Laboratory of Advanced Electronics Final Project

For the final project, I wrote a code for the Spartan-3 FPGA able to simulate on a monitor a game in which a ball falls down and a bar should catch it. In order to do that, I considered the following steps:

1. VGA signal generator
2. Bar generator
3. Random ball generator
4. Score counter
5. Monitor display.

1 VGA signal generator

For the project, I used a Dell E173FP 17" monitor, which is able to adapt the resolution automatically. The FPGA includes a VGA display port, with four outputs for each color (`VGA_G`, `VGA_B`, `VGA_R` respectively for green, blue and red colors) and two connections for the timing of the signals on the monitor, respectively `VGA_HSYNC` for the horizontal signal and `VGA_VSYNC` for the vertical signal. In Figure 1, the basic colors are reported.

VGA_R[3:0]	VGA_G[3:0]	VGA_B[4:0]	Resulting Color
0000	0000	0000	Black
0000	0000	1111	Blue
0000	1111	0000	Green
0000	1111	1111	Cyan
1111	0000	0000	Red
1111	0000	1111	Magenta
1111	1111	0000	Yellow
1111	1111	1111	White

Figure 1

I used a 640x480 pixel resolution, with a refresh rate of 60 Hz. The required clock frequency is 25.175 MHz, but the monitor is able to exploit also a frequency of 25 MHz, easily produced from the FPGA 50MHz-clock by a Frequency Divider with period 1. The module is:

```
module    Module_FrequencyDivider (   input clk_in,
                                     input [29:0] period,

                                     output reg clk_out);
reg   [29:0] counter;

always @(posedge clk_in) begin
   if (counter >= (period - 1)) begin
      counter = 0;
      clk_out = ~clk_out;
   end else
      counter = counter + 1;
end
endmodule
```

The sync period should take into account also some blanking time, which consists on a pulse width and a back porch, before the display area, and a front porch after the display area. All the values are reported in Figure 2[1].

[1]Both Figure 1 and Figure 2 are taken from Spartan-3A/3AN FPGA Starter Kit Board User Guide, UG334 (v1.1)

Symbol	Parameter	Vertical Sync			Horizontal Sync	
		Time	Clocks	Lines	Time	Clocks
T_S	Sync pulse time	16.7 ms	416,800	521	32 µs	800
T_{DISP}	Display time	15.36 ms	384,000	480	25.6 µs	640
T_{PW}	Pulse width	64 µs	1,600	2	3.84 µs	96
T_{FP}	Front porch	320 µs	8,000	10	640 ns	16
T_{BP}	Back porch	928 µs	23,200	29	1.92 µs	48

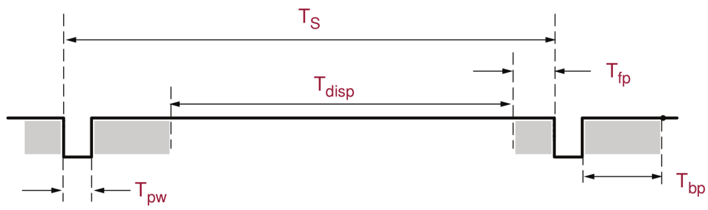


Figure 2

The module to generate the horizontal pulse is reported here below, which takes as input the clock at 25 MHz and gives as output `VGA_HSYNC` (needed for the VGA port), the horizontal coordinates on the display area as `H_Coord` and the signal at which the information can be displayed as `H_Display` (if it is 1, the pixels on the screen can light up, if it is 0, the signal is in the blanking area):

```
module Module_HorizontalSync ( input clk,        //25 MHz

                               output H_Sync,
                               output reg [9:0] H_Coord, //Coordinates from 0 to 639
                               output reg H_Display);
reg [10:0] counter;

always @(posedge clk) begin
    if(counter == 11'd799)  //800 = Total number of clocks needed for the Horizontal Sync
    begin
        counter = 11'b0;
    end else begin
        counter = counter + 1'b1;
    end
end

assign H_Sync = (counter < 11'd96)? 0 : 1;   //96 = Clocks in the Pulse Width for the Horizontal Sync
always @(posedge clk) begin
    if((counter > 11'd144) && (counter < 11'd784)) //Display area: from 144 to 784 clk
    begin
        H_Display <= 1'b1;
        H_Coord <= H_Coord + 1'b1;
    end else begin
        H_Coord <= 7'b0;
        H_Display <= 0;
    end
end
endmodule
```

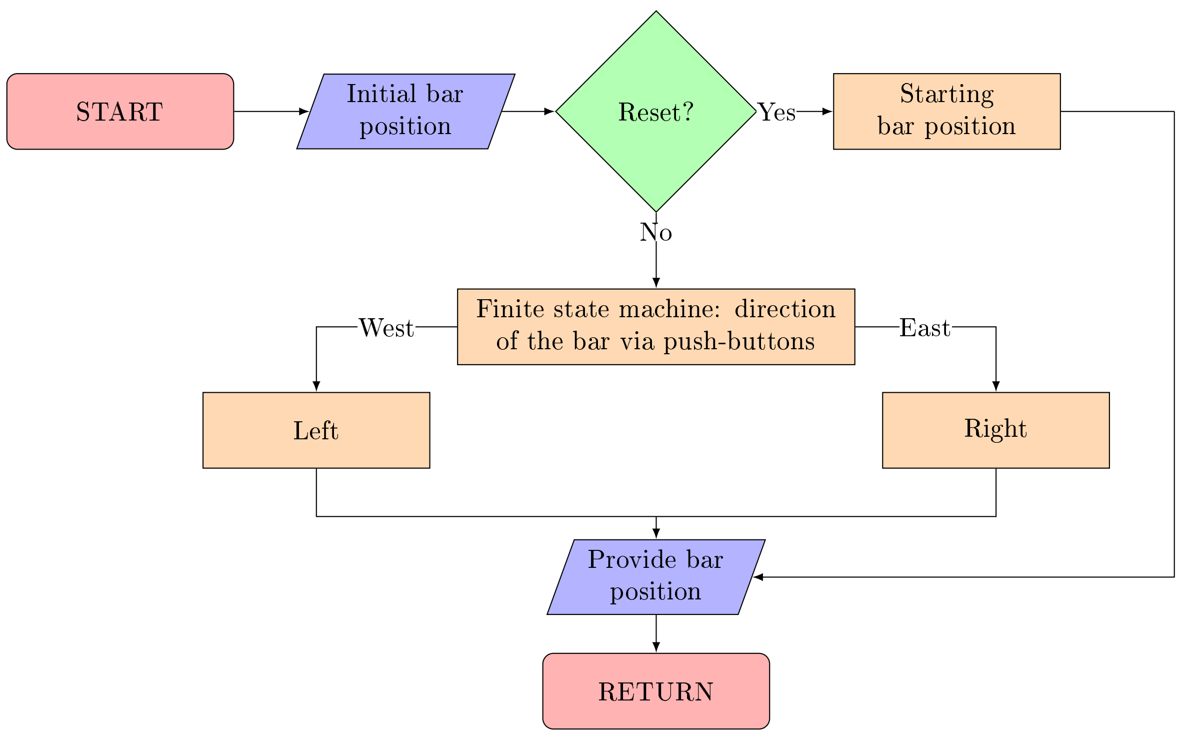
Similar idea for the vertical sync, but with the difference that to obtain the vertical coordinates on the display area, `V_Coord`, a divider is needed, `divide_counter`. In fact, the total number of clocks needed for the vertical sync is 416800, which is the product of the 521 lines times the 800 columns. In order to simplify the reference of the lines on the display area, the divider divides the vertical clock with the relative line, and in this way as output the array `V_Coord` going from 0 to 479 is obtained.

```
module Module_VerticalSync ( input clk,        //25 MHz
```

June 19, 2008, www.xilinx.com

```
                            output V_Sync,
                            output reg [9:0] V_Coord,     //Coordinates from 0 to 479
                            output reg V_Display);
reg [19:0] counter;
reg [10:0] divide_counter;

always @(posedge clk) begin
   if(counter == 20'd416799) begin  //416800 = Total number of clocks needed for the Vertical Sync
      counter <= 20'b0;
   end else begin
      counter <= counter + 1'b1;
   end
end

assign V_Sync = (counter < 20'd1600)? 0 : 1;     //1600 = Clocks in the Pulse Width for the Vertical Sync (2 lines x 800 columns)
always @(posedge clk) begin
   if((counter > 20'd24800) && (counter < 20'd408801)) begin   //Display Area is on
      V_Display <= 1'b1;
      if(divide_counter == 11'd799) begin
         divide_counter <= 0;
         if(V_Coord == 10'd479) begin
            V_Coord <= 0;
         end else begin
            V_Coord <= V_Coord + 1'b1;
         end
      end else begin
         divide_counter <= divide_counter + 1'b1;
         V_Coord <= V_Coord;
      end
   end else begin
      V_Coord <= 7'b0;
      V_Display <= 0;
   end
end
endmodule
```

2 Bar generator



To generate the coordinates of the bar, a Finite State Machine has been used. The state is determined by the direction of the bar, which can be or on the left or on the right depending on the pushed push-button. The speed of the movement depends on the clock, which has a rate of 31.25 Hz (produced by a Frequency Divider). The bar moves of 10 pixels on, set in the `dimension` input.

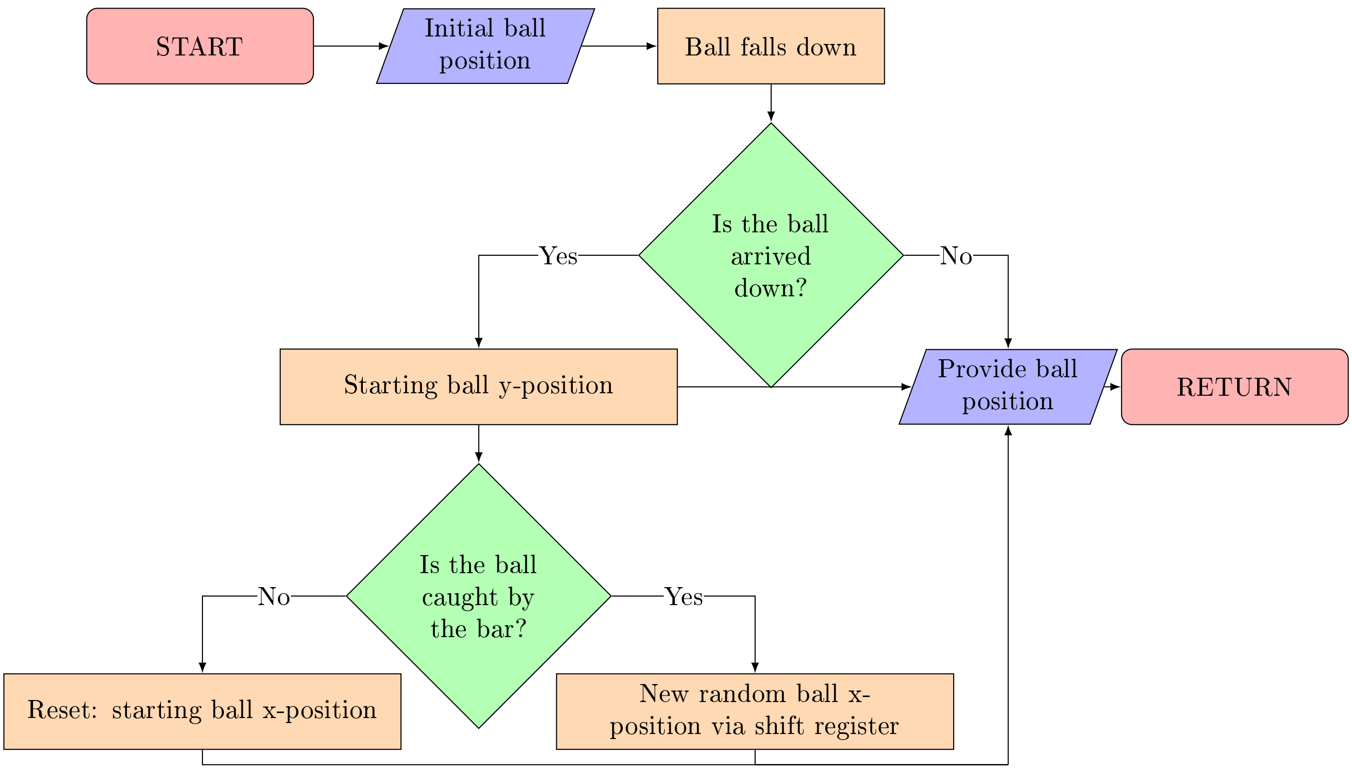
```
module Module_FSM_Bar(  input clk_in,
                        input reset, new_game,
                        input [3:0] direction,
                        input [9:0] dimension,

                        output [9:0] movement);
reg signed [9:0] H_Direction;
buf(movement, H_Direction);

always @(posedge clk_in) begin
   if (reset || new_game) begin     //Starting position
      H_Direction <= 10'd300;
   end else if(~reset) begin
      case (direction)
         4'b0001 : H_Direction <= (H_Direction + dimension); // right
         4'b0010 : H_Direction <= (H_Direction - dimension); // left
      endcase
   end
end
endmodule
```

The bar is then generated in the top module `CatchTheBallGame.v` with the following statement:

```
wire w_bar = ( ((wb_HCoord >= (wb_barMovement)) && (wb_HCoord <= (wb_barMovement +
    bar_lenght)) && (wb_VCoord >= 10'd455) && (wb_VCoord < 10'd465)) );
```

with length `bar_lenght = 11'd50`.

3 Random ball generator



The generation of the position of the falling balls is pseudo-random, so a shift register has been exploited. When `reset` or `new_game` are up, the initial position is set at coordinates `(X_rnd, Y_rnd =`

10'd320, 10'd10) of the display area. When the ball hits the bar, the signal `caught` becomes 1 and a new initial horizontal position is generated, while the initial vertical position remains at 10 pixels. At lines 12 and 14 of the code `Module_RNDnumberGenerator`, two `if` conditions avoid the generation of the ball outside the display area and in the corner of the screen. Similarly, at line 19, another `if` condition obliges the ball not going under the bar, by restoring its initial vertical position.

The radius of the ball is called through the input `radius` and it is 5 pixels.

Finally, the initial clock determines the speed of the ball falling down, and in the top module three different clocks are defined, depending on the number of caught balls, in order to increase the difficulty of the game (respectively at 10 Hz for the first 5 points, at 16 Hz for the following 5 points and after this at 25 Hz).

```
1  module Module_RNDnumberGenerator (  input clk_in,
2                                      input reset, new_game, caught,
3                                      input [10:0] radius,
4  
5                                      output reg [9:0] X_rnd, Y_rnd);
6  always @(posedge clk_in) begin
7     if (reset || new_game) begin
8        X_rnd = 10'd320;
9        Y_rnd <= 10'd10;
10    end else if (caught) begin    //When the ball touches the bar: random new x-position
11       X_rnd[9:0] = {X_rnd[8:0], ~(X_rnd[9]^X_rnd[7])};
12       if (X_rnd >= 10'd639) begin          //No balls in the corner
13          X_rnd = 10'd629;
14       end else if (X_rnd <= 10'd22) begin //No balls in the corner
15          X_rnd = 10'd23;
16       end
17    end
18    Y_rnd <= Y_rnd + radius + radius;
19    if (Y_rnd >= 10'd457) begin    //When the ball is under the bar: start from the top
20       Y_rnd <= 10'd10;
21    end
22 end
23 endmodule
```

Also the ball is generated in the top module:

```
wire w_ball = ((wb_HCoord-wb_Xball)*(wb_HCoord-wb_Xball)) <= (radius*radius-((wb_VCoord
    -wb_Yball)*(wb_VCoord-wb_Yball)));
```

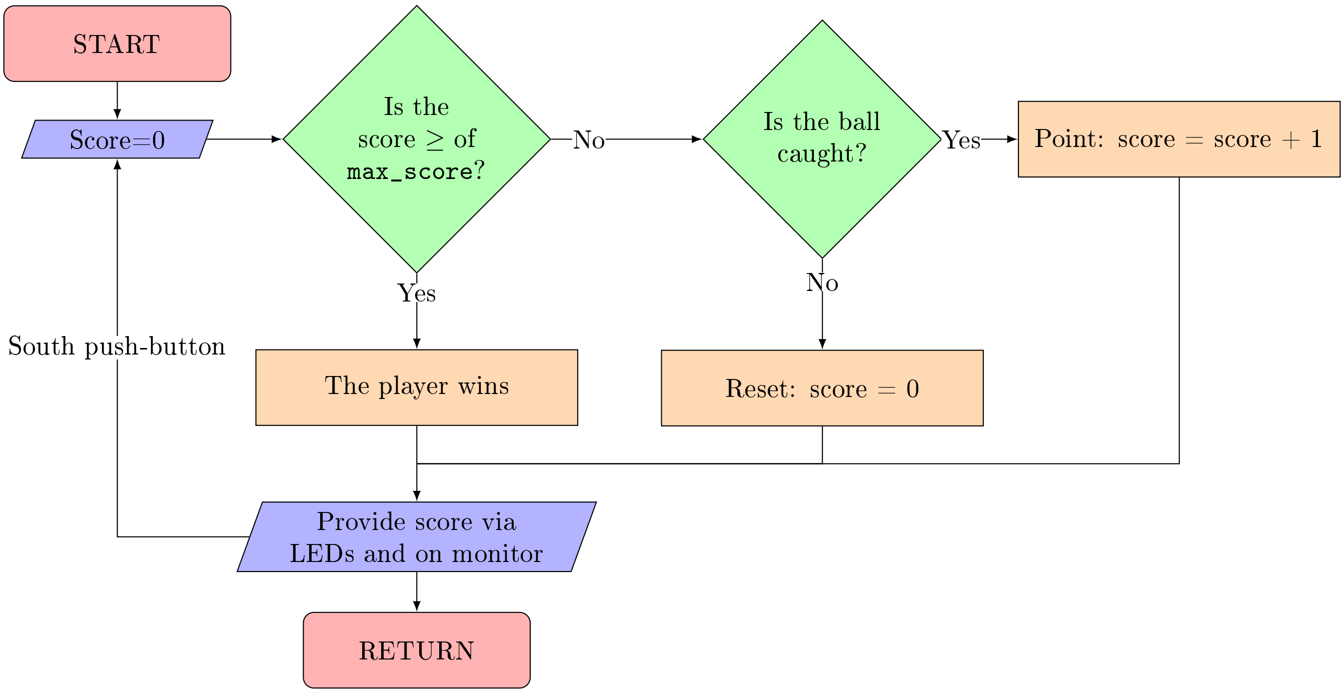
where the circle is centered in (x_c, y_c) = (`wb_Xball, wb_Yball`) and is defined by the equation:

$$(x - x_c)^2 \leq r^2 - (y - y_c)^2 \tag{1}$$

4 Score counter

At this point, I defined a module, called `Module_Score`, which counts how many times the ball is caught, in order to provide the player's score. The flowchart of the process is shown here below. This module needs the same three different clocks of the clocks for `Module_RNDnumberGenerator`, depending on the speed of the ball. If the clock is too fast, the point can be counted twice, if too slow, the point can be lost.

In input, the maximum score `max_score = 8'd15` is given to provide the end of the game, whose signal exits through the wire `win`. When the game ends, it is possible to restart by pushing the `south` button (`new_game` wire), which resets the score (lines 26-27). Also the parameters of the ball and the bar feed the module, because they are necessary to call a variable `caught` which is 1 when the ball hits the bar, and 0 otherwise (line 11 of `Module_Score` code). The reset of the points happens when the ball is not caught by the bar.




```
module Module_Score  (  input clk,
                        input new_game,
                        input [7:0]  max_score,
                        input [9:0]  Xball, Yball,
                        input [9:0]  barMovement,
                        input [10:0] radius, bar_lenght,

                        output reg [7:0] score,
                        output reg reset,
                        output caught, win);
assign caught = (((Xball-radius) >= (barMovement-radius)) && ((Xball+radius) <= (barMovement + bar_lenght)) && (Yball >= 10'd455));
assign win = (score >= max_score);

always @(posedge clk) begin
    if (!win) begin
        if (caught) begin    //Point
            score <= score + 8'd1;
            reset <= 0;
        end else if (((((Xball-radius) < (barMovement)) || ((Xball+radius) > (barMovement + bar_lenght))) && (Yball >= 10'd455))) begin  // reset
            score <= 8'd0;
            reset <= 1;
        end
    end else if (win) begin    //When the player wins, reaching max_score
        if (!new_game) begin
            score <= (8'b11111111);
        end else if (new_game) begin
            score <= 8'd0;
        end
    end
end
endmodule
```

The points are showed by the eight LEDs provided on the FPGA board and by the screen via the module `Module_Score_Numbers`, in which the numbers are defined by imposing the correct boundary conditions on the (`H_coord,V_coord`) coordinates:

```
module Module_Score_Numbers ( input clk,
                              input [7:0] score,
                              input [9:0] H_Coord, V_Coord,

                              output reg points_display);
```

```
reg zero, zero2, one, one2, two, two2, three, three2, four, four2, five, five2, six, six2, seven, seven2, eight, eight2, nine, nine2;
parameter shift = 10'd20;

always @(posedge clk) begin
    //zero
    if ((V_Coord >= 10'd15 && V_Coord <= 10'd18) || (V_Coord >= 10'd26 && V_Coord <= 10'd29)) begin
        zero = (H_Coord >= 10'd11) && (H_Coord <= 10'd22);
    end else if (V_Coord >= 10'd19 && V_Coord <= 10'd25) begin
        zero = ((H_Coord >= 10'd10) && (H_Coord <= 10'd14) || (H_Coord >= 10'd19) && (H_Coord <= 10'd23));
    end
    //one
    if ((V_Coord >= 10'd15 && V_Coord <= 10'd29)) begin
        one = (H_Coord >= 10'd15) && (H_Coord <= 10'd18);
    end
    //two
    if ((V_Coord >= 10'd15 && V_Coord <= 10'd18) || (V_Coord >= 10'd21 && V_Coord <= 10'd23) || (V_Coord >= 10'd26 && V_Coord <= 10'd29)) begin
        two = (H_Coord >= 10'd10) && (H_Coord <= 10'd22);
    end else if (V_Coord >= 10'd19 && V_Coord <= 10'd20) begin
        two = ((H_Coord >= 10'd20) && (H_Coord <= 10'd22));
    end else if (V_Coord >= 10'd24 && V_Coord <= 10'd25) begin
        two = ((H_Coord >= 10'd10) && (H_Coord <= 10'd12));
    end
    //three
    if ((V_Coord >= 10'd15 && V_Coord <= 10'd18) || (V_Coord >= 10'd21 && V_Coord <= 10'd23) || (V_Coord >= 10'd26 && V_Coord <= 10'd29)) begin
        three = (H_Coord >= 10'd10) && (H_Coord <= 10'd22);
    end else if (V_Coord >= 10'd19 && V_Coord <= 10'd20) begin
        three = ((H_Coord >= 10'd20) && (H_Coord <= 10'd22));
    end else if (V_Coord >= 10'd24 && V_Coord <= 10'd25) begin
        three = ((H_Coord >= 10'd20) && (H_Coord <= 10'd22));
    end
    //four
    if (V_Coord >= 10'd21 && V_Coord <= 10'd23) begin
        four = (H_Coord >= 10'd10) && (H_Coord <= 10'd22);
    end else if (V_Coord >= 10'd15 && V_Coord <= 10'd20) begin
        four = ((H_Coord >= 10'd10) && (H_Coord <= 10'd14) || (H_Coord >= 10'd20) && (H_Coord <= 10'd22));
    end else if (V_Coord >= 10'd24 && V_Coord <= 10'd29) begin
        four = ((H_Coord >= 10'd20) && (H_Coord <= 10'd22));
    end
    //five
    if ((V_Coord >= 10'd15 && V_Coord <= 10'd18) || (V_Coord >= 10'd21 && V_Coord <= 10'd23) || (V_Coord >= 10'd26 && V_Coord <= 10'd29)) begin
        five = (H_Coord >= 10'd10) && (H_Coord <= 10'd22);
    end else if (V_Coord >= 10'd24 && V_Coord <= 10'd25) begin
        five = ((H_Coord >= 10'd20) && (H_Coord <= 10'd22));
    end else if (V_Coord >= 10'd19 && V_Coord <= 10'd20) begin
        five = ((H_Coord >= 10'd10) && (H_Coord <= 10'd12));
    end
    //six
    if ((V_Coord >= 10'd15 && V_Coord <= 10'd18) || (V_Coord >= 10'd21 && V_Coord <= 10'd23) || (V_Coord >= 10'd26 && V_Coord <= 10'd29)) begin
        six = (H_Coord >= 10'd10) && (H_Coord <= 10'd22);
    end else if (V_Coord >= 10'd24 && V_Coord <= 10'd25) begin
        six = ((H_Coord >= 10'd20) && (H_Coord <= 10'd22) || (H_Coord >= 10'd10) && (H_Coord <= 10'd12));
    end else if (V_Coord >= 10'd19 && V_Coord <= 10'd20) begin
        six = ((H_Coord >= 10'd10) && (H_Coord <= 10'd12));
    end
    //seven
    if ((V_Coord >= 10'd15 && V_Coord <= 10'd18)) begin
        seven = (H_Coord >= 10'd11) && (H_Coord <= 10'd22);
```

```
    end else if (V_Coord >= 10'd19 && V_Coord <= 10'd29) begin
        seven = ((H_Coord >= 10'd19) && (H_Coord <= 10'd22));
    end
    //eight
    if ((V_Coord >= 10'd15 && V_Coord <= 10'd18) || (V_Coord >= 10'd21 && V_Coord <= 10'd23) || (V_Coord >= 10'd26 && V_Coord <= 10'd29)) begin
        eight = (H_Coord >= 10'd10) && (H_Coord <= 10'd22);
    end else if (V_Coord >= 10'd19 && V_Coord <= 10'd20) begin
        eight = ((H_Coord >= 10'd20) && (H_Coord <= 10'd22) || (H_Coord >= 10'd10) && (H_Coord <= 10'd12));
    end else if (V_Coord >= 10'd24 && V_Coord <= 10'd25) begin
        eight = ((H_Coord >= 10'd20) && (H_Coord <= 10'd22) || (H_Coord >= 10'd10) && (H_Coord <= 10'd12));
    end
    //nine
    if ((V_Coord >= 10'd15 && V_Coord <= 10'd18) || (V_Coord >= 10'd21 && V_Coord <= 10'd23) || (V_Coord >= 10'd26 && V_Coord <= 10'd29)) begin
        nine = (H_Coord >= 10'd10) && (H_Coord <= 10'd22);
    end else if (V_Coord >= 10'd24 && V_Coord <= 10'd25) begin
        nine = ((H_Coord >= 10'd20) && (H_Coord <= 10'd22));
    end else if (V_Coord >= 10'd19 && V_Coord <= 10'd20) begin
        nine = ((H_Coord >= 10'd20) && (H_Coord <= 10'd22) || (H_Coord >= 10'd10) && (H_Coord <= 10'd12));
    end
    //traslated numbers for the second digit
    if ((V_Coord >= 10'd15 && V_Coord <= 10'd18) || (V_Coord >= 10'd26 && V_Coord <= 10'd29)) begin
        zero2 = (H_Coord >= 10'd11 + shift) && (H_Coord <= 10'd22 + shift);
    end else if (V_Coord >= 10'd19 && V_Coord <= 10'd25) begin
        zero2 = ((H_Coord >= 10'd10 + shift) && (H_Coord <= 10'd14 + shift) || (H_Coord >= 10'd19 + shift) && (H_Coord <= 10'd23 + shift));
    end
    //one
    if ((V_Coord >= 10'd15 && V_Coord <= 10'd29)) begin
        one2 = (H_Coord >= 10'd15 + shift) && (H_Coord <= 10'd18 + shift);
    end
    //two
    if ((V_Coord >= 10'd15 && V_Coord <= 10'd18) || (V_Coord >= 10'd21 && V_Coord <= 10'd23) || (V_Coord >= 10'd26 && V_Coord <= 10'd29)) begin
        two2 = (H_Coord >= 10'd10 + shift) && (H_Coord <= 10'd22 + shift);
    end else if (V_Coord >= 10'd19 && V_Coord <= 10'd20) begin
        two2 = ((H_Coord >= 10'd20 + shift) && (H_Coord <= 10'd22 + shift));
    end else if (V_Coord >= 10'd24 && V_Coord <= 10'd25) begin
        two2 = ((H_Coord >= 10'd10 + shift) && (H_Coord <= 10'd12 + shift));
    end
    //three
    if ((V_Coord >= 10'd15 && V_Coord <= 10'd18) || (V_Coord >= 10'd21 && V_Coord <= 10'd23) || (V_Coord >= 10'd26 && V_Coord <= 10'd29)) begin
        three2 = (H_Coord >= 10'd10 + shift) && (H_Coord <= 10'd22 + shift);
    end else if (V_Coord >= 10'd19 && V_Coord <= 10'd20) begin
        three2 = ((H_Coord >= 10'd20 + shift) && (H_Coord <= 10'd22 + shift));
    end else if (V_Coord >= 10'd24 && V_Coord <= 10'd25) begin
        three2 = ((H_Coord >= 10'd20 + shift) && (H_Coord <= 10'd22 + shift));
    end
    //four
    if (V_Coord >= 10'd21 && V_Coord <= 10'd23) begin
        four2 = (H_Coord >= 10'd10 + shift) && (H_Coord <= 10'd22 + shift);
    end else if (V_Coord >= 10'd15 && V_Coord <= 10'd20) begin
        four2 = ((H_Coord >= 10'd10 + shift) && (H_Coord <= 10'd14 + shift) || (H_Coord >= shift + 10'd20) && (H_Coord <= 10'd22 + shift));
    end else if (V_Coord >= 10'd24 && V_Coord <= 10'd29) begin
        four2 = ((H_Coord >= 10'd20 + shift) && (H_Coord <= 10'd22 + shift));
    end
end

always @(posedge clk) begin
```

```
    if (score == 8'd0) begin
        points_display = zero;
    end else if (score == 8'd1) begin
        points_display = one;
    end else if (score == 8'd2) begin
        points_display = two;
    end else if (score == 8'd3) begin
        points_display = three;
    end else if (score == 8'd4) begin
        points_display = four;
    end else if (score == 8'd5) begin
        points_display = five;
    end else if (score == 8'd6) begin
        points_display = six;
    end else if (score == 8'd7) begin
        points_display = seven;
    end else if (score == 8'd8) begin
        points_display = eight;
    end else if (score == 8'd9) begin
        points_display = nine;
    end else if (score == 8'd10) begin
        points_display = (one | zero2);
    end else if (score == 8'd11) begin
        points_display = one | one2;
    end else if (score == 8'd12) begin
        points_display = one | two2;
    end else if (score == 8'd13) begin
        points_display = one | three2;
    end else if (score == 8'd14) begin
        points_display = one | four2;
    end
end
endmodule
```

The end of the game results with a "WIN!" written on the screen, defined by the module `Module_Win`. The letters are written in a peculiar font, created by using geometrical shape such as triangles, rectangles and squares:

- The isosceles triangles are build by defining the initial coordinates (`H_start_t, V_start`) and the height (`V_start` and `height` are common for all the letters). The vertex is given by the starting y-position plus the height, while the oblique sides are counted by adding or subtracting the wire `wb_width_t` from `H_start_t`. This wire is the difference between the height and relative y-coordinate, in order to be equal to the height in correspondence of the basis, and 0 in the vertex.

- The right triangles are similar than the isosceles, but with the x-position of one side is constant and equal to `H_start_t`.

- The rectangles are given by a starting position (`H_start_r, V_start`) to which a base and a height are added, respectively.

- The square has the base equal to the height.

```
module Module_Win (   input [9:0] H_Coord, V_Coord,

                      output [4:0] win_letters);
// Common parameters
parameter V_start = 10'd200;
parameter height  = 10'd100;
parameter base    = 10'd20;

// W: two isosceles triangles with the basis up
parameter H_start_t1 = 10'd160;
parameter H_start_t2 = 10'd220;
```

```
wire [9:0] wb_v_end_t1   = V_start + height;
wire [9:0] wb_width_t1   = height - (V_Coord - V_start);
wire [9:0] wb_h_start_t1 = H_start_t1 - wb_width_t1;
wire [9:0] wb_h_end_t1   = H_start_t1 + wb_width_t1;
wire [9:0] wb_v_end_t2   = V_start +  height;
wire [9:0] wb_width_t2   = height - (V_Coord - V_start);
wire [9:0] wb_h_start_t2 = H_start_t2 - wb_width_t2;
wire [9:0] wb_h_end_t2   = H_start_t2 + wb_width_t2;

assign W1 = ((V_Coord >= V_start) && (V_Coord <= wb_v_end_t1) && (H_Coord >=
    wb_h_start_t1) && (H_Coord <= wb_h_end_t1));
assign W2 = ((V_Coord >= V_start) && (V_Coord <= wb_v_end_t2) && (H_Coord >=
    wb_h_start_t2) && (H_Coord <= wb_h_end_t2));

// I: a vertical bar
parameter H_start_r1 = 10'd350;

wire [9:0] wb_v_end_r1 = V_start + height;
wire [9:0] wb_h_end_r1 = H_start_r1 + base;

assign I = ((H_Coord >= H_start_r1) && (H_Coord <= wb_h_end_r1) && (V_Coord >= V_start)
     && (V_Coord <= wb_v_end_r1));

// N: a right triangle + a vertical bar
parameter H_start_t3 = 10'd400;
parameter H_start_r2 = 10'd480;

wire [9:0] wb_v_end_t3   = V_start + height;
wire [9:0] wb_width_t3   = V_Coord - V_start;
wire [9:0] wb_h_start_t3 = H_start_t3;
wire [9:0] wb_h_end_t3   = H_start_t3 + wb_width_t3;
wire [9:0] wb_v_end_r2   = V_start + height;
wire [9:0] wb_h_end_r2   = H_start_r2 + base;

assign N2 = ((H_Coord >= H_start_r2) && (H_Coord <= wb_h_end_r2) && (V_Coord >=V_start)
        && (V_Coord <= wb_v_end_r2));
assign N1 = ((V_Coord >= V_start) && (V_Coord <= wb_v_end_t3) && (H_Coord >=
        wb_h_start_t3) && (H_Coord <= wb_h_end_t3));

// !: a vertical bar + a square
parameter H_start_r3 = 10'd560;

wire [9:0] wb_v_end_r3   = V_start + height - (base + 10'd10);
wire [9:0] wb_h_end_r3   = H_start_r3 + base;
wire [9:0] wb_v_start_s  = wb_v_end_r3 + 10'd10;
wire [9:0] wb_v_end_s    = wb_v_end_r3 + (base + 10'd10);

assign Excl_pnt = ((H_Coord >= H_start_r3) && (H_Coord <= wb_h_end_r3) && (V_Coord >=
        V_start) && (V_Coord <= wb_v_end_r3)) ||
       ((H_Coord >= H_start_r3) && (H_Coord <= wb_h_end_r3) && (V_Coord >= wb_v_start_s)
       && (V_Coord <= wb_v_end_s));

buf(win_letters[4:0],{Excl_pnt,(N2 | N1),I,W2,W1});
endmodule
```

5 Monitor display

Finally, the last needed module produces the signal with the colors and the images to display on the monitor. The clock is the VGA clock at 25 MHz generated in the first step.

```
module Module_Display(  input clk,
                        input Hdisplay, Vdisplay,
                        input ball, bar,
                        input win, points,
```

```
                         input [4:0] win_letters,

                         output reg [3:0] Green, Blue, Red);
always @(posedge clk) begin
   if(Hdisplay && Vdisplay) begin
      if (!win) begin
         if (bar) begin              //Shows the green bar
            Red <= 4'b0000;
            Green <= 4'b1111;
            Blue <= 4'b0000;
         end else if (ball) begin    //Shows the red ball
            Red <= 4'b1111;
            Green <= 4'b0000;
            Blue <= 4'b0000;
         end else if (points) begin //Shows the score
            Red <= 4'b1111;
            Green <= 4'b1111;
            Blue <= 4'b1111;
         end else begin
            Red <= 4'b0000;
            Green <= 4'b0000;
            Blue <= 4'b0000;
         end
      end else if (win) begin
         if (win_letters == 5'b00001) begin      //Shows the W
            Red <= 4'b1111;
            Green <= 4'b1111;
            Blue <= 4'b1111;
         end else if (win_letters == 5'b00010) begin
            Red <= 4'b1111;
            Green <= 4'b1111;
            Blue <= 4'b1111;
         end else if (win_letters == 5'b00100) begin //Shows the I
            Red <= 4'b1111;
            Green <= 4'b1111;
            Blue <= 4'b1111;
         end else if (win_letters == 5'b01000) begin //Shows the N
            Red <= 4'b1111;
            Green <= 4'b1111;
            Blue <= 4'b1111;
         end else if (win_letters == 5'b10000) begin //Shows the !
            Red <= 4'b1111;
            Green <= 4'b1111;
            Blue <= 4'b1111;
         end else                 //Background
            Red <= 4'b0000;
            Green <= 4'b0111;
            Blue <= 4'b0001;
      end
   end else begin
      Red <= 4'b0000;
      Green <= 4'b0000;
      Blue <= 4'b0000;
   end
end
endmodule
```

Top module

Here the top module is reported.

```
`define 10Hz_period     30'd2500000
`define 16Hz_period     30'd1562500
`define 25Hz_period     30'd1200000
`define 31_25Hz_period  30'd900000
```

```
`define  25MHz_period   30'd1

module CatchTheBallGame (  input CLK_50M,
                           input BTN_EAST, BTN_SOUTH, BTN_WEST,

                           output VGA_HSYNC, VGA_VSYNC,
                           output [3:0] VGA_R, VGA_G, VGA_B,
                           output [7:0] LED);

// VGA
wire          w_clock_VGA;
wire          w_Vdisplay;
wire          w_Hdisplay;
wire  [9:0]   wb_HCoord; // 640 -> needs 10 bits
wire  [9:0]   wb_VCoord; // 480 -> needs 9 bits

// Bar
wire          w_right;
wire          w_left;
wire          w_clockBar_31_25Hz;
wire  [9:0]   wb_barMovement;
parameter     bar_lenght = 11'd50;

// Balls
wire          w_clockBall_10Hz;
wire          w_clockBall_16Hz;
wire          w_clockBall_25Hz;
wire  [9:0]   wb_Xball;
wire  [9:0]   wb_Yball;
parameter     radius = 11'd5;

// Score
wire          w_caught;
wire          w_points_display;
wire          w_win;
wire          w_reset;
wire  [4:0]   wb_win_letters;
wire  [7:0]   wb_score;
parameter     max_score = 8'd15;

// VGA sync
Module_FrequencyDivider generator_VGA  (  .clk_in(CLK_50M),
                                          .period(`25MHz_period),

                                          .clk_out(w_clock_VGA));

Module_HorizontalSync   HSync (  .clk(w_clock_VGA),

                                 .H_Sync(VGA_HSYNC),
                                 .H_Coord(wb_HCoord),
                                 .H_Display(w_Hdisplay));

Module_VerticalSync     VSync (  .clk(w_clock_VGA),

                                 .V_Sync(VGA_VSYNC),
                                 .V_Coord(wb_VCoord),
                                 .V_Display(w_Vdisplay));


// Bar position, via a Finite State Machine
Module_FrequencyDivider generator_clockBar31_25 (  .clk_in(CLK_50M),
                                                   .period(`31_25Hz_period),

                                                   .clk_out(w_clockBar_31_25Hz));

Module_FSM_Bar Bar_movements  (  .clk_in(w_clockBar_31_25Hz),
```

```
                                        .reset(w_reset),
                                        .new_game(BTN_SOUTH),
                                        .direction({BTN_WEST,BTN_EAST}),
                                        .dimension(11'd10),

                                        .movement(wb_barMovement));

wire w_bar = ((!w_win)? ((wb_HCoord >= (wb_barMovement)) && (wb_HCoord <= (wb_barMovement + bar_lenght)) && (wb_VCoord >= 10'd455) && (wb_VCoord <10'd465)) : 0);

//Random balls, via a Shift Register
Module_FrequencyDivider generator_clockBall10   (   .clk_in(CLK_50M),
                                                    .period(`10Hz_period),

                                                    .clk_out(w_clockBall_10Hz));

Module_FrequencyDivider generator_clockBall16   (   .clk_in(CLK_50M),
                                                    .period(`16Hz_period),

                                                    .clk_out(w_clockBall_16Hz));

Module_FrequencyDivider generator_clockBall25   (   .clk_in(CLK_50M),
                                                    .period(`25Hz_period),

                                                    .clk_out(w_clockBall_25Hz));

Module_RNDnumberGenerator Ball_generation (   .clk_in((wb_score<=8'd5)? w_clockBall_10Hz : ((wb_score<=8'd10)? w_clockBall_16Hz : w_clockBall_25Hz)),
                                              .reset(w_reset),
                                              .new_game(BTN_SOUTH),
                                              .caught(w_caught),
                                              .radius(radius),

                                              .X_rnd(wb_Xball),
                                              .Y_rnd(wb_Yball));

wire w_ball = ((!w_win)? (((wb_HCoord-wb_Xball)*(wb_HCoord-wb_Xball)) <= (radius*radius-((wb_VCoord-wb_Yball)*(wb_VCoord-wb_Yball)))) : 0);

//Score counter
Module_Score   Score    (   .clk((wb_score<=8'd5)? w_clockBall_10Hz : ((wb_score<=8'd10)? w_clockBall_16Hz : w_clockBall_25Hz)),
                            .new_game(BTN_SOUTH),
                            .max_score(max_score),
                            .Xball(wb_Xball),
                            .radius(radius),
                            .barMovement(wb_barMovement),
                            .bar_lenght(bar_lenght),
                            .Yball(wb_Yball),

                            .reset(w_reset),
                            .caught(w_caught),
                            .win(w_win),
                            .score(wb_score));


buf(LED, wb_score);

Module_Score_Numbers Score_Numbers  (   .clk(w_clock_VGA),
                                        .score(wb_score),
                                        .H_Coord(wb_HCoord),
                                        .V_Coord(wb_VCoord),

                                        .points_display(w_points_display));

```

```
Module_Win     Win_letters      ( .H_Coord(wb_HCoord),
                                  .V_Coord(wb_VCoord),

                                  .win_letters(wb_win_letters));


Module_Display      DisplayOnMonitor   (   .clk(w_clock_VGA),
                                           .Hdisplay(w_Hdisplay),
                                           .Vdisplay(w_Vdisplay),
                                           .ball(w_ball),
                                           .bar(w_bar),
                                           .win(w_win),
                                           .points(w_points_display),
                                           .win_letters(wb_win_letters),

                                           .Green(VGA_G),
                                           .Blue(VGA_B),
                                           .Red(VGA_R));

endmodule
```