =>
RECTANGLE_VSTART_paddle_2 <= writedata;
                                                   when "0110" => l1 <=
to_integer(writedata);
                                                   when "0111" => l2 <=
to_integer(writedata);
                                                   when "1000" => l3 <=
to_integer(writedata);
                                                   when "1001" => r1 <=
to_integer(writedata);
                                                   when "1010" => r2 <=
to_integer(writedata);
                                                   when "1011" => r3 <=
to_integer(writedata);
                                                   when "1100" =>
```

```
RECTANGLE_HSTART_ball1 <= writedata;
                                                    when        "1101"        =>
RECTANGLE_VSTART_ball1 <= writedata;
                                                    when        "1110"        =>
RECTANGLE_HSTART_ball2 <= writedata;
                                                    when        "1111"        =>
RECTANGLE_VSTART_ball2 <= writedata;
                                                    when                  others=>
RECTANGLE_HSTART <= writedata;
                        end case;
                    end if;
         end if;
end if;
   end process one;

   HCounter : process (clk25)
   begin
      if rising_edge(clk25) then
         if reset = '1' then
            Hcount <= (others => '0');
         elsif EndOfLine = '1' then
            Hcount <= (others => '0');
         else
            Hcount <= Hcount + 1;

         end if;
      end if;
   end process HCounter;

   EndOfLine <= '1' when Hcount = HTOTAL - 1 else '0';

   VCounter: process (clk25)
   begin
      if rising_edge(clk25) then
         if reset = '1' then
            irq <= '0';
            Vcount <= (others => '0');
         elsif EndOfLine = '1' then

               if EndOfField = '1' then
                  Vcount <= (others => '0');
                  flag <= '1';
                  irq <= '1';
--                elsif write = '1' and chipselect = '1'    then
```

```
--                    irq <= '0';
--                    Vcount <= Vcount + 1;
--                    flag <= '0';
                  else
                        Vcount <= Vcount + 1;
                    flag <= '0';
                    irq<= '0';
                  end if;
            else
                  if write = '1' and chipselect = '1' then
                        irq <= '0';
                 end if;
            end if;
        end if;
    end process VCounter;

    EndOfField <= '1' when Vcount = VTOTAL - 1 else '0';

    -- State machines to generate HSYNC, VSYNC, HBLANK, and VBLANK

    HSyncGen : process (clk25)
    begin
        if rising_edge(clk25) then
            if reset = '1' or EndOfLine = '1' then
                vga_hsync <= '1';
            elsif Hcount = HSYNC - 1 then
                vga_hsync <= '0';
            end if;
        end if;
    end process HSyncGen;

    HBlankGen : process (clk25)
    begin
        if rising_edge(clk25) then
            if reset = '1' then
                vga_hblank <= '1';
            elsif Hcount = HSYNC + HBACK_PORCH then
                vga_hblank <= '0';
            elsif Hcount = HSYNC + HBACK_PORCH + HACTIVE then
                vga_hblank <= '1';
            end if;
        end if;
    end process HBlankGen;
```

```
   VSyncGen : process (clk25)
   begin
      if rising_edge(clk25) then
         if reset = '1' then
            vga_vsync <= '1';
         elsif EndOfLine ='1' then
            if EndOfField = '1' then
               vga_vsync <= '1';
            elsif Vcount = VSYNC - 1 then
               vga_vsync <= '0';
            end if;
         end if;
      end if;
   end process VSyncGen;

   VBlankGen : process (clk25)
   begin
      if rising_edge(clk25) then
         if reset = '1' then
            vga_vblank <= '1';
         elsif EndOfLine = '1' then
         if Vcount = VSYNC + VBACK_PORCH - 1 then
               vga_vblank <= '0';
            elsif Vcount = VSYNC + VBACK_PORCH + VACTIVE - 1 then
               vga_vblank <= '1';
            end if;
         end if;
      end if;
   end process VBlankGen;


   -- back ground generator
   background_x <= to_integer(Hcount - HSYNC - HBACK_PORCH - 1 );
   background_y <= to_integer(Vcount - VSYNC - VBACK_PORCH );
   -- Rectangle generator


   RectangleHGenball : process (clk25)
   begin
      if rising_edge(clk25) then

         if reset = '1' or Hcount = HSYNC + HBACK_PORCH + RECTANGLE_HSTART
then
               rectangle_h_ball <= '1';
```

```
        elsif Hcount = HSYNC + HBACK_PORCH + RECTANGLE_HSTART + 16 then
           rectangle_h_ball <= '0';
        end if;
      end if;
   end process RectangleHGenball;

   RectangleHGenball1 : process (clk25)
   begin
      if rising_edge(clk25) then

        if reset = '1' or Hcount = HSYNC + HBACK_PORCH +
RECTANGLE_HSTART_ball1  then
              rectangle_h_ball1 <= '1';
        elsif Hcount = HSYNC + HBACK_PORCH + RECTANGLE_HSTART_ball1 + 16
then
           rectangle_h_ball1 <= '0';
        end if;
      end if;
   end process RectangleHGenball1;

   RectangleHGenball2 : process (clk25)
   begin
      if rising_edge(clk25) then

        if reset = '1' or Hcount = HSYNC + HBACK_PORCH +
RECTANGLE_HSTART_ball2  then
              rectangle_h_ball2 <= '1';
        elsif Hcount = HSYNC + HBACK_PORCH + RECTANGLE_HSTART_ball2 + 16
then
           rectangle_h_ball2 <= '0';
        end if;
      end if;
   end process RectangleHGenball2;




   RectangleHGenl : process (clk25)
   begin
      if rising_edge(clk25) then
        if reset = '1' or Hcount = HSYNC + HBACK_PORCH +
RECTANGLE_HSTART_paddle  then
              rectangle_h_paddle <= '1';
```

```
        elsif Hcount = HSYNC + HBACK_PORCH + RECTANGLE_HSTART_paddle + 21
then
            rectangle_h_paddle <= '0';
        end if;
      end if;
   end process RectangleHGenl;

   RectangleHGenr : process (clk25)
   begin
      if rising_edge(clk25) then
        if reset = '1' or Hcount = HSYNC + HBACK_PORCH +
RECTANGLE_HSTART_paddle_2  then
              rectangle_h_paddle_2 <= '1';
        elsif Hcount = HSYNC + HBACK_PORCH + RECTANGLE_HSTART_paddle_2 + 21
then
            rectangle_h_paddle_2 <= '0';
        end if;
      end if;
   end process RectangleHGenr;



   RectangleVGenball : process (clk25)--rectangle of the ball
   begin
      if rising_edge(clk25) then
        if reset = '1' then
           rectangle_v_ball <= '0';
         elsif EndOfLine = '1' then
            if Vcount = VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART then
                rectangle_v_ball <= '1';
            elsif Vcount = VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART + 16
then
                rectangle_v_ball <= '0';
            elsif Vcount < VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART or
Vcount > VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART + 16  then
                rectangle_v_ball <= '0';
            end if;
          end if;
       end if;
   end process RectangleVGenball;

   RectangleVGenball1 : process (clk25)--rectangle of the ball1
   begin
      if rising_edge(clk25) then
```

```
        if reset = '1' then
           rectangle_v_ball1 <= '0';
         elsif EndOfLine = '1' then
            if Vcount = VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART_ball1 then
               rectangle_v_ball1 <= '1';
            elsif Vcount = VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART_ball1 +
16 then
               rectangle_v_ball1 <= '0';
            elsif Vcount < VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART_ball1
or Vcount > VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART_ball1 + 16    then
               rectangle_v_ball1 <= '0';
            end if;
         end if;
      end if;
   end process RectangleVGenball1;

   RectangleVGenball2 : process (clk25)--rectangle of the ball2
   begin
      if rising_edge(clk25) then
         if reset = '1' then
            rectangle_v_ball2 <= '0';
         elsif EndOfLine = '1' then
            if Vcount = VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART_ball2 then
               rectangle_v_ball2 <= '1';
            elsif Vcount = VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART_ball2 +
16 then
               rectangle_v_ball2 <= '0';
            elsif Vcount < VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART_ball2
or Vcount > VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART_ball2 + 16    then
               rectangle_v_ball2 <= '0';
            end if;
         end if;
      end if;
   end process RectangleVGenball2;


  RectangleVGenl : process (clk25)--rectangle of the left paddle
   begin
      if rising_edge(clk25) then
         if reset = '1' then
            rectangle_v_paddle <= '0';
         elsif EndOfLine = '1' then
            if Vcount = VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART_paddle then
               rectangle_v_paddle <= '1';
```

```
            elsif Vcount = VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART_paddle +
120 then
                rectangle_v_paddle <= '0';
        elsif Vcount < VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART_paddle or
Vcount > VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART_paddle + 120    then
                rectangle_v_paddle <= '0';
            end if;
          end if;
       end if;
   end process RectangleVGenl;

  RectangleVGenr : process (clk25)--rectangle of the right paddle
    begin
       if rising_edge(clk25) then
          if reset = '1' then
             rectangle_v_paddle_2 <= '0';
          elsif EndOfLine = '1' then
             if Vcount = VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART_paddle_2
then
                rectangle_v_paddle_2 <= '1';
             elsif Vcount = VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART_paddle_2
+ 120 then
                rectangle_v_paddle_2 <= '0';

        elsif Vcount < VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART_paddle_2 or
Vcount > VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART_paddle_2 + 120    then
                rectangle_v_paddle_2 <= '0';
            end if;
          end if;
       end if;
   end process RectangleVGenr;

     rectangle_ball <= rectangle_h_ball and rectangle_v_ball;--ball area
     rectangle_ball1 <= rectangle_h_ball1 and rectangle_v_ball1;--ball_1    area
     rectangle_ball2 <= rectangle_h_ball2 and rectangle_v_ball2;--ball_2    area
     rectangle_paddle <= rectangle_h_paddle and rectangle_v_paddle;--left paddle
area
     rectangle_paddle_2 <= rectangle_h_paddle_2 and rectangle_v_paddle_2;-- right
paddle area
   -- Registered video signals going to the video DAC

backgroundgen: process(clk25)
begin
        if rising_edge(clk25) then
```

```
            if background_x >= 0 and background_x <= 640 and background_y >= 0 and
background_y <= 480 then
                  num <= background( background_y/32, (background_x mod
640)/32 );
                        case num is
                                    when 1 => RGB_bg <= background_r( background_y
mod 32, background_x mod 32);
                                                RGB_G_bg<=
background_g( background_y mod 32, background_x mod 32);
                                                RGB_B_bg<=
background_b( background_y mod 32, background_x mod 32);
                                                bg_flag <='1';
                                    when 2 => RGB_bg <= score_r( background_y mod
32, background_x mod 32);
                                                RGB_G_bg<= score_g( background_y
mod 32, background_x mod 32);
                                                RGB_B_bg<= score_b( background_y
mod 32, background_x mod 32);
                                                bg_flag <='1';

                                    when others => RGB_bg  <="1111111111";
                                                      RGB_G_bg<="1111111111";
                                                RGB_B_bg<="1111111111";
                                                bg_flag <='0';
                        end case;
               else
                     RGB_bg <="0000000000";
                     RGB_G_bg<="0000000000";
                   RGB_B_bg<="0000000000";
                   bg_flag <='0';
            end if;
         end if;
  end process backgroundgen;

    VideoOut: process (clk25, reset)
    begin
       if reset = '1' then
          VGA_R <= "1111111111";
          VGA_G <= "1111111111";
          VGA_B <= "1111111111";
       elsif clk25'event and clk25 = '1' then
-- ball
            if rectangle_ball = '1' and
              ball_r(to_integer  (Vcount  -(  VSYNC  +  VBACK_PORCH  -  1  +
```

```
RECTANGLE_VSTART+1)),to_integer(Hcount - (HSYNC + HBACK_PORCH + RECTANGLE_HSTART)-1)) >= "0000000010"  then
    --  ball_g(to_integer (Vcount -( VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART+1)),to_integer(Hcount - (HSYNC + HBACK_PORCH + RECTANGLE_HSTART)-1)) /=X"00"  and
    --  ball_b(to_integer (Vcount -( VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART+1)),to_integer(Hcount - (HSYNC + HBACK_PORCH + RECTANGLE_HSTART)-1)) /=X"00"  then
        VGA_R <= ball_r(to_integer (Vcount -( VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART+1)),to_integer(Hcount - (HSYNC + HBACK_PORCH + RECTANGLE_HSTART)-1)) ;
        VGA_G <= ball_g(to_integer (Vcount -( VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART+1)),to_integer(Hcount - (HSYNC + HBACK_PORCH + RECTANGLE_HSTART)-1)) ;
        VGA_B <= ball_b(to_integer (Vcount -( VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART+1)),to_integer(Hcount - (HSYNC + HBACK_PORCH + RECTANGLE_HSTART)-1)) ;

--end of ball

--ball1
        elsif rectangle_ball1 = '1' and
        ball1_r(to_integer (Vcount -( VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART_ball1+1)),to_integer(Hcount - (HSYNC + HBACK_PORCH + RECTANGLE_HSTART_ball1)-1)) /= "0000000000" then
    --  ball_g(to_integer (Vcount -( VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART+1)),to_integer(Hcount - (HSYNC + HBACK_PORCH + RECTANGLE_HSTART)-1)) /=X"00"  and
    --  ball_b(to_integer (Vcount -( VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART+1)),to_integer(Hcount - (HSYNC + HBACK_PORCH + RECTANGLE_HSTART)-1)) /=X"00"  then
        VGA_R <= ball1_r(to_integer (Vcount -( VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART_ball1+1)),to_integer(Hcount - (HSYNC + HBACK_PORCH + RECTANGLE_HSTART_ball1)-1)) ;
        VGA_G <= ball1_g(to_integer (Vcount -( VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART_ball1+1)),to_integer(Hcount - (HSYNC + HBACK_PORCH + RECTANGLE_HSTART_ball1)-1)) ;
        VGA_B <= ball1_b(to_integer (Vcount -( VSYNC + VBACK_PORCH - 1 + RECTANGLE_VSTART_ball1+1)),to_integer(Hcount - (HSYNC + HBACK_PORCH + RECTANGLE_HSTART_ball1)-1)) ;
--end of ball1

--ball2
        elsif rectangle_ball2 = '1' and
```

```
            ball2_r(to_integer (Vcount -( VSYNC + VBACK_PORCH - 1 +
RECTANGLE_VSTART_ball2+1)),to_integer(Hcount - (HSYNC + HBACK_PORCH +
RECTANGLE_HSTART_ball2)-1)) /= "0000000000" then
      --       ball_g(to_integer (Vcount -( VSYNC + VBACK_PORCH - 1 +
RECTANGLE_VSTART+1)),to_integer(Hcount - (HSYNC + HBACK_PORCH +
RECTANGLE_HSTART)-1)) /=X"00"      and
      --       ball_b(to_integer (Vcount -( VSYNC + VBACK_PORCH - 1 +
RECTANGLE_VSTART+1)),to_integer(Hcount - (HSYNC + HBACK_PORCH +
RECTANGLE_HSTART)-1)) /=X"00"    then
            VGA_R <= ball2_r(to_integer (Vcount -( VSYNC + VBACK_PORCH - 1 +
RECTANGLE_VSTART_ball2+1)),to_integer(Hcount - (HSYNC + HBACK_PORCH +
RECTANGLE_HSTART_ball2)-1)) ;
            VGA_G <= ball2_g(to_integer (Vcount -( VSYNC + VBACK_PORCH - 1 +
RECTANGLE_VSTART_ball2+1)),to_integer(Hcount - (HSYNC + HBACK_PORCH +
RECTANGLE_HSTART_ball2)-1)) ;
            VGA_B <= ball2_b(to_integer (Vcount -( VSYNC + VBACK_PORCH - 1 +
RECTANGLE_VSTART_ball2+1)),to_integer(Hcount - (HSYNC + HBACK_PORCH +
RECTANGLE_HSTART_ball2)-1)) ;

--end of ball2

-- left paddle
            elsif rectangle_paddle = '1' then
--          VGA_R <= rgb;
--          VGA_G <= rgb_g;
--          VGA_B <= rgb_b;
             VGA_R <= pad_r(to_integer(Vcount -( VSYNC + VBACK_PORCH - 1 +
RECTANGLE_VSTART_paddle + 1)), to_integer(Hcount - (HSYNC + HBACK_PORCH +
RECTANGLE_HSTART_paddle)-1 ) );
             VGA_G <= pad_g(to_integer(Vcount -( VSYNC + VBACK_PORCH - 1 +
RECTANGLE_VSTART_paddle + 1)), to_integer(Hcount - (HSYNC + HBACK_PORCH +
RECTANGLE_HSTART_paddle)-1 ) );
             VGA_B <= pad_b(to_integer(Vcount -( VSYNC + VBACK_PORCH - 1 +
RECTANGLE_VSTART_paddle + 1)), to_integer(Hcount - (HSYNC + HBACK_PORCH +
RECTANGLE_HSTART_paddle)-1 ) );
--end of left paddle

--right paddle
            elsif rectangle_paddle_2 = '1' then
--          VGA_R <= rgb_r;
--          VGA_G <= rgb_g_r;
--          VGA_B <= rgb_b_r;
             VGA_R <= pad_r(to_integer(Vcount -( VSYNC + VBACK_PORCH - 1 +
RECTANGLE_VSTART_paddle_2 + 1)), to_integer(Hcount - (HSYNC + HBACK_PORCH +
```

```
RECTANGLE_HSTART_paddle_2)-1 ) );
            VGA_G <= pad_g(to_integer(Vcount -( VSYNC + VBACK_PORCH - 1 +
RECTANGLE_VSTART_paddle_2 + 1)), to_integer(Hcount - (HSYNC + HBACK_PORCH +
RECTANGLE_HSTART_paddle_2)-1 ) );
            VGA_B <= pad_b(to_integer(Vcount -( VSYNC + VBACK_PORCH - 1 +
RECTANGLE_VSTART_paddle_2 + 1)), to_integer(Hcount - (HSYNC + HBACK_PORCH +
RECTANGLE_HSTART_paddle_2)-1 ) );

----end of right paddle

-- back ground
            elsif bg_flag = '1' then
              VGA_R <=    RGB_bg;
              VGA_G <=    RGB_G_bg;
              VGA_B <=    RGB_B_bg;
-- end of back ground

            elsif vga_hblank = '0' and vga_vblank = '0' then
              VGA_R <= "0000000000";
              VGA_G <= "0000000000";
              VGA_B <= "0000000000";
            else
              VGA_R <= "0000000000";
              VGA_G <= "0000000000";
              VGA_B <= "0000000000";
         end if;
    end if;
    end process VideoOut;

    VGA_clk <= clk25;
    VGA_SYNC <= '0';
    VGA_HS <= not vga_hsync;
    VGA_VS <= not vga_vsync;
    VGA_BLANK <= not (vga_hsync or vga_vsync);

end rtl;
```