MIDDLE EAST TECHNICAL UNIVERSITY

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

EE 314 DIGITAL ELECTRONICS LAB REPORT
Game Two: WAR

Student Name: Najmush Sakib Khan

Student ID: 1906643

Student Name: Görkem Kılınç

Student ID: 1937143

Table of Content

List of figures

Abstract

This is a report about the EE314 project i.e. war. The game of war basically has a ball rotating in circular or elliptical orbit and the objective of the game is to hit that target. If we hit that target the game is over otherwise the game reloads the missile and makes it ready. This game was implemented using Altera V FPGA and Quartus II software.

Introduction

In this project, goal is to implement a game called "War", which is desired to be playable on FPGA. As FPGA, Altera's DE1-SoC development board and as the programming software Quartus II 14.1 Software was used. The code of the game is written in Verilog Hardware Description Language.

The main purpose of this game is to shoot the rotating ball. In the requirements of the original game, many colourful balls are supposed to rotate and the missile is supposed to hit the same colours. But in this project there is only one ball to hit. If the player hits the ball, a white screen appears; if a miss occurs the missile returns back to its initial position to be shot again. The game actually ends only if the player shoots the target. There is no real challenge in the game since this is the case. However, when we played the game we had miss hits for more than 75% of the time which made this game quite fun to play with.

One of the stages in the project was to perform the operation of the monitor via VGA interface. After having a meaningful display in the monitor, logic of the game is applied. These two stages are combined as two hierarchical blocks of the project as will be mentioned again.

Design

Firstly, VGA interface of the project is implemented. During this step, two signals horizontal synchronization and vertical synchronization are defined. These are very important to get a display from the screen. The horizontal synchronization signal regulates the pixels in a horizontal line segment to synchronize the RGB output of the FPGA with the input of the monitor. Likewise, vertical synchronization is important to synchronize the vertical stripes and signal the end of the frame. The main need for these adjustments is the fact that it takes time for the monitor to jump to one end of line pixel to one new line pixel to process which is known as retrace. Also, the monitor scans more than the visible area. The time it takes to scan these areas are called back porch and front porch. At front porch, the pixel being processed is in front of the visible area. At back porch, the pixel being processed is behind the visible area. During these durations, monitor needs not to have input from VGA interface. Also, a VGA clock is required to synchronize the output of the FPGA.

Secondly, the game code is considered. Game is a simple game except the rotating balls. It is not easy to write an algorithm to achieve a rotational motion in HDL. Instead, we specified 8 points to move around. Also, instead of a ball shape, the moving object Is a square in this project. We defined the objects in the project: the target and the missile. A "wire" type of variable is defined for the collisions.

Implementation

Two hierarchical blocks are defined. One of them is for the VGA Interface. This block is necessary for the transfer of the display to the monitor. It has nothing to do with the logic of the game. The other one is the main block. It involves the logic of the game and the output of the other block as well. This top block calls the other block as a function, together they constitute the project.

a. **The VGA Interface**

The durations of the porches and retraces are defined according to specifications of 60 Hz monitor.

```
localparam HD = 640; // horizontal display area

localparam HF = 48; // h front left border

localparam HB = 16; // h back right border

localparam HR = 96; // h retrace

localparam VD = 480; // vertical display area

localparam VF = 10;  // v front top border

localparam VB = 33;  // b back bottom border

localparam VR = 2;   // v retrace
```

25 MHz clock which will be called as pixel_tick from now on is generated from 50MHz clock to have VGA clock

```
reg mod2_reg;

wire mod2_next;

assign mod2_next = ~mod2_reg;

assign pixel_tick = mod2_reg;
```

Whenever reset pin is not on, hsync and vsync is on, and the coordinates of the pixel being processed is stocked by the counters, which are assigned to pixel_x and pixel_y. Those are previously defined registers, which is not necessary to mention about.

```
always@(posedge clk, posedge reset)

if (reset)
```

```
begin
mod2_reg<= 1'b0;
v_count_reg <= 0;
h_count_reg <= 0;
v_sync_reg <= 1'b0;
h_sync_reg <= 1'b0;
end
else
begin
mod2_reg <= mod2_next;
v_count_reg <= v_count_next;
h_count_reg <= h_count_next;
v_sync_reg <= v_sync_next;
h_sync_reg <= h_sync_next;
end
```

When h_end (v_end) reaches to the end of the cycle, the counting operation repeats itself by first resetting h_end (v_end) to 0

```
//end of horizontal counter at 799
assign h_end = (h_count_reg == (HD+HF+HB+HR-1));
// end of vertical counter at 524
assign v_end = (v_count_reg==(VD+VF+VB+VR-1));
always@*
if (pixel_tick) // 25 MHz clock
if (h_end)
h_count_next=0;
else
h_count_next= h_count_reg +1;
else
h_count_next= h_count_reg;
always@*
```

```
if (pixel_tick) // 25 MHz clock
if (h_end)
h_count_next=0;
else
h_count_next= h_count_reg +1;
else
h_count_next= h_count_reg;
```

video_on is defined to see that the pixel being treated is inside the visible area

```
assign video_on = (h_count_reg<HD) && (v_count_reg < VD);
```

hsync, vsync, video_on, p_tick, pixel_x, pixel_y are the outputs of this subblock.

b. **The Game**

The outputs of the previous part are transferred to process like this:

```
 vga_sync vsync_unit
 (.clk(clk), .reset(reset), .hsync(hsync), .vsync(vsync),
.video_on(video_on), .p_tick(p_tick), .pixel_x(pixel_x), .pixel_y(pixel_y));
```

In this part, two counters are generated. count 1 will be used in general purpose whereas count2 is the counter for the missile. Here, the limits 900000 and 400 are found by trial and error to accomplish a playable game.

```
wire mov_clk;
assign mov_clk = (count1==0);
always@(posedge clk)
begin count1=count1+1;
if (count1==900000)
count1=0;
end
always@(posedge mov_clk)
if (count2>400)
count2=0;
```

```
else if ( ((count2==1)||(count2==0))&& shoot)
count2=0;
else
count2=count2+1;
```

A signal with frequency of hundredth of mov_clk is generated to be the enemy clock. The purpose of the enemy clock is the adjustment of the state changes of the target. At positive edges of this clock, the target changes its position to the next position, forming a circular shape which has 8 corners.

```
always@(posedge mov_clk)
if (count4<100) count4=count4+1;
else count4=0;
wire enemy_clk;
assign enemy_clk=(count4==1);
always@(posedge enemy_clk)
begin
count3=count3+1;
case(count3)
0 : begin x= MAX_X/2; y= 100; end
1 : begin x= MAX_X/2+40; y= 125; end
2 : begin x= MAX_X/2+50; y= 150; end
3 : begin x= MAX_X/2+40; y= 175; end
4 : begin x= MAX_X/2; y= 200; end
5 : begin x= MAX_X/2-40; y= 175; end
6 : begin x= MAX_X/2-50; y= 150; end
7 : begin x= MAX_X/2-40; y= 125; end
endcase
end
```

Here game_over is defined as the collision of the missile and the target. When game_over is on, count5, which is only a latch, is set and the preceding display output is generated by inquiring every step.

```
wire game_over;

assign game_over = (((440-count2)==(y-5))&& (x==320));

reg count5;

initial count5=0;

always@(posedge clk)

if (game_over)

count5 = 1;

else if(reset)
```

```
count5=0;
```

If reset pin is on, which has the highest precedence, the screen will turn to black. Otherwise, if the game is over (count5 is 1) screen will turn to white. Otherwise, if the pixel is in the borders of the missile or the enemy, the screen is painted to green or red respectively. If none of the specifications match, the the pixel will be painted to blue, these pixels together forming the background.

```
always@(posedge clk , posedge reset)
if(reset)
rgb_reg <=0;
else if (count5)
rgb_reg <=3'b111;
else if (wall_on)
rgb_reg <=3'b010;
else if (enemy_on)
rgb_reg <=3'b100;
else
rgb_reg <= 3'b001;
```

Finally, small but a very important line of code is shown below. As explained, video_on represents that pixel is in the visible area. For the monitor to function properly, it is necessary to paint the porch and retrace to black.

```
assign rgb = (video_on) ? rgb_reg : 3'b0;
```

Test

Having completed all the code work in our computer, we tried the code in our FPGA, Altera III. We tried it in computer labs, obtained the expected screen. Afterwards, when we used the proper pin assignments and tried the project in Altera V, we achieved the same result, and mission is accomplished.

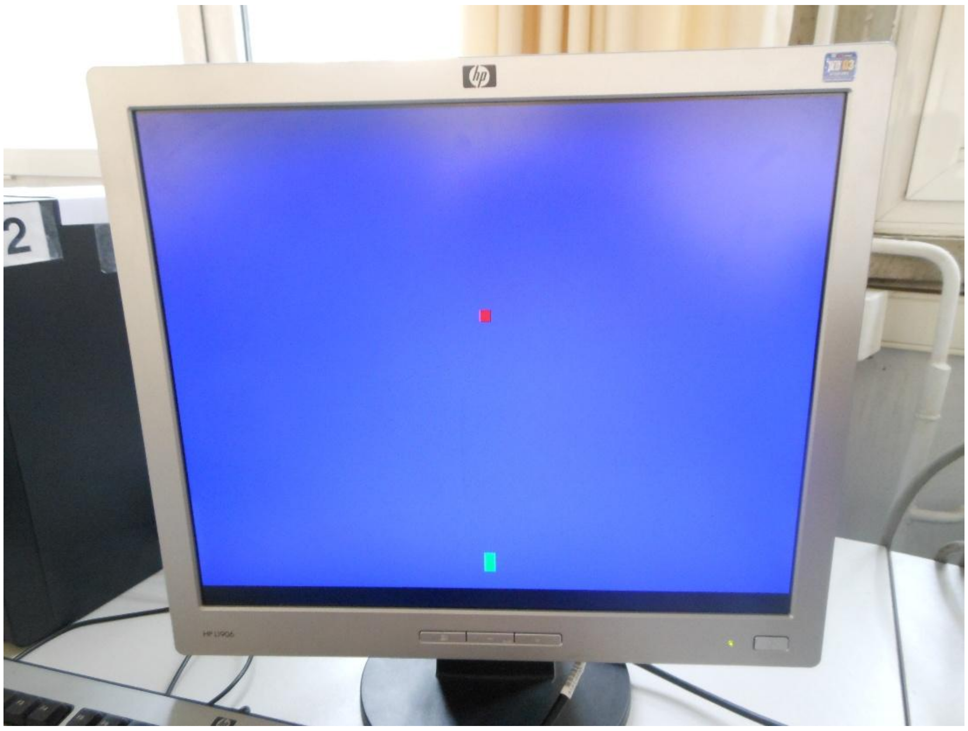
Figure1:In-game screen

Figure2:When the player hits the target, a white end-game screen appears.

Conclusion

In this project, we have designed a classic arcade game of rotating balls codenamed WAR. We were able to successfully complete most of the requirements of the project and was able to play the game without any glitch. We have designed the project using Altera cyclone V FPGA using Verilog Hardware Description Language. During our design we faced many issues due to the limitations of both the hardware and the Quartus II software. However, using our knowledge of verilog and a sudden stroke of inspiration, we were able to finally finish our project. In the end, our final project was quite immersive and fun to play.

Biosketch

NAME **Najmush Sakib Khan**	**POSITION TITLE** **Student**
COMMON USER NAME **SAKIB**	
EDUCATION / TRAINING	

INSTITUTION AND LOCATION	**DEGREE**	**MM/YY**	**FIELD OF STUDY**
MIDDLE EAST TECHNICAL UNIVERSITY	BS	06/16	ELECTRICAL & ELECTRONICS ENGINEERING
EDEXEL UK	A LEVELS	06/12	GENERAL

A. Personal Statement

The goal of this project is to create a game of war. Using our expertise on the field on Verilog programming we were able to carry out the required specifications.

B. Positions and Honours

2014-2015) Research Assistant, METU

(June-July 2014) Internship at 210 MW Sidddirgonj Power Plant, BPDB

NAME	**POSITION TITLE**

Görkem Kılınç	**Student**
COMMON USER NAME GÖRKEM	
EDUCATION / TRAINING	

INSTITUTION AND LOCATION	**DEGREE**	**MM/YY**	**FIELD OF STUDY**
MIDDLE EAST TECHNICAL UNIVERSITY	BS	06/16	ELECTRICAL & ELECTRONICS ENGINEERING

A. Personal Statement

This projects aims to make a game based on FPGA. Although HDL is not a suitable language for writing complicated codes, we can make small games Using our expertise on the field on Verilog programming we were able to carry out the required specifications.

B. Positions and Honours

(August 2014) Internship at Arçelik Washing Machine Plant, Çayırova/İstanbul

Reference

1) FPGA programming by Verilog by Pong P. Chu
2) FPGA now what? by Dave Vandenbout
3)Design through Verilog HDL by T.R. Padmanabhan and B. Bala Tripura Sundari

Appendix

Verilog HDL code of this project:

Top Level Code:

```
module trial3
(
input wire clk, reset,shoot,
output wire hsync, vsync, p_tick,
output wire [2:0] rgb
);

// signal declaration

reg[2:0] rgb_reg;
wire video_on;
wire [9:0] pixel_x, pixel_y;

// instantiate vga sync circuit
 vga_sync vsync_unit
 (.clk(clk), .reset(reset), .hsync(hsync), .vsync(vsync),
.video_on(video_on), .p_tick(p_tick), .pixel_x(pixel_x), .pixel_y(pixel_y));

//////////////////////////////// some registers
reg [28:0] count1;
reg [8:0] count2;
reg [2:0] count3;

////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
//////////   ///////////////////////////////////////////

wire mov_clk;
assign mov_clk = (count1==0);
always@(posedge clk)
begin count1=count1+1;
if (count1==900000)
count1=0;
end
always@(posedge mov_clk)
if (count2>400)
count2=0;
else if ( ((count2==1)||(count2==0))&& shoot)
count2=0;
```

```
else
count2=count2+1;

/////////////////////////////////////////////////////////////////

reg[8:0] x,y;
reg[7:0]count4;

localparam MAX_X= 640;

always@(posedge mov_clk)
if (count4<100) count4=count4+1;
else count4=0;

wire enemy_clk;
assign enemy_clk=(count4==1);

always@(posedge enemy_clk)
begin
count3=count3+1;
case(count3)
0 : begin x= MAX_X/2; y= 100; end
1 : begin x= MAX_X/2+40; y= 125; end
2 : begin x= MAX_X/2+50; y= 150; end
3 : begin x= MAX_X/2+40; y= 175; end
4 : begin x= MAX_X/2; y= 200; end
5 : begin x= MAX_X/2-40; y= 175; end
6 : begin x= MAX_X/2-50; y= 150; end
7 : begin x= MAX_X/2-40; y= 125; end
endcase
end

/////////////////////////////////////////////////////
// rgb buffer
wire wall_on;
assign wall_on = ((pixel_x <=325) &&  (pixel_x >=315) && (pixel_y <=(460-count2)) &&
(pixel_y>=(440-count2)));


wire enemy_on;
assign enemy_on = ((x-5)<=pixel_x)&&(pixel_x<=(x+5))&&((y-
5)<=pixel_y)&&(pixel_y<=(y+5));


//////////////////////////////////////////////////////////////////////////////////////////////////
```

```
wire game_over;
assign game_over = (((440-count2)==(y-5))&& (x==320));
reg count5;
initial count5=0;
always@(posedge clk)
if (game_over)
count5 = 1;
else if(reset)
count5=0;

/////////////////////////////
always@(posedge clk , posedge reset)
if(reset)
rgb_reg <=0;
else if (count5)
rgb_reg <=3'b111;
else if (wall_on)
rgb_reg <=3'b010;
else if (enemy_on)
rgb_reg <=3'b100;
else
rgb_reg <= 3'b001;
//output

assign rgb = (video_on) ? rgb_reg : 3'b0;


endmodule
```

VGA Sync Code:

```
module vga_sync
( input wire clk, reset,
output wire hsync, vsync, video_on, p_tick,
output wire [9:0] pixel_x, pixel_y
);

// constant declaration
// VGA 640 by 480 sync parameters

localparam HD = 640; // horizontal display area
localparam HF = 48; // h front left border
localparam HB = 16; // h back right border
localparam HR = 96; // h retrace
localparam VD = 480; // vertical display area
localparam VF = 10;  // v front top border
localparam VB = 33;  // b back bottom border
localparam VR = 2;   // v retrace

// mod2 counter

reg mod2_reg;
wire mod2_next;

// sync counters
reg[9:0] h_count_reg, h_count_next;
reg[9:0] v_count_reg, v_count_next;

// output buffers
reg v_sync_reg, h_sync_reg;
wire v_sync_next, h_sync_next;

//status signals
wire h_end, v_end, pixel_tick;
//body, registers
always@(posedge clk, posedge reset)
if (reset)
begin
mod2_reg<= 1'b0;
v_count_reg <= 0;
h_count_reg <= 0;
v_sync_reg <= 1'b0;
h_sync_reg <= 1'b0;
end
else
begin
mod2_reg <= mod2_next;
v_count_reg <= v_count_next;
h_count_reg <= h_count_next;
v_sync_reg <= v_sync_next;
h_sync_reg <= h_sync_next;
end

// mod2 circuit to generate 25MHz from 50 MHz internal clock

assign mod2_next = ~mod2_reg;
assign pixel_tick = mod2_reg;
```

```
/// status signals
//end of horizontal counter at 799

assign h_end = (h_count_reg == (HD+HF+HB+HR-1));

// end of vertical counter at 524

assign v_end = (v_count_reg==(VD+VF+VB+VR-1));

// next state logic of mod 800 horizontal sync counter

always@*
if (pixel_tick) // 25 MHz clock
if (h_end)
h_count_next=0;
else
h_count_next= h_count_reg +1;
else
h_count_next= h_count_reg;
// next state logic of mod 525 vertical sync counter

always@*
if (pixel_tick & h_end)
if (v_end)
v_count_next=0;
else
v_count_next = v_count_reg +1;
else
v_count_next = v_count_reg;

// h and v sync buffered to avoid glitch
// h sync next assigned between 656 and 751

assign h_sync_next = (h_count_reg >=(HD+HB) && h_count_reg <=(HD+HB+HR-1));

// v sync next assigned between 490 & 491

assign v_sync_next = (v_count_reg >= (VD+VB) && v_count_reg <= (VD+VB+VR-1));

// video on/off
assign video_on = (h_count_reg<HD) && (v_count_reg < VD);

//output
```

```
assign hsync= h_sync_reg;
assign vsync= v_sync_reg;
assign pixel_x= h_count_reg;
assign pixel_y= v_count_reg;
assign p_tick= pixel_tick;


endmodule
```