Project Technical Data

DOS Mario

Created by Wojciech Gręde and Hubert Górowski

Written in Verilog 2001 – Syntax using Vivado 2016.1

Written for Basys 3 FPGA Board with XC7A35-1CPG236C from Xilinx

Version 1.0 - Issued June 3, 2016

This document is used to give information about algorithms used in the game and to describe game proper behavior.

Project: DOS Mario - milestones	Planned Issue Date:	Date Issued:
Project consult	March 1	March 1
Project plan	March 8	March 7
Create project base	March 15	March 7
Adding the monsters	March 22	Not finished
Programming the game physics	March 29	April 20
Step into keyboard steering	April 5	March 10
Add special blocks and bonuses	April 12	May 25
Add coins and lives indicators	April 19	June 2
Improving game graphics	April 26	May 10
Add music	May 5	April 25
Add additional stages	May 17	Not finished
Testing the project	May 20	June 1
Documentation	May 25	June 2
Releasing the project	May 27	June 3

Project Overview:

The goal of the project was to create Mario game similar to the DOS Mario version published in 1995. The project uses Basys 3 board from Digilent as well as external module for creating VGA signal and module for music amplification. The game requires an external keyboard to be connected to the Basys

3 board. The game is displayed on VGA compatible display, with resolution 640x480 and 24bit color depth. The game is written in Verilog, few Matlab scripts are also used.

External modules:

VGA extension board – uses four PMOD connectors on the Basys 3 board, contains Analog Devices ADV7125KSTZ50 chip. It extends bits per color to 8, so the whole color depth is 24 bit. This module was made by us. The PMOD's connectors are not matched pairs but the VGA clock frequency 25MHz appears to be low enough to not cause any problems.

Music amplifier – uses single LM358 operational amplifier to amplify the signal from Basys 3 and put it on the speaker.

Game rules and description:

The rules are quite simple for someone who played Mario at least ones. At the beginning your player - Mario has three lives. The goal is to go through the whole stage and collect as many coins as possible. You collect coins by collision with them. The board is build from different blocks. Some of them are solid, which means that you can't destroy them, some of them are fragile which means that after collision they disappear. To destroy the block you must hit it with at least half of player body. When you fall into a water you will lose one life, after losing all three lives the game will over. The player has two speeds, greater speed does not cause player to jump higher.

Steering:

CTRL - speed up

ALT – jump

RIGHT and LEFT ARROWS – moving

The BTNC on Basys 3 restarts the Game.

Clocking sources:

The game uses sets of generated clocks for operation:

25MHz for VGA display

50MHz for complex math formulas

100MHz for high speed algorithms

600Hz for very slow tasks

200kHz for keyboard

We used clocking Wizard (5.3) for generation of 25MHz for buffering main 100MHz clock, and custom module for other clocks.

Output Ports:

Our game uses all four external PMOD connectors, below is pinout we use:

JA1 – VGAext Clk

JA4 – Music output

JA7-9 – VGAext synchronizations signals

JB, JC, JXADC – VGAext color bits

For more detailed description please see the .xdc constraint file.

Program structure and used algorithms:

In the program displaying part is written in separation to the game algorithms part. The most important module in the game is GameEngine. This module calculates actual Mario position, communicates with stage decoder, read data from keyboard, count points and Mario lives, gives instruction to stage decoder to delete block in RAM.

Actual board is saved in the RAM, there is a copy in the ROM which is used when restarting the stage or when reseting.

GAME ENGINE:

Mario movement:

Mario movement is divided into vertical and horizontal movement, which are independent form each other. Those are simple state machines that checks if an arrow on keyboard is pressed. Those state machines checks also blocking register which contains data about blocking blocks around Mario.

Background movement:

If player reaches the most right or left allowed position the background starts to move, it moves faster than player, as it was in the DOS game.

Board movement:

It is achieved similarly to the background movement, the main difference is that it moves with the same speed as Mario.

Checking blocks around:

It is written using state machine Every time Mario changes position it checks what kind of blocks are around him. State machine contains of three main states:

- Checking if the block is blocking or not. It update blocking register which is used by Mario movement state machines.
- Saving what kind of blocks are around Mario.
- Checking if the block is a special one. If that certain kind of operation is performed and new block is saved into RAM

Mario lives:

If Mario falls to the water, it finally reaches the lower position possible, a signal is then send to the combinational logic which lower lives by one, a restart_game signal is also send to different state machines and modules. Every module resets parameters to default values, original stage is also loaded into RAM. When every module using restart_game finishes its operation a new game is started.

Mario coins:

When coin_containing_block or regular coin is scored signal is send from state machine. Every time the signal is send number of coins is incremented by one.

KEYBOARD:

We are using keyboard module found in the Internet. We modified it for our purpose. It simply decodes the PS/2 protocol and sends forward information about pressed keys.

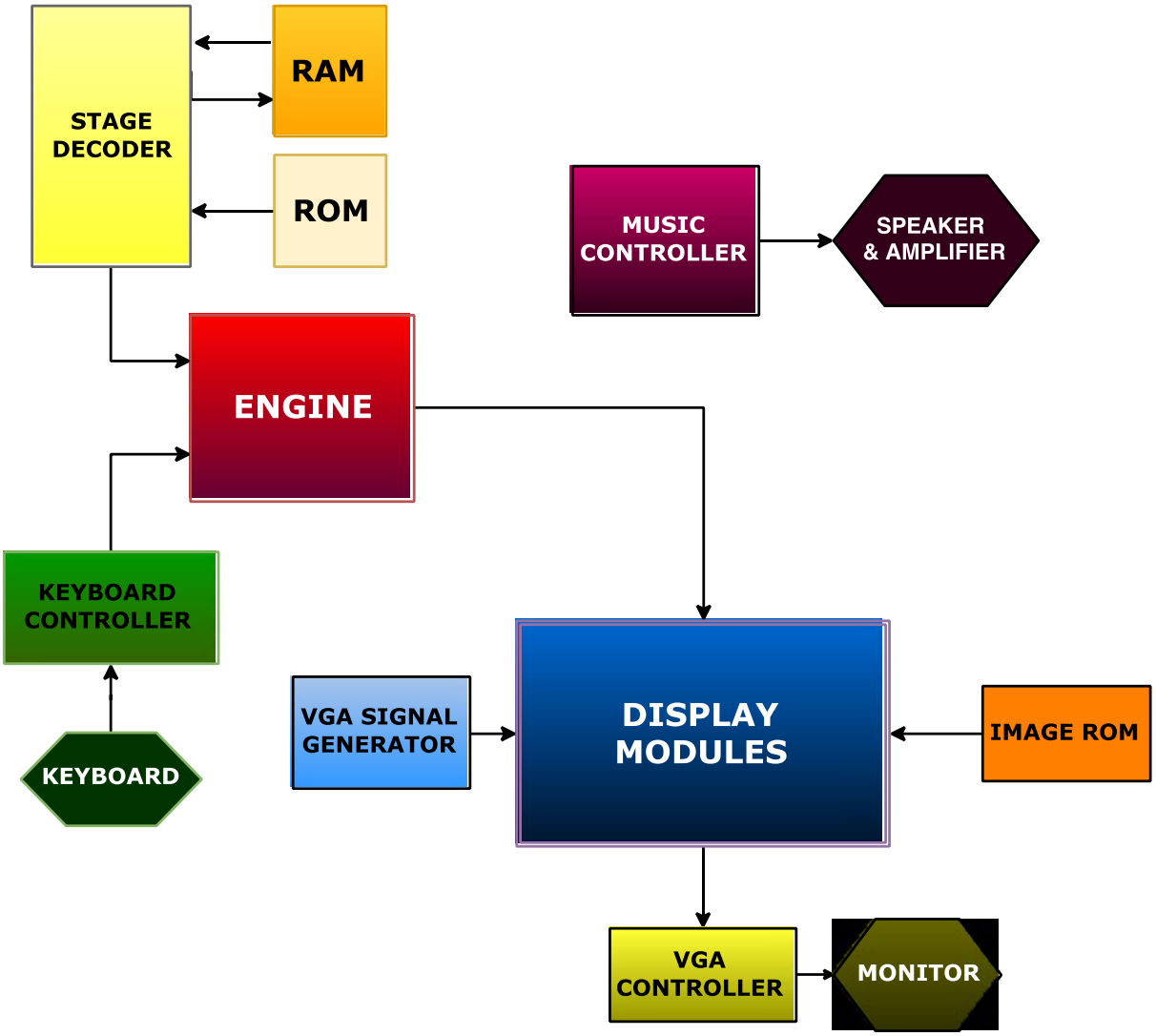
BOARD DISPLAY:

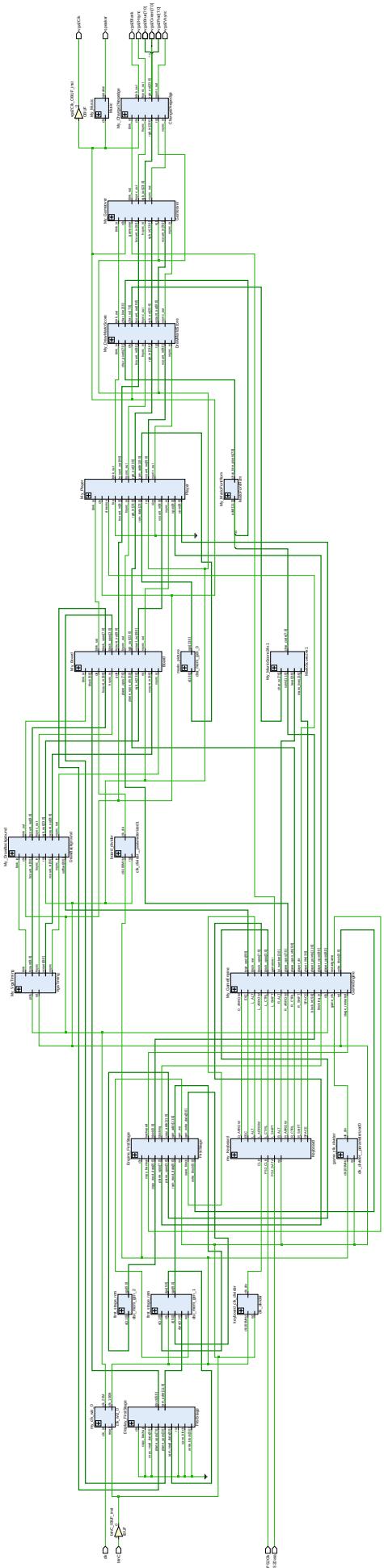
This module gets data from GameEngine about actual board position. Is reads data from ROM and put different colors on the rgb bus depending on what kind of block actual pixel is going through.

The operation of other modules is quite straightforward, comments are also added if needed.

Below we include block diagram, Vivado RTL schematic and code of not-ROM modules.

Block diagram:





```
`timescale 1 ns / 1 ps
//////////////////////////////////////////////////////////////////////////////////
///////
// Company:          AGH UST
// Engineer: Wojciech Gredel, Hubert Górowski
//
// Create Date:
// Design Name:
// Module Name:      Board
// Project Name:     DOS_Mario
// Target Devices:   Basys3
// Tool versions:    Vivado 2016.1
// Description:
//    This module displays board: blocks, special blocks, coins and
everything saved in stage RAM.
//
// Dependencies:
//
// Revision:
// Revision 0.01 - Module created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
///////

module Board (
    input wire [9:0] hcount_in,
    input wire hsync_in,
    input wire [9:0] vcount_in,
    input wire vsync_in,
    input wire pclk,
    input wire [23:0] rgb_in,
    input wire blnk_in,
    input wire clk,
    input wire rst,
    input wire [7:0] plane_xpos,
    input wire [5:0] plane_xpos_ofs,
    input wire [5:0] block,

    output reg [7:0] block_xpos,
    output reg [3:0] block_ypos,
    output reg [9:0] hcount_out,
    output reg hsync_out,
    output reg [9:0] vcount_out,
    output reg vsync_out,
    output reg [23:0] rgb_out,
    output reg blnk_out
    );
    //Definition of blocks, makes it easier to use
    localparam  A   =   1 ;
    localparam  B   =   0 ;
    localparam  C   =   2 ;
    localparam  D   =   3 ;
    localparam  E   =   4 ;
    localparam  F   =   5 ;
    localparam  G   =   6 ;
    localparam  H   =   7 ;
    localparam  I   =   8 ;
    localparam  J   =   9 ;
    localparam  K   =   10;
    localparam  L   =   11;
```

```
    localparam  M   =   12;
    localparam  N   =   13;
    localparam  O   =   14;
    localparam  P   =   15;
    localparam  Q   =   16;
    localparam  R   =   17;
    localparam  S   =   18;
    localparam  T   =   19;
    localparam  U   =   20;
    localparam  V   =   21;
    localparam  W   =   22;
    localparam  X   =   23;
    localparam  Y   =   24;
    localparam  Z   =   25;
    localparam  AY  =   26;
    localparam  IY  =   27;
    localparam  GY  =   28;
    localparam  KY  =   29;
    localparam  PY  =   30;
    localparam  TY  =   31;
    localparam  UY  =   32;
    localparam  WY  =   33;
    localparam  DY  =   34;

    reg [23:0] rgb_nxt;
    reg [7:0] block_xpos_nxt;
    reg [3:0] block_ypos_nxt;
    reg [11:0] ram_addr_nxt;

    always @* begin
        block_ypos_nxt  = ((479 - vcount_in)>>2)/10;
        block_xpos_nxt  = plane_xpos + ((hcount_in +
plane_xpos_ofs)>>2)/10;
    end

    always @* begin
        case(block)//block colors
            A : rgb_nxt  = 24'h0_20_20 ;
            B : rgb_nxt  = rgb_in;
            C : rgb_nxt  = 24'h6A_3D_1E ;
            D : rgb_nxt  = 24'hBB_77_00 ;
            E : rgb_nxt  = 24'h00_00_00 ;
            F : rgb_nxt  = 24'h00_00_00 ;
            G : rgb_nxt  = 24'h10_50_10 ;
            H : rgb_nxt  = 24'h00_00_00 ;
            I : rgb_nxt  = 24'h00_00_00 ;
            J : rgb_nxt  = 24'h70_10_10 ;
            K : rgb_nxt  = 24'h00_00_00 ;
            L : rgb_nxt  = 24'h6A_3D_1E ;
            M : rgb_nxt  = 24'h6A_3D_1E ;
            N : rgb_nxt  = 24'h6A_3D_1E ;
            O : rgb_nxt  = 24'h50_30_20 ;
            P : rgb_nxt  = 24'h6A_3D_1E ;
            Q : rgb_nxt  = 24'h10_50_10 ;
            R : rgb_nxt  = rgb_in ;
            S : rgb_nxt  = 24'h6A_3D_1E ;
            T : rgb_nxt  = rgb_in ;
            U : rgb_nxt  = 24'h10_50_10 ;
            V : rgb_nxt  = 24'h10_50_10 ;
            W : rgb_nxt  = 24'h00_00_f0 ;
            X : rgb_nxt  = 24'h27_75_02 ;
```

```
            Y : rgb_nxt  = rgb_in ;
            Z : rgb_nxt  = 24'h30_30_f0 ;
            AY: rgb_nxt  = 24'h00_00_00 ;
            GY: rgb_nxt  = 24'hff_ff_00 ;
            KY: rgb_nxt  = 24'h00_00_00 ;
            PY: rgb_nxt  = 24'h00_00_00 ;
            TY: rgb_nxt  = 24'h00_00_00 ;
            WY: rgb_nxt  = 24'hb0_70_20 ;
            DY: rgb_nxt  = 24'h7A_4D_2E ;
            default: rgb_nxt = rgb_in;
        endcase
    end

    always @(posedge clk or posedge rst) begin
        if(rst) begin
            block_xpos  <= #1 0;
            block_ypos  <= #1 0;
        end
        else begin
            block_xpos  <= #1 block_xpos_nxt;
            block_ypos  <= #1 block_ypos_nxt;
        end
    end

    always @(posedge pclk or posedge rst) begin
        if(rst) begin
            rgb_out     <= #1 0;
            hcount_out  <= #1 0;
            hsync_out   <= #1 0;
            vcount_out  <= #1 0;
            vsync_out   <= #1 0;
            blnk_out    <= #1 0;
        end
        else begin
            rgb_out     <= #1 rgb_nxt;
            hcount_out  <= #1 hcount_in;
            hsync_out   <= #1 hsync_in;
            vcount_out  <= #1 vcount_in;
            vsync_out   <= #1 vsync_in;
            blnk_out    <= #1 blnk_in;
        end
    end

endmodule
`timescale 1 ns / 1 ps
//////////////////////////////////////////////////////////////////////////////////
///////
// Company:         AGH UST
// Engineer: Wojciech Gredel, Hubert Górowski
//
// Create Date:
// Design Name:
// Module Name:     Change2Negedge
// Project Name:    DOS_Mario
// Target Devices:  Basys3
// Tool versions:   Vivado 2016.1
// Description:
//    A module which is used to put input data out on negedge of clock
//
// Dependencies:
//
```

```
// Revision:
// Revision 0.01 - Module created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
///////

module Change2Negedge
(
    input wire hsync_in,
    input wire vsync_in,
    input wire blnk_in,
    input wire [23:0] rgb_in,
    input wire clk,
    input wire rst,

    output reg hsync_out,
    output reg vsync_out,
    output reg blnk_out,
    output reg [23:0] rgb_out
    );

    always @(negedge clk or posedge rst) begin
        if(rst) begin
            hsync_out   <= #1 0;
            vsync_out   <= #1 0;
            blnk_out    <= #1 0;
            rgb_out     <= #1 0;
        end
        else begin
            hsync_out   <= #1 hsync_in;
            vsync_out   <= #1 vsync_in;
            blnk_out    <= #1 blnk_in;
            rgb_out     <= #1 rgb_in;
        end
    end

endmodule
`timescale 1 ns / 1 ps
//////////////////////////////////////////////////////////////////////////////////
///////
// Company:         AGH UST
// Engineer: Wojciech Gredel, Hubert Górowski
//
// Create Date:
// Design Name:
// Module Name:     Clouds
// Project Name:    DOS_Mario
// Target Devices:  Basys3
// Tool versions:   Vivado 2016.1
// Description:
//    This module displays clouds
//
// Dependencies:
//
// Revision:
// Revision 0.01 - Module created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
///////
```

```
module Clouds (
      input wire clk,
      input wire rst,
      input wire [9:0] xoffset,
      input wire [9:0] hcount_in,
      input wire hsync_in,
      input wire [9:0] vcount_in,
      input wire vsync_in,
      input wire [23:0] rgb_in,
      input wire blnk_in,

      output reg [9:0] hcount_out,
      output reg hsync_out,
      output reg [9:0] vcount_out,
      output reg vsync_out,
      output reg [23:0] rgb_out,
      output reg blnk_out
      );

      localparam MAX_CLOUDS = 11;
      localparam YOFFSET  = 100;
      localparam XRES = 640;
      localparam YRES = 480;
      localparam CLOUD_COLOR = 24'h88_99_cc;
      localparam reg [10:0] CLOUD_MAP_X [0:10] = { 30, 90, 180, 220, 320,
450, 530, 590, 615, 0, 0};
      localparam reg [10:0] CLOUD_MAP_Y [0:10] = {100,100, 100, 100,  50,
70, 100, 100,  80, 0, 0};
      localparam reg [10:0] CLOUD_MAP_S [0:10] = { 30, 70,  25,  50, 100,
70,  25,  50,  40, 0, 0};

      reg [23:0] rgb_nxt;
      reg [3:0]  i;
      reg [10:0] xpos;

      always @(posedge clk or posedge rst) begin
        if(rst) begin
            hcount_out  <= #1 0;
            hsync_out   <= #1 0;
            vcount_out  <= #1 0;
            vsync_out   <= #1 0;
            rgb_out     <= #1 0;
            blnk_out    <= #1 0;
        end
        else begin
            hcount_out  <= #1 hcount_in;
            hsync_out   <= #1 hsync_in;
            vcount_out  <= #1 vcount_in;
            vsync_out   <= #1 vsync_in;
            rgb_out     <= #1 rgb_nxt;
            blnk_out    <= #1 blnk_in;
        end
      end

    always @* begin
        xpos = (hcount_in + xoffset) % (XRES - 1);
      end

      always @* begin
            if((YRES -1 - vcount_in) < YOFFSET ) rgb_nxt = CLOUD_COLOR;
```

```
            else begin
            //REQUIRES CHANGE BECAUSE OF TAKING TOO MUCH LUTS AND DSP, IT
IS JUST CIRCLE EQUATION
                if(((xpos - CLOUD_MAP_X[0])*(xpos - CLOUD_MAP_X[0]) +
(YRES -1 - vcount_in - CLOUD_MAP_Y[0])*(YRES -1 - vcount_in -
CLOUD_MAP_Y[0]))< CLOUD_MAP_S[0]*CLOUD_MAP_S[0]) rgb_nxt = CLOUD_COLOR;
            else if(((xpos - CLOUD_MAP_X[1])*(xpos - CLOUD_MAP_X[1]) +
(YRES -1 - vcount_in - CLOUD_MAP_Y[1])*(YRES -1 - vcount_in -
CLOUD_MAP_Y[1]))< CLOUD_MAP_S[1]*CLOUD_MAP_S[1]) rgb_nxt = CLOUD_COLOR;
            else if(((xpos - CLOUD_MAP_X[2])*(xpos - CLOUD_MAP_X[2]) +
(YRES -1 - vcount_in - CLOUD_MAP_Y[2])*(YRES -1 - vcount_in -
CLOUD_MAP_Y[2]))< CLOUD_MAP_S[2]*CLOUD_MAP_S[2]) rgb_nxt = CLOUD_COLOR;
            else if(((xpos - CLOUD_MAP_X[3])*(xpos - CLOUD_MAP_X[3]) +
(YRES -1 - vcount_in - CLOUD_MAP_Y[3])*(YRES -1 - vcount_in -
CLOUD_MAP_Y[3]))< CLOUD_MAP_S[3]*CLOUD_MAP_S[3]) rgb_nxt = CLOUD_COLOR;
            else if(((xpos - CLOUD_MAP_X[4])*(xpos - CLOUD_MAP_X[4]) +
(YRES -1 - vcount_in - CLOUD_MAP_Y[4])*(YRES -1 - vcount_in -
CLOUD_MAP_Y[4]))< CLOUD_MAP_S[4]*CLOUD_MAP_S[4]) rgb_nxt = CLOUD_COLOR;
            else if(((xpos - CLOUD_MAP_X[5])*(xpos - CLOUD_MAP_X[5]) +
(YRES -1 - vcount_in - CLOUD_MAP_Y[5])*(YRES -1 - vcount_in -
CLOUD_MAP_Y[5]))< CLOUD_MAP_S[5]*CLOUD_MAP_S[5]) rgb_nxt = CLOUD_COLOR;
            else if(((xpos - CLOUD_MAP_X[6])*(xpos - CLOUD_MAP_X[6]) +
(YRES -1 - vcount_in - CLOUD_MAP_Y[6])*(YRES -1 - vcount_in -
CLOUD_MAP_Y[6]))< CLOUD_MAP_S[6]*CLOUD_MAP_S[6]) rgb_nxt = CLOUD_COLOR;
            else if(((xpos - CLOUD_MAP_X[7])*(xpos - CLOUD_MAP_X[7]) +
(YRES -1 - vcount_in - CLOUD_MAP_Y[7])*(YRES -1 - vcount_in -
CLOUD_MAP_Y[7]))< CLOUD_MAP_S[7]*CLOUD_MAP_S[7]) rgb_nxt = CLOUD_COLOR;
            else if(((xpos - CLOUD_MAP_X[8])*(xpos - CLOUD_MAP_X[8]) +
(YRES -1 - vcount_in - CLOUD_MAP_Y[8])*(YRES -1 - vcount_in -
CLOUD_MAP_Y[8]))< CLOUD_MAP_S[8]*CLOUD_MAP_S[8]) rgb_nxt = CLOUD_COLOR;
            else if(((xpos - CLOUD_MAP_X[9])*(xpos - CLOUD_MAP_X[9]) +
(YRES -1 - vcount_in - CLOUD_MAP_Y[9])*(YRES -1 - vcount_in -
CLOUD_MAP_Y[9]))< CLOUD_MAP_S[9]*CLOUD_MAP_S[9]) rgb_nxt = CLOUD_COLOR;
            else rgb_nxt = rgb_in;
        end
      end

endmodule



`timescale 1 ns / 1 ps
//////////////////////////////////////////////////////////////////////////////////
///////
// Company:         AGH UST
// Engineer: Wojciech Gredel, Hubert Górowski
//
// Create Date:
// Design Name:
// Module Name:     DrawBackground
// Project Name:    DOS_Mario
// Target Devices:  Basys3
// Tool versions:   Vivado 2016.1
// Description:
//    This module displays background
//
// Dependencies:
//
// Revision:
// Revision 0.01 - Module created
// Additional Comments:
```

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

module DrawBackground (
      input wire [9:0] hcount_in,
      input wire hsync_in,
      input wire [9:0] vcount_in,
      input wire vsync_in,
      input wire blnk_in,
      input wire [9:0] xoffset,
      input wire clk,
      input wire rst,

      output reg [9:0] hcount_out,
      output reg hsync_out,
      output reg [9:0] vcount_out,
      output reg vsync_out,
      output reg [23:0] rgb_out,
      output reg blnk_out
 );
    //Colors
      localparam BLACK  = 24'h00_00_00;
      localparam WHITE  = 24'hff_ff_ff;
      //Cords
      localparam OFFSET = 100;
      localparam YRES         = 480;

      reg [7:0] r, g, b;
      reg [23:0] rgb_cl;
      reg [9:0]  hcount_cl, vcount_cl;
      reg hsync_cl, vsync_cl, blnk_cl;

    wire [9:0] hcount_cl_out, vcount_cl_out;
    wire [23:0] rgb_cl_out;

    Clouds My_Clouds(
        .clk(clk),
        .rst(rst),
        .xoffset(xoffset),
        .hcount_in(hcount_cl),
        .hsync_in(hsync_cl),
        .vcount_in(vcount_cl),
        .vsync_in(vsync_cl),
        .rgb_in(rgb_cl),
        .blnk_in(blnk_cl),

        .hcount_out(hcount_cl_out),
        .hsync_out(hsync_cl_out),
        .vcount_out(vcount_cl_out),
        .vsync_out(vsync_cl_out),
        .rgb_out(rgb_cl_out),
        .blnk_out(blnk_cl_out)
      );

      always @(posedge clk or posedge rst) begin
        if(rst) begin
            hcount_cl   <= #1 0;
            hsync_cl    <= #1 0;
            vcount_cl   <= #1 0;
            vsync_cl    <= #1 0;
```

```
            rgb_cl      <= #1 0;
            blnk_cl     <= #1 0;
            hcount_out  <= #1 0;
            hsync_out   <= #1 0;
            vcount_out  <= #1 0;
            vsync_out   <= #1 0;
            rgb_out     <= #1 0;
            blnk_out    <= #1 0;
        end
        else begin
            hcount_cl   <= #1 hcount_in;
            hsync_cl    <= #1 hsync_in;
            vcount_cl   <= #1 vcount_in;
            vsync_cl    <= #1 vsync_in;
            rgb_cl      <= #1 {r,g,b};
            blnk_cl     <= #1 blnk_in;
            hcount_out  <= #1 hcount_cl_out;
            hsync_out   <= #1 hsync_cl_out;
            vcount_out  <= #1 vcount_cl_out;
            vsync_out   <= #1 vsync_cl_out;
            rgb_out     <= #1 rgb_cl_out;
            blnk_out    <= #1 blnk_cl_out;
        end
    end

    //Display gradiented sky
      always @* begin
        if((YRES - vcount_in - 1) < OFFSET) {r,g,b} = WHITE; // any color,
will be overwritten
        else begin
            {b} = 8'hff;
            {r} = (vcount_in >> 1);
            {g} = (vcount_in >> 1);
        end
      end

endmodule
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
///////
// Company:          AGH UST
// Engineer: Wojciech Gredel, Hubert Górowski
//
// Create Date:
// Design Name:
// Module Name:      DrawMarioScore
// Project Name:     DOS_Mario
// Target Devices:   Basys3
// Tool versions:    Vivado 2016.1
// Description:
//    This module displays mario score, game level and mario lives
//
// Dependencies:
//
// Revision:
// Revision 0.01 - Module created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
///////
```

```
module DrawMarioScore(
    input wire clk,
    input wire rst,
    input wire [9:0] hcount_in,
    input wire hsync_in,
    input wire [9:0] vcount_in,
    input wire vsync_in,
    input wire [23:0] rgb_in,
    input wire blnk_in,
    input wire [7:0] char_pixels,

    output reg [9:0] hcount_out,
    output reg hsync_out,
    output reg [9:0] vcount_out,
    output reg vsync_out,
    output reg [23:0] rgb_out,
    output reg blnk_out,
    output reg [7:0] char_xy,
    output reg [3:0] char_line
    );

    reg [23:0] rgb_nxt;

    localparam XPOS     = 40;
    localparam YPOS     = 50;
    localparam WIDTH    = 552;
    localparam HEIGHT   = 16;

    always @(posedge clk or posedge rst) begin
        if(rst) begin
            hcount_out  <= #1 0;
            vcount_out  <= #1 0;
            hsync_out   <= #1 0;
            vsync_out   <= #1 0;
            rgb_out     <= #1 0;
            blnk_out    <= #1 0;
        end
        else begin
            hcount_out  <= #1 hcount_in;
            vcount_out  <= #1 vcount_in;
            hsync_out   <= #1 hsync_in;
            vsync_out   <= #1 vsync_in;
            rgb_out     <= #1 rgb_nxt;
            blnk_out    <= #1 blnk_in;
        end
    end

    always @* begin
        if ((hcount_in >= XPOS) && (hcount_in < XPOS + WIDTH) && (vcount_in
>= YPOS) && (vcount_in < YPOS + HEIGHT) && (char_pixels[(XPOS -
hcount_in)]))
        begin
            if(char_xy == 8'h20)
                rgb_nxt = 24'hff_ff_00;
            else
                rgb_nxt = 24'hff_ff_ff;
        end
        else begin
            rgb_nxt = rgb_in; // pass signal through
        end
    end
```

```
    always @* begin
        char_xy = (hcount_in - XPOS - 1)>>3;
    end

    always @* begin
        char_line = vcount_in - YPOS;
    end

endmodule
`timescale 1 ns / 1 ps
//////////////////////////////////////////////////////////////////////////////////
///////
// Company:          AGH UST
// Engineer: Wojciech Gredel, Hubert Górowski
//
// Create Date:
// Design Name:
// Module Name:      FirstStage
// Project Name:     DOS_Mario
// Target Devices:   Basys3
// Tool versions:    Vivado 2016.1
// Description:
//    A module which is used connect RAM with saved board and other
modules.
//
// Dependencies:
//
// Revision:
// Revision 0.01 - Module created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
///////

module FirstStage
(
    input wire [7:0] plane_xpos,
    input wire [3:0] plane_ypos,
    input wire [5:0] ram_read_data,
    input wire [5:0] copy_read_data,
    input wire [5:0] write_block,
    input wire save_block,
    input wire copy_backup,
    input wire clk,
    input wire rst,

    output reg [11:0] ram_addr,
    output reg [11:0] copy_addr,
    output reg [5:0] block,
    output reg blocking,
    output reg ram_we,
    output reg [5:0] ram_write_data,
    output reg backuped
 );
    localparam  A   =   1 ;
    localparam  B   =   0 ;
    localparam  C   =   2 ;
    localparam  D   =   3 ;
    localparam  E   =   4 ;
```

```
localparam  F   =   5 ;
localparam  G   =   6 ;
localparam  H   =   7 ;
localparam  I   =   8 ;
localparam  J   =   9 ;
localparam  K   =   10;
localparam  L   =   11;
localparam  M   =   12;
localparam  N   =   13;
localparam  O   =   14;
localparam  P   =   15;
localparam  Q   =   16;
localparam  R   =   17;
localparam  S   =   18;
localparam  T   =   19;
localparam  U   =   20;
localparam  V   =   21;
localparam  W   =   22;
localparam  X   =   23;
localparam  Y   =   24;
localparam  Z   =   25;
localparam  AY  =   26;
localparam  IY  =   27;
localparam  GY  =   28;
localparam  KY  =   29;
localparam  PY  =   30;
localparam  TY  =   31;
localparam  UY  =   32;
localparam  WY  =   33;
localparam  DY  =   34;

reg [1:0] state, state_nxt;
localparam NORMAL_MODE  = 2'b00;
localparam START_BACKUP = 2'b01;
localparam BACKUP       = 2'b10;

reg backuped_nxt;
reg [11:0] saving_addr, saving_addr_nxt;
localparam ROM_STAGE_SIZE   = 2160;

always @(posedge clk or posedge rst) begin
    if(rst) begin
        backuped    <= #1 0;
        state       <= #1 START_BACKUP;
        saving_addr <= #1 0;
    end
    else begin
        backuped    <= #1 backuped_nxt;
        state       <= #1 state_nxt;
        saving_addr <= #1 saving_addr_nxt;
    end
end

always @* begin
    case(state)
        NORMAL_MODE: begin
            backuped_nxt    = 0;
            copy_addr       = 0;
            saving_addr_nxt = 0;
            ram_addr = (11-plane_ypos)*180 + plane_xpos;
            if(save_block) begin
```

```
                    ram_write_data = write_block;
                    ram_we  = 1;
                    block   = ram_read_data;
                    blocking= 0;
                end
                else begin
                    ram_write_data = write_block;
                    ram_we  = 0;
                    block   = ram_read_data;
                    if((block == A) || (block == D)|| (block == C)|| (block
== S)|| (block == L)|| (block == N)|| (block == J)|| (block == M)|| (block
== P)|| (block == WY) || (block == DY)) begin
                       blocking = 1;
                    end
                    else begin
                       blocking = 0;
                    end
                end
                if(copy_backup)
                    state_nxt = START_BACKUP;
                else
                    state_nxt = NORMAL_MODE;
            end
            START_BACKUP: begin
                backuped_nxt    = 0;
                copy_addr       = 0;
                saving_addr_nxt = 0;
                ram_addr        = 0;
                ram_write_data  = 0;
                ram_we          = 0;
                block           = 0;
                blocking        = 0;
                state_nxt       = BACKUP;
            end
            BACKUP: begin
                copy_addr       = saving_addr;
                saving_addr_nxt = saving_addr + 1;
                ram_addr        = saving_addr;
                ram_write_data  = copy_read_data;
                if(saving_addr == ROM_STAGE_SIZE) begin
                    backuped_nxt= 1;
                    ram_we      = 0;
                    state_nxt   = NORMAL_MODE;
                end
                else begin
                    backuped_nxt= 0;
                    ram_we      = 1;
                    state_nxt   = BACKUP;
                end
                block   = 0;
                blocking= 0;
            end
            default: begin
                backuped_nxt    = 0;
                copy_addr       = 0;
                saving_addr_nxt = 0;
                ram_addr        = 0;
                ram_write_data  = 0;
                ram_we          = 0;
                block           = 0;
                blocking        = 0;
```

```
                state_nxt       = NORMAL_MODE;
            end
        endcase
    end

endmodule
`timescale 1 ns / 1 ps
//////////////////////////////////////////////////////////////////////////////////
///////
// Company:         AGH UST
// Engineer: Wojciech Gredel, Hubert Górowski
//
// Create Date:
// Design Name:
// Module Name:     MarioScore24x1
// Project Name:    DOS_Mario
// Target Devices:  Basys3
// Tool versions:   Vivado 2016.1
// Description:
// This is the engine module, it controlls:
//  *Mario movement
//  *Mario lives
//  *Background movement
//  *Hitting the blocks
//
//
// Dependencies:
//
// Revision:
// Revision 0.01 - Module created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
///////
module GameEngine
(
    input wire clk,
    input wire game_clk,
    input wire rst,
      input wire L_ALT,
    input wire R_ALT,
    input wire L_CTRL,
    input wire R_CTRL,
    input wire SPACE,
    input wire L_SHIFT,
    input wire R_SHIFT,
    input wire ESC,
    input wire D_ARROW,
    input wire L_ARROW,
    input wire R_ARROW,
    input wire blocking_in,
    input wire [5:0] block_in,
    input wire stage_restarted,

    output reg [7:0] block_xpos,
    output reg [3:0] block_ypos,
    output reg [9:0] bcgr_xpos,
    output reg [7:0] plane_xpos,
    output reg [5:0] plane_xpos_ofs,
    output reg [9:0] player_xpos,
    output reg [8:0] player_ypos,
```

```
    output reg player_dir,
    output reg monster_1_dir,
    output reg [1:0] player_speed,
    output reg gameover,
    output reg [5:0] write_block,
    output reg block_we,
    output reg restartgame,
    output reg [3:0] player_life,
    output reg [11:0] player_points,
    output reg [3:0] lvl_number
);

    always @(posedge clk) begin
        lvl_number <=1 ;
    end

    reg gameover_nxt;
    reg restartgame_nxt;
    reg player_xpos_restarted;
    reg player_ypos_restarted;
    reg bcgr_restarted;
    reg board_restarted;
    reg player_xpos_restarted_nxt;
    reg player_ypos_restarted_nxt;
    reg bcgr_restarted_nxt;
    reg board_restarted_nxt;
    //************************
    //State Machine for player
    //************************

    reg [3:0] player_life_nxt;
    reg [1:0] player_hor_state;
    reg [1:0] player_ver_state;
    reg [1:0] player_hor_state_nxt;
    reg [1:0] player_ver_state_nxt;
    reg [7:0] jump_height;
    reg [7:0] jump_height_nxt;
    reg [9:0] player_xpos_nxt;
    reg [8:0] player_ypos_nxt;
    reg player_dir_nxt;
    reg [1:0] player_speed_nxt;
    localparam PL_STOP = 2'b00;
    localparam PL_RIGHT = 2'b01;
    localparam PL_LEFT = 2'b10;
    localparam PL_JUMP = 2'b01;
    localparam PL_FALL = 2'b10;
    localparam DIR_RIGHT = 1'b0;
    localparam DIR_LEFT = 1'b1;
    localparam PL_STAND = 2'b00;
    localparam PL_WALK = 2'b01;
    localparam PL_RUN = 2'b10;
    localparam PLAYER_WIDTH = 40;
    localparam PLAYER_XPOS_MAX = 480;
    localparam PLAYER_XPOS_MIN = 120;
    localparam PLAYER_YPOS_MAX = 480;

   //Speed
    always @* begin
        if(L_ARROW || R_ARROW) begin
            if(L_CTRL || R_CTRL)
                player_speed_nxt = PL_RUN;
```

```
            else
                player_speed_nxt = PL_WALK;
        end
        else
            player_speed_nxt = PL_STAND;
    end

    //Lifes
    always @(posedge clk or posedge rst) begin
        if(rst) begin
            player_life <= 3;
            gameover <= 0;
            restartgame <= 0;
        end
        else begin
            player_life <= player_life_nxt;
            gameover <= gameover_nxt;
            restartgame <= restartgame_nxt;
        end
    end

    reg [11:0] player_points_nxt;
    reg [11:0] player_points_latched;
    reg [11:0] player_points_latched_nxt;

    //Points
    always @(posedge clk or posedge rst) begin
        if(rst) begin
            player_points <= 0;
            player_points_latched <= 0;
        end
        else begin
            player_points <= player_points_nxt;//player_points_nxt;
            player_points_latched <= player_points_latched_nxt;
        end
    end

    always @* begin
        if((new_point) && (player_points_latched == player_points)) begin
            player_points_nxt = player_points + 1;
            player_points_latched_nxt = player_points_latched;
        end
        else begin
            player_points_nxt = player_points;
            if(new_point)
                player_points_latched_nxt = player_points_latched;
            else
                player_points_latched_nxt = player_points;
        end
    end

    //loosing lifes, gameover
    always @* begin
        if(gameover) begin
            restartgame_nxt = 1;
            player_life_nxt = 0;
            gameover_nxt = 1;
        end
        else if(restartgame) begin
            if((player_xpos_restarted) && (player_ypos_restarted) &&
(board_restarted) && (bcgr_restarted) && (stage_restarted)) begin
```

```
                restartgame_nxt = 0;
                player_life_nxt = player_life;
                gameover_nxt = gameover;
            end
            else begin
                restartgame_nxt = 1;
                player_life_nxt = player_life;
                gameover_nxt = gameover;
            end
        end
        else if(player_ypos == 0) begin
            if(player_life == 1) begin
                gameover_nxt = 1;
                player_life_nxt = 0;
                restartgame_nxt = 1;
            end
            else begin
                player_life_nxt = player_life - 1;
                gameover_nxt = 0;
                restartgame_nxt = 1;
            end
        end
        else begin
            gameover_nxt = 0;
            player_life_nxt = player_life;
            restartgame_nxt = 0;
        end
    end

    //Player horizontal movement
    always @* begin
        if(restartgame) begin
            player_xpos_nxt     = 120;
            player_hor_state_nxt= 0;
            player_dir_nxt      = 0;
            player_xpos_restarted_nxt = 1;
        end
        else begin
            player_xpos_restarted_nxt = 0;
            case(player_hor_state)
                PL_STOP: begin
                    if(L_ARROW) begin
                        if(blocking[1]) player_hor_state_nxt = PL_STOP;
                        else player_hor_state_nxt = PL_LEFT;
                    end
                    else if(R_ARROW) begin
                        if(blocking[0]) player_hor_state_nxt = PL_STOP;
                        else player_hor_state_nxt = PL_RIGHT;
                    end
                    else begin
                        player_hor_state_nxt = PL_STOP;
                    end
                    player_xpos_nxt = player_xpos;
                    player_dir_nxt = player_dir;
                end
                PL_RIGHT: begin
                    if(L_ARROW) begin
                        if(blocking[1]) player_hor_state_nxt = PL_STOP;
                        else player_hor_state_nxt = PL_LEFT;
                    end
                    else if(R_ARROW) begin
```

```
                        if(blocking[0]) player_hor_state_nxt = PL_STOP;
                        else player_hor_state_nxt = PL_RIGHT;
                    end
                    else begin
                        player_hor_state_nxt = PL_STOP;
                    end
                    if(((player_xpos) < PLAYER_XPOS_MAX)&&(blocking[0]==0))
begin
                        player_xpos_nxt = player_xpos + 1;
                        player_dir_nxt = DIR_RIGHT;
                    end
                    else begin
                        player_xpos_nxt = player_xpos;
                        player_dir_nxt = DIR_RIGHT;
                    end
                end
                PL_LEFT: begin
                    if(L_ARROW) begin
                        if(blocking[1]) player_hor_state_nxt = PL_STOP;
                        else player_hor_state_nxt = PL_LEFT;
                    end
                    else if(R_ARROW) begin
                        if(blocking[0]) player_hor_state_nxt = PL_STOP;
                        else player_hor_state_nxt = PL_RIGHT;
                    end
                    else begin
                        player_hor_state_nxt = PL_STOP;
                    end
                    if(((player_xpos) > PLAYER_XPOS_MIN)&&(blocking[1] ==
0)) begin
                        player_xpos_nxt = player_xpos - 1;
                        player_dir_nxt = DIR_LEFT;
                    end
                    else begin
                        player_xpos_nxt = player_xpos;
                        player_dir_nxt = DIR_LEFT;
                    end
                end
                default: begin
                    player_hor_state_nxt = PL_STOP;
                    player_xpos_nxt = player_xpos;
                    player_dir_nxt = player_dir;
                end
            endcase
        end
    end

    //Player vertical movement
    always @* begin
        if(restartgame) begin
            player_ypos_nxt     = 100;
            player_ver_state_nxt= 0;
            jump_height_nxt     = 0;
            player_ypos_restarted_nxt = 1;
        end
        else begin
        player_ypos_restarted_nxt = 0;
            case(player_ver_state)
                PL_STOP: begin
                    if(L_ALT || R_ALT) begin
                        if(blocking[2])
```

```
                        player_ver_state_nxt = PL_STOP;
                    else
                        player_ver_state_nxt = PL_JUMP;
                end
                else if(blocking[3] == 0)
                    player_ver_state_nxt = PL_FALL;
                else
                    player_ver_state_nxt = PL_STOP;

                player_ypos_nxt = player_ypos;
                jump_height_nxt = 0;
            end
            PL_JUMP: begin
                if((jump_height < 200) && (blocking[2] == 0)) begin
                    player_ver_state_nxt = PL_JUMP;
                    jump_height_nxt = jump_height + 1;
                    if(player_ypos == PLAYER_YPOS_MAX)
                        player_ypos_nxt = player_ypos;
                    else
                        player_ypos_nxt = player_ypos + 1;
                end
                else begin
                    player_ver_state_nxt = PL_FALL;
                    jump_height_nxt = jump_height;
                    player_ypos_nxt = player_ypos;
                end
            end
            PL_FALL: begin
                if((blocking[3] == 0)) begin
                    player_ver_state_nxt = PL_FALL;
                    jump_height_nxt = jump_height - 1;
                    player_ypos_nxt = player_ypos - 1;
                end
                else begin
                    player_ver_state_nxt = PL_STOP;
                    jump_height_nxt = jump_height;
                    player_ypos_nxt = player_ypos;
                end
            end
            default: begin
                player_ver_state_nxt = PL_STOP;
                jump_height_nxt = jump_height;
                player_ypos_nxt = player_ypos;
            end
        endcase
    end
end

always @(posedge game_clk or posedge rst) begin
    if(rst) begin
        player_ypos     <= 100;
        player_ver_state<= 0;
        jump_height     <= 0;
        player_ypos_restarted <= 0;
    end
    else begin
        player_ypos     <= player_ypos_nxt;
        player_ver_state<= player_ver_state_nxt;
        jump_height     <= jump_height_nxt;
        player_ypos_restarted <= player_ypos_restarted_nxt;
    end
```

```
    end

    reg [1:0] clk_hor_divider;
    always @(posedge game_clk or posedge rst) begin
        if(rst) begin
            player_xpos_restarted <= 0;
            player_xpos     <= 120;
            player_hor_state<= 0;
            player_dir      <= 0;
            clk_hor_divider <= 2'b00;
        end
        else begin
            if((clk_hor_divider == 2'b01)&&(player_speed == PL_RUN)) begin
                player_xpos     <= player_xpos_nxt;
                player_hor_state<= player_hor_state_nxt;
                player_dir      <= player_dir_nxt;
                clk_hor_divider <= 2'b10;
                player_xpos_restarted <= 0;
            end
            else if(clk_hor_divider == 2'b11) begin
                player_xpos     <= player_xpos_nxt;
                player_hor_state<= player_hor_state_nxt;
                player_dir      <= player_dir_nxt;
                clk_hor_divider <= 2'b00;
                player_xpos_restarted <= player_xpos_restarted_nxt;
            end
            else
                clk_hor_divider <= clk_hor_divider + 1;
        end
    end

    always @(posedge game_clk or posedge rst) begin
        if(rst)
            player_speed    <= PL_STOP;
        else
            if(clk_hor_divider == 2'b11)
                player_speed    <= player_speed_nxt;
    end
    //*******************************
    // End of State Machine for player
    //*******************************


    //***********************************************
    // State Machine for board and background movement (uses player states
because depends on player movement)
    //***********************************************
    reg [9:0] bcgr_xpos_nxt;
    localparam XRES = 640;

    always @* begin
        if(restartgame) begin
            bcgr_xpos_nxt = 0;
            bcgr_restarted_nxt = 1;
        end
        else begin
        bcgr_restarted_nxt = 0;
            case(player_hor_state)
                PL_STOP:
                    bcgr_xpos_nxt = bcgr_xpos;
                PL_RIGHT:
```

```
                    if((player_xpos == PLAYER_XPOS_MAX)&&((plane_xpos_ofs
!= MAX_OFFSET)||(plane_xpos != BOARD_END)))
                        bcgr_xpos_nxt = (bcgr_xpos + 2*player_speed) %
(XRES - 1);
                    else
                        bcgr_xpos_nxt = bcgr_xpos;
                PL_LEFT:
                    if((player_xpos == PLAYER_XPOS_MIN)&&((plane_xpos_ofs
!= 0)||(plane_xpos != 0)))
                        if(bcgr_xpos < 2*player_speed)
                            bcgr_xpos_nxt = ((XRES-1) + bcgr_xpos -
2*player_speed);
                        else
                            bcgr_xpos_nxt = bcgr_xpos - 2*player_speed;
                    else
                       bcgr_xpos_nxt = bcgr_xpos;
                default:
                    bcgr_xpos_nxt = bcgr_xpos;
            endcase
        end
    end

    reg [7:0] plane_xpos_nxt;
    reg [5:0] plane_xpos_ofs_nxt;
    localparam MAX_OFFSET = 39;
    localparam BOARD_END = 163;

    always @* begin
        if(restartgame) begin
            plane_xpos_ofs_nxt = 0;
            plane_xpos_nxt = 0;
            board_restarted_nxt = 1;
        end
        else begin
            board_restarted_nxt = 0;
            case(player_hor_state)
                PL_STOP: begin
                    plane_xpos_ofs_nxt = plane_xpos_ofs;
                    plane_xpos_nxt = plane_xpos;
                end
                PL_RIGHT:
                    if(player_xpos == PLAYER_XPOS_MAX) begin
                        if(blocking[0] == 0) begin
                            if(plane_xpos_ofs == MAX_OFFSET) begin
                                if(plane_xpos == BOARD_END) begin
                                    plane_xpos_ofs_nxt = MAX_OFFSET;
                                    plane_xpos_nxt = BOARD_END;
                                end
                                else begin
                                    plane_xpos_ofs_nxt = 0;
                                    plane_xpos_nxt = plane_xpos + 1;
                                end
                            end
                            else begin
                                plane_xpos_ofs_nxt = plane_xpos_ofs + 1;
                                plane_xpos_nxt = plane_xpos;
                            end
                        end
                        else begin
                            plane_xpos_ofs_nxt = plane_xpos_ofs;
                            plane_xpos_nxt = plane_xpos;
```

```
                    end
                end
                else begin
                    plane_xpos_ofs_nxt = plane_xpos_ofs;
                    plane_xpos_nxt = plane_xpos;
                end
            PL_LEFT: begin
                if(player_xpos == PLAYER_XPOS_MIN) begin
                    if(blocking[1] == 0) begin
                        if(plane_xpos_ofs == 0) begin
                            if(plane_xpos == 0) begin
                                plane_xpos_ofs_nxt = 0;
                                plane_xpos_nxt = 0;
                            end
                            else begin
                                plane_xpos_ofs_nxt = MAX_OFFSET;
                                plane_xpos_nxt = plane_xpos - 1;
                            end
                        end
                        else begin
                            plane_xpos_ofs_nxt = plane_xpos_ofs - 1;
                            plane_xpos_nxt = plane_xpos;
                        end
                    end
                    else begin
                        plane_xpos_ofs_nxt = plane_xpos_ofs;
                        plane_xpos_nxt = plane_xpos;
                    end
                end
                else begin
                    plane_xpos_ofs_nxt = plane_xpos_ofs;
                    plane_xpos_nxt = plane_xpos;
                end
            end
            default: begin
                plane_xpos_ofs_nxt = plane_xpos_ofs;
                plane_xpos_nxt = plane_xpos;
            end
        endcase
    end
end

always @(posedge game_clk or posedge rst) begin
    if(rst) begin
        plane_xpos_ofs  <= 0;
        plane_xpos      <= 0;
        board_restarted <= 0;
    end
    else begin
        if((clk_hor_divider == 2'b01)&&(player_speed == PL_RUN)) begin
            plane_xpos_ofs  <= plane_xpos_ofs_nxt;
            plane_xpos      <= plane_xpos_nxt;
            board_restarted <= 0;
        end
        else if(clk_hor_divider == 2'b11) begin
            plane_xpos_ofs  <= plane_xpos_ofs_nxt;
            plane_xpos      <= plane_xpos_nxt;
            board_restarted <= board_restarted_nxt;
        end
    end
end
```

```
always @(posedge game_clk or posedge rst) begin
        if(rst) begin
            bcgr_xpos       <= 0;
            bcgr_restarted <= 0;
        end
        else begin
            if(clk_hor_divider == 2'b11) begin
                bcgr_xpos       <= bcgr_xpos_nxt;
                bcgr_restarted <= bcgr_restarted_nxt;
            end
        end
    end
//*********************************************
// End of State Machine for board and background
//*********************************************

//***********************************************************
// Logic for blocking the player and checking special blocks
//***********************************************************
reg [7:0] block_xpos_nxt;
reg [3:0] block_ypos_nxt;
reg [3:0] blocking;//D:U:L:R
reg [3:0] blocking_nxt;
reg [47:0] modi_block; //DL:DR:UL:UR:LD:LU:RD:RU
reg [47:0] modi_block_nxt;
reg [3:0] blocking_state;
reg [3:0] blocking_state_nxt;
reg [1:0] special;
reg [1:0] special_nxt;
reg [5:0] write_block_nxt;
reg block_we_nxt;
reg [1:0] writing_phase;
reg [1:0] writing_phase_nxt;
reg position_changed;
reg position_changed_nxt;
localparam SPECIAL      = 2'b01;
localparam BLOCKING     = 2'b00;
localparam MOD_BLOCK    = 2'b11;
localparam PREPARE   = 4'b0000;
localparam START     = 4'b0001;
localparam DOWN_L    = 4'b0010;
localparam DOWN_R    = 4'b0011;
localparam UP_L      = 4'b0100;
localparam UP_R      = 4'b0101;
localparam LEFT_D    = 4'b0110;
localparam LEFT_U    = 4'b0111;
localparam RIGHT_D   = 4'b1000;
localparam RIGHT_U   = 4'b1001;

//position changed
reg [7:0] old_plane_xpos;
reg [5:0] old_plane_xpos_ofs;
reg [9:0] old_player_xpos;
reg [8:0] old_player_ypos;
reg [7:0] old_plane_xpos_nxt;
reg [5:0] old_plane_xpos_ofs_nxt;
reg [9:0] old_player_xpos_nxt;
reg [8:0] old_player_ypos_nxt;
reg position_changed1;
reg position_changed1_nxt;
```

```
    always @* begin
        old_plane_xpos_nxt = plane_xpos;
        old_plane_xpos_ofs_nxt = plane_xpos_ofs;
        old_player_xpos_nxt = player_xpos;
        old_player_ypos_nxt = player_ypos;
        if((old_plane_xpos != plane_xpos) || (old_plane_xpos_ofs != 
plane_xpos_ofs) || (old_player_xpos != player_xpos) || (old_player_ypos != 
player_ypos))
            position_changed1_nxt = 1;
         else
            position_changed1_nxt = 0;
    end

    always @(posedge clk or posedge rst) begin
        if(rst) begin
            old_plane_xpos <= 0;
            old_plane_xpos_ofs <= 0;
            old_player_xpos <= 0;
            old_player_ypos <= 0;
            position_changed1 <= 0;
        end
        else begin
            position_changed1 <= position_changed1_nxt;
            old_plane_xpos <= old_plane_xpos_nxt;
            old_plane_xpos_ofs <= old_plane_xpos_ofs_nxt;
            old_player_xpos <= old_player_xpos_nxt;
            old_player_ypos <= old_player_ypos_nxt;
        end
    end

    //rest of blocking
    always @(posedge clk or posedge rst) begin
        if(rst) begin
            special <= 0;
            blocking_state <= PREPARE;
            block_ypos <= 0;
            block_xpos <= 0;
            blocking <= 4'b1000;
            modi_block <= 0;
            write_block <= 0;
            writing_phase <= 2'b00;
            old_block <=0 ;
            position_changed <= 0;
        end
        else begin
            modi_block <= modi_block_nxt;
            special <= special_nxt;
            blocking_state <= blocking_state_nxt;
            block_xpos <= block_xpos_nxt;
            block_ypos <= block_ypos_nxt;
            blocking <= blocking_nxt;
            write_block <= write_block_nxt;
            writing_phase <= writing_phase_nxt;
            old_block <= old_block_nxt;
            position_changed <= position_changed_nxt;
        end
    end

    //And here we are, the code below does the job, it really does
    always @* begin
```

```
        if(position_changed) begin
        case(special)
            BLOCKING: begin
                left_right_block = 0;
                position_changed_nxt = position_changed;
                up_direction = 0;
                writing_phase_nxt = 2'b00;
                write_block_nxt = 0;
                block_we = 0;
                old_block_nxt = 0;
                modi_block_nxt = modi_block;
                case(blocking_state)
                    PREPARE: begin
                        blocking_state_nxt = START;
                        block_xpos_nxt = plane_xpos + (player_xpos +
plane_xpos_ofs)/40;
                        block_ypos_nxt = player_ypos/40;
                        blocking_nxt = blocking;
                        special_nxt = BLOCKING;
                    end
                    START: begin
                        blocking_state_nxt = DOWN_L;
                        block_xpos_nxt = block_xpos;
                        block_ypos_nxt = block_ypos - 1;
                        blocking_nxt = blocking;
                        special_nxt = BLOCKING;
                    end
                    DOWN_L: begin
                        special_nxt = BLOCKING;
                        if((player_xpos + plane_xpos_ofs)%40 == 0) begin
                            blocking_state_nxt = UP_L;
                            block_xpos_nxt = block_xpos;
                            block_ypos_nxt = block_ypos + 2;
                        end
                        else begin
                            block_xpos_nxt = block_xpos + 1;
                            block_ypos_nxt = block_ypos;
                            blocking_state_nxt = DOWN_R;
                        end
                        if(player_ypos % 40 == 0)
                            blocking_nxt[3] = blocking_in;
                        else
                            blocking_nxt[3] = 0;

                        blocking_nxt[2] = blocking[2];
                        blocking_nxt[1] = blocking[1];
                        blocking_nxt[0] = blocking[0];
                    end
                    DOWN_R: begin
                        special_nxt = BLOCKING;
                        blocking_state_nxt = UP_L;
                        block_xpos_nxt = block_xpos - 1;
                        block_ypos_nxt = block_ypos + 2;
                        if(player_ypos % 40 == 0)
                            blocking_nxt[3] = (blocking_in || blocking[3]);
                        else
                            blocking_nxt[3] = 0;

                        blocking_nxt[2] = blocking[2];
                        blocking_nxt[1] = blocking[1];
                        blocking_nxt[0] = blocking[0];
```

```
end
UP_L: begin
    special_nxt = BLOCKING;
    if((player_xpos + plane_xpos_ofs)%40 == 0) begin
        blocking_state_nxt = LEFT_D;
        block_xpos_nxt = block_xpos - 1;
        block_ypos_nxt = block_ypos - 1;
    end
    else begin
        block_xpos_nxt = block_xpos + 1;
        blocking_state_nxt = UP_R;
        block_ypos_nxt = block_ypos;
    end

    blocking_nxt[3] = blocking[3];
    if(player_ypos % 40 == 0)
        blocking_nxt[2] = blocking_in;
    else
        blocking_nxt[2] = 0;

    blocking_nxt[1] = blocking[1];
    blocking_nxt[0] = blocking[0];
end
UP_R: begin
    special_nxt = BLOCKING;
    blocking_state_nxt = LEFT_D;
    block_xpos_nxt = block_xpos - 2;
    block_ypos_nxt = block_ypos - 1;
    blocking_nxt[3] = blocking[3];
    if(player_ypos % 40 == 0)
        blocking_nxt[2] = (blocking_in || blocking[2]);
    else
        blocking_nxt[2] = 0;

    blocking_nxt[1] = blocking[1];
    blocking_nxt[0] = blocking[0];
end
LEFT_D: begin
    special_nxt = BLOCKING;
    if(player_ypos % 40 == 0) begin
        blocking_state_nxt = RIGHT_D;
        block_xpos_nxt = block_xpos + 2;
        block_ypos_nxt = block_ypos;
    end
    else begin
        blocking_state_nxt = LEFT_U;
        block_xpos_nxt = block_xpos;
        block_ypos_nxt = block_ypos + 1;
    end

    blocking_nxt[3] = blocking[3];
    blocking_nxt[2] = blocking[2];
    if((player_xpos + plane_xpos_ofs) % 40 == 0)
        blocking_nxt[1] = blocking_in;
    else
        blocking_nxt[1] = 0;
    blocking_nxt[0] = blocking[0];
end
LEFT_U: begin
    special_nxt = BLOCKING;
    blocking_state_nxt = RIGHT_D;
```

```
                block_xpos_nxt = block_xpos + 2;
                block_ypos_nxt = block_ypos - 1;
                blocking_nxt[3] = blocking[3];
                blocking_nxt[2] = blocking[2];
                if((player_xpos + plane_xpos_ofs) % 40 == 0)
                    blocking_nxt[1] = (blocking_in || blocking[1]);
                else
                    blocking_nxt[1] = 0;
                blocking_nxt[0] = blocking[0];
            end
            RIGHT_D: begin
                if(player_ypos % 40 == 0) begin
                    blocking_state_nxt = PREPARE;
                    special_nxt = SPECIAL;
                    block_xpos_nxt = 0;
                    block_ypos_nxt = 0;
                end
                else begin
                    special_nxt = BLOCKING;
                    blocking_state_nxt = RIGHT_U;
                    block_xpos_nxt = block_xpos;
                    block_ypos_nxt = block_ypos + 1;
                end
                blocking_nxt[3] = blocking[3];
                blocking_nxt[2] = blocking[2];
                blocking_nxt[1] = blocking[1];
                if((player_xpos + plane_xpos_ofs) % 40 == 0)
                    blocking_nxt[0] = blocking_in;
                else
                    blocking_nxt[0] = 0;
            end
            RIGHT_U: begin
                special_nxt = SPECIAL;
                blocking_state_nxt = PREPARE;
                block_xpos_nxt = 0;
                block_ypos_nxt = 0;
                blocking_nxt[3] = blocking[3];
                blocking_nxt[2] = blocking[2];
                blocking_nxt[1] = blocking[1];
                if((player_xpos + plane_xpos_ofs) % 40 == 0)
                    blocking_nxt[0] = (blocking_in || blocking[0]);
                else
                    blocking_nxt[0] = 0;
            end
            default: begin
                blocking_state_nxt = PREPARE;
                block_xpos_nxt = 0;
                block_ypos_nxt = 0;
                blocking_nxt = blocking;
                special_nxt = BLOCKING;
            end
        endcase
    end
    SPECIAL: begin
        left_right_block = 0;
        position_changed_nxt = position_changed;
        up_direction = 0;
        writing_phase_nxt = 2'b00;
        block_we = 0;
        old_block_nxt = 0;
        write_block_nxt = 0;
```

```
                case(blocking_state)
                    PREPARE: begin
                        blocking_state_nxt = START;
                        block_xpos_nxt = plane_xpos + (player_xpos +
plane_xpos_ofs)/40;
                        block_ypos_nxt = player_ypos/40;
                        blocking_nxt = blocking;
                        special_nxt = SPECIAL;
                        modi_block_nxt = 0;
                    end
                    START: begin
                        blocking_state_nxt = DOWN_L;
                        block_xpos_nxt = block_xpos;
                        block_ypos_nxt = block_ypos - 1;
                        blocking_nxt = blocking;
                        special_nxt = SPECIAL;
                        modi_block_nxt = 0;
                    end
                    DOWN_L: begin
                        special_nxt = SPECIAL;
                        blocking_nxt = blocking;
                        if((player_xpos + plane_xpos_ofs)%40 == 0) begin
                            blocking_state_nxt = UP_L;
                            block_xpos_nxt = block_xpos;
                            block_ypos_nxt = block_ypos + 2;
                        end
                        else begin
                            block_xpos_nxt = block_xpos + 1;
                            blocking_state_nxt = DOWN_R;
                            block_ypos_nxt = block_ypos;
                        end
                        if(player_ypos % 40 == 0)
                            modi_block_nxt[47:42] = block_in;
                        else
                            modi_block_nxt[47:42] = 0;

                        modi_block_nxt[41:0] = 0;
                    end
                    DOWN_R: begin
                        special_nxt = SPECIAL;
                        blocking_nxt = blocking;
                        blocking_state_nxt = UP_L;
                        block_xpos_nxt = block_xpos - 1;
                        block_ypos_nxt = block_ypos + 2;
                        if(player_ypos % 40 == 0)
                            modi_block_nxt[41:36] = block_in;
                        else
                            modi_block_nxt[41:36] = 0;

                        modi_block_nxt[47:42] = modi_block[47:42];
                        modi_block_nxt[35:0]  = 0;
                    end
                    UP_L: begin
                        special_nxt = SPECIAL;
                        blocking_nxt = blocking;
                        if((player_xpos + plane_xpos_ofs)%40 == 0) begin
                            blocking_state_nxt = LEFT_D;
                            block_xpos_nxt = block_xpos - 1;
                            block_ypos_nxt = block_ypos - 1;
                        end
                        else begin
```

```
            block_xpos_nxt = block_xpos + 1;
            blocking_state_nxt = UP_R;
            block_ypos_nxt = block_ypos;
        end

        if(player_ypos % 40 == 0)
            modi_block_nxt[35:30] = block_in;
        else
            modi_block_nxt[35:30] = 0;

        modi_block_nxt[47:36] = modi_block[47:36];
        modi_block_nxt[29:0]  = 0;
    end
    UP_R: begin
        special_nxt = SPECIAL;
        blocking_nxt = blocking;
        blocking_state_nxt = LEFT_D;
        block_xpos_nxt = block_xpos - 2;
        block_ypos_nxt = block_ypos - 1;
        if(player_ypos % 40 == 0)
            modi_block_nxt[29:24] = block_in;
        else
            modi_block_nxt[29:24] = 0;

        modi_block_nxt[47:30] = modi_block[47:30];
        modi_block_nxt[23:0]  = 0;
    end
    LEFT_D: begin
        special_nxt = SPECIAL;
        blocking_nxt = blocking;
        if(player_ypos % 40 == 0) begin
            blocking_state_nxt = RIGHT_D;
            block_xpos_nxt = block_xpos + 2;
            block_ypos_nxt = block_ypos;
        end
        else begin
            blocking_state_nxt = LEFT_U;
            block_xpos_nxt = block_xpos;
            block_ypos_nxt = block_ypos + 1;
        end

        if((player_xpos + plane_xpos_ofs) % 40 == 0)
            modi_block_nxt[23:18] = block_in;
        else
            modi_block_nxt[23:18] = 0;
        modi_block_nxt[47:24] = modi_block[47:24];
        modi_block_nxt[17:0]  = 0;
    end
    LEFT_U: begin
        special_nxt = SPECIAL;
        blocking_nxt = blocking;
        blocking_state_nxt = RIGHT_D;
        block_xpos_nxt = block_xpos + 2;
        block_ypos_nxt = block_ypos - 1;
        if((player_xpos + plane_xpos_ofs) % 40 == 0)
            modi_block_nxt[17:12] = block_in;
        else
            modi_block_nxt[17:12] = 0;

        modi_block_nxt[47:18] = modi_block[47:18];
        modi_block_nxt[11:0]  = 0;
```

```
                    end
                    RIGHT_D: begin
                        blocking_nxt = blocking;
                        if(player_ypos % 40 == 0) begin
                            blocking_state_nxt = PREPARE;
                            special_nxt = MOD_BLOCK;
                            block_xpos_nxt = 0;
                            block_ypos_nxt = 0;
                        end
                        else begin
                            special_nxt = SPECIAL;
                            blocking_state_nxt = RIGHT_U;
                            block_xpos_nxt = block_xpos;
                            block_ypos_nxt = block_ypos + 1;
                        end
                        if((player_xpos + plane_xpos_ofs) % 40 == 0)
                            modi_block_nxt[11:6] = block_in;
                        else
                            modi_block_nxt[11:6] = 0;

                        modi_block_nxt[47:12] = modi_block[47:12];
                        modi_block_nxt[5:0]  = 0;
                    end
                    RIGHT_U: begin
                        blocking_nxt = blocking;
                        blocking_state_nxt = PREPARE;
                        special_nxt = MOD_BLOCK;
                        block_xpos_nxt = 0;
                        block_ypos_nxt = 0;
                        if((player_xpos + plane_xpos_ofs) % 40 == 0)
                            modi_block_nxt[5:0] = block_in;
                        else
                            modi_block_nxt[5:0] = block_in;

                        modi_block_nxt[47:6] = modi_block[47:6];
                    end
                    default: begin
                        special_nxt = BLOCKING;
                        blocking_state_nxt = PREPARE;
                        block_xpos_nxt = 0;
                        block_ypos_nxt = 0;
                        blocking_nxt = blocking;
                        modi_block_nxt = modi_block;
                    end
                endcase
            end
            MOD_BLOCK: begin
                case(blocking_state)
                    PREPARE: begin
                        left_right_block = 0;
                        position_changed_nxt = position_changed;
                        up_direction = 0;
                        blocking_state_nxt = START;
                        block_xpos_nxt = plane_xpos + (player_xpos +
plane_xpos_ofs)/40;
                        block_ypos_nxt = player_ypos/40;
                        blocking_nxt = blocking;
                        special_nxt = MOD_BLOCK;
                        modi_block_nxt = modi_block;
                        writing_phase_nxt = 2'b00;
                        old_block_nxt = 0;
```

```
        block_we = 0;
        write_block_nxt = 0;
    end
    START: begin
        left_right_block = 0;
        position_changed_nxt = position_changed;
        up_direction = 0;
        blocking_state_nxt = DOWN_L;
        block_xpos_nxt = block_xpos;
        block_ypos_nxt = block_ypos - 1;
        blocking_nxt = blocking;
        special_nxt = MOD_BLOCK;
        modi_block_nxt = modi_block;
        write_block_nxt = new_block;
        old_block_nxt = modi_block[47:42];
        writing_phase_nxt = 2'b00;
        block_we = 0;
    end
    DOWN_L: begin
        left_right_block = 0;
        position_changed_nxt = position_changed;
        up_direction = 0;
        special_nxt = MOD_BLOCK;
        blocking_nxt = blocking;
        modi_block_nxt = modi_block;
        write_block_nxt = new_block;
        case(writing_phase)
            2'b00: begin
                writing_phase_nxt = 2'b01;
                block_we = new_block_we;
                block_xpos_nxt = block_xpos;
                blocking_state_nxt = blocking_state;
                block_ypos_nxt = block_ypos;
                old_block_nxt = modi_block[47:42];
            end
            2'b01: begin
                writing_phase_nxt = 2'b11;
                block_we = new_block_we;
                block_xpos_nxt = block_xpos;
                blocking_state_nxt = blocking_state;
                block_ypos_nxt = block_ypos;
                old_block_nxt = modi_block[47:42];
            end
            2'b11: begin
                writing_phase_nxt = 2'b00;
                block_we = 0;
                block_xpos_nxt = block_xpos + 1;
                blocking_state_nxt = DOWN_R;
                block_ypos_nxt = block_ypos;
                old_block_nxt = modi_block[41:36];
            end
            default: begin
                writing_phase_nxt = 2'b00;
                block_we = 0;
                block_xpos_nxt = 0;
                blocking_state_nxt = PREPARE;
                block_ypos_nxt = 0;
                old_block_nxt = 0;
            end
        endcase
    end
```

```
DOWN_R: begin
    left_right_block = 1;
    position_changed_nxt = position_changed;
    up_direction = 0;
    special_nxt = MOD_BLOCK;
    blocking_nxt = blocking;
    modi_block_nxt = modi_block;
    write_block_nxt = new_block;
    case(writing_phase)
        2'b00: begin
            writing_phase_nxt = 2'b01;
            block_we = new_block_we;
            block_xpos_nxt = block_xpos;
            blocking_state_nxt = blocking_state;
            block_ypos_nxt = block_ypos;
            old_block_nxt = modi_block[41:36];
        end
        2'b01: begin
            writing_phase_nxt = 2'b11;
            block_we = new_block_we;
            block_xpos_nxt = block_xpos;
            blocking_state_nxt = blocking_state;
            block_ypos_nxt = block_ypos;
            old_block_nxt = modi_block[41:36];
        end
        2'b11: begin
            writing_phase_nxt = 2'b00;
            block_we = 0;
            block_xpos_nxt = block_xpos - 1;
            blocking_state_nxt = UP_L;
            block_ypos_nxt = block_ypos + 2;
            old_block_nxt = modi_block[35:30];
        end
        default: begin
            writing_phase_nxt = 2'b00;
            block_we = 0;
            block_xpos_nxt = 0;
            blocking_state_nxt = PREPARE;
            block_ypos_nxt = 0;
            old_block_nxt = 0;
        end
    endcase
end
UP_L: begin
    left_right_block = 0;
    position_changed_nxt = position_changed;
    up_direction = 1;
    special_nxt = MOD_BLOCK;
    blocking_nxt = blocking;
    modi_block_nxt = modi_block;
    write_block_nxt = new_block;
    case(writing_phase)
        2'b00: begin
            writing_phase_nxt = 2'b01;
            block_we = new_block_we;
            block_xpos_nxt = block_xpos;
            blocking_state_nxt = blocking_state;
            block_ypos_nxt = block_ypos;
            old_block_nxt = modi_block[35:30];
        end
        2'b01: begin
```

```
                writing_phase_nxt = 2'b11;
                block_we = new_block_we;
                block_xpos_nxt = block_xpos;
                blocking_state_nxt = blocking_state;
                block_ypos_nxt = block_ypos;
                old_block_nxt = modi_block[35:30];
            end
            2'b11: begin
                writing_phase_nxt = 2'b00;
                block_we = 0;
                block_xpos_nxt = block_xpos + 1;
                blocking_state_nxt = UP_R;
                block_ypos_nxt = block_ypos;
                old_block_nxt = modi_block[29:24];
            end
            default: begin
                writing_phase_nxt = 2'b00;
                block_we = 0;
                block_xpos_nxt = 0;
                blocking_state_nxt = PREPARE;
                block_ypos_nxt = 0;
                old_block_nxt = 0;
            end
        endcase
    end
    UP_R: begin
        left_right_block = 1;
        position_changed_nxt = position_changed;
        up_direction = 1;
        special_nxt = MOD_BLOCK;
        blocking_nxt = blocking;
        modi_block_nxt = modi_block;
        write_block_nxt = new_block;
        case(writing_phase)
            2'b00: begin
                writing_phase_nxt = 2'b01;
                block_we = new_block_we;
                block_xpos_nxt = block_xpos;
                blocking_state_nxt = blocking_state;
                block_ypos_nxt = block_ypos;
                old_block_nxt = modi_block[29:24];
            end
            2'b01: begin
                writing_phase_nxt = 2'b11;
                block_we = new_block_we;
                block_xpos_nxt = block_xpos;
                blocking_state_nxt = blocking_state;
                block_ypos_nxt = block_ypos;
                old_block_nxt = modi_block[29:24];
            end
            2'b11: begin
                writing_phase_nxt = 2'b00;
                block_we = 0;
                block_xpos_nxt = block_xpos - 2;
                blocking_state_nxt = LEFT_D;
                block_ypos_nxt = block_ypos - 1;
                old_block_nxt = modi_block[23:18];
            end
            default: begin
                writing_phase_nxt = 2'b00;
                block_we = 0;
```

```
                block_xpos_nxt = 0;
                blocking_state_nxt = PREPARE;
                block_ypos_nxt = 0;
                old_block_nxt = 0;
            end
        endcase
    end
    LEFT_D: begin
        left_right_block = 0;
        position_changed_nxt = position_changed;
        up_direction = 0;
        special_nxt = MOD_BLOCK;
        blocking_nxt = blocking;
        modi_block_nxt = modi_block;
        write_block_nxt = new_block;
        case(writing_phase)
            2'b00: begin
                writing_phase_nxt = 2'b01;
                block_we = new_block_we;
                block_xpos_nxt = block_xpos;
                blocking_state_nxt = blocking_state;
                block_ypos_nxt = block_ypos;
                old_block_nxt = modi_block[23:18];
            end
            2'b01: begin
                writing_phase_nxt = 2'b11;
                block_we = new_block_we;
                block_xpos_nxt = block_xpos;
                blocking_state_nxt = blocking_state;
                block_ypos_nxt = block_ypos;
                old_block_nxt = modi_block[23:18];
            end
            2'b11: begin
                writing_phase_nxt = 2'b00;
                block_we = 0;
                block_xpos_nxt = block_xpos;
                blocking_state_nxt = LEFT_U;
                block_ypos_nxt = block_ypos + 1;
                old_block_nxt = modi_block[17:12];
            end
            default: begin
                writing_phase_nxt = 2'b00;
                block_we = 0;
                block_xpos_nxt = 0;
                blocking_state_nxt = PREPARE;
                block_ypos_nxt = 0;
                old_block_nxt = 0;
            end
        endcase
    end
    LEFT_U: begin
        left_right_block = 0;
        position_changed_nxt = position_changed;
        up_direction = 0;
        special_nxt = MOD_BLOCK;
        blocking_nxt = blocking;
        modi_block_nxt = modi_block;
        write_block_nxt = new_block;
        case(writing_phase)
            2'b00: begin
                writing_phase_nxt = 2'b01;
```

```
                block_we = new_block_we;
                block_xpos_nxt = block_xpos;
                blocking_state_nxt = blocking_state;
                block_ypos_nxt = block_ypos;
                old_block_nxt = modi_block[17:12];
            end
            2'b01: begin
                writing_phase_nxt = 2'b11;
                block_we = new_block_we;
                block_xpos_nxt = block_xpos;
                blocking_state_nxt = blocking_state;
                block_ypos_nxt = block_ypos;
                old_block_nxt = modi_block[17:12];
            end
            2'b11: begin
                writing_phase_nxt = 2'b00;
                block_we = 0;
                block_xpos_nxt = block_xpos + 2;
                blocking_state_nxt = RIGHT_D;
                block_ypos_nxt = block_ypos - 1;
                old_block_nxt = modi_block[11:6];
            end
            default: begin
                writing_phase_nxt = 2'b00;
                block_we = 0;
                block_xpos_nxt = 0;
                blocking_state_nxt = PREPARE;
                block_ypos_nxt = 0;
                old_block_nxt = 0;
            end
        endcase
    end
    RIGHT_D: begin
        left_right_block = 1;
        position_changed_nxt = position_changed;
        up_direction = 0;
        special_nxt = MOD_BLOCK;
        blocking_nxt = blocking;
        modi_block_nxt = modi_block;
        write_block_nxt = new_block;
        case(writing_phase)
            2'b00: begin
                writing_phase_nxt = 2'b01;
                block_we = new_block_we;
                block_xpos_nxt = block_xpos;
                blocking_state_nxt = blocking_state;
                block_ypos_nxt = block_ypos;
                old_block_nxt = modi_block[11:6];
            end
            2'b01: begin
                writing_phase_nxt = 2'b11;
                block_we = new_block_we;
                block_xpos_nxt = block_xpos;
                blocking_state_nxt = blocking_state;
                block_ypos_nxt = block_ypos;
                old_block_nxt = modi_block[11:6];
            end
            2'b11: begin
                writing_phase_nxt = 2'b00;
                block_we = 0;
                block_xpos_nxt = block_xpos;
```

```
                blocking_state_nxt = RIGHT_U;
                block_ypos_nxt = block_ypos + 1;
                old_block_nxt = modi_block[5:0];
            end
            default: begin
                writing_phase_nxt = 2'b00;
                block_we = 0;
                block_xpos_nxt = 0;
                blocking_state_nxt = PREPARE;
                block_ypos_nxt = 0;
                old_block_nxt = 0;
            end
        endcase
    end
    RIGHT_U: begin
        left_right_block = 1;
        position_changed_nxt = 0;
        up_direction = 0;
        blocking_nxt = blocking;
        modi_block_nxt = modi_block;
        write_block_nxt = new_block;
        case(writing_phase)
            2'b00: begin
                special_nxt = MOD_BLOCK;
                writing_phase_nxt = 2'b01;
                block_we = new_block_we;
                block_xpos_nxt = block_xpos;
                blocking_state_nxt = blocking_state;
                block_ypos_nxt = block_ypos;
                old_block_nxt = modi_block[5:0];
            end
            2'b01: begin
                special_nxt = MOD_BLOCK;
                writing_phase_nxt = 2'b11;
                block_we = new_block_we;
                block_xpos_nxt = block_xpos;
                blocking_state_nxt = blocking_state;
                block_ypos_nxt = block_ypos;
                old_block_nxt = modi_block[5:0];
            end
            2'b11: begin
                special_nxt = BLOCKING;
                writing_phase_nxt = 2'b00;
                block_we = 0;
                block_xpos_nxt = block_xpos;
                blocking_state_nxt = PREPARE;
                block_ypos_nxt = block_ypos;
                old_block_nxt = old_block;
            end
            default: begin
                special_nxt = BLOCKING;
                writing_phase_nxt = 2'b00;
                block_we = 0;
                block_xpos_nxt = 0;
                blocking_state_nxt = PREPARE;
                block_ypos_nxt = 0;
                old_block_nxt = 0;
            end
        endcase
    end
    default: begin
```

```
                    left_right_block = 0;
                    position_changed_nxt = 0;
                    up_direction = 0;
                    write_block_nxt = 0;
                    special_nxt = BLOCKING;
                    blocking_state_nxt = PREPARE;
                    block_xpos_nxt = 0;
                    block_ypos_nxt = 0;
                    blocking_nxt = blocking;
                    modi_block_nxt = modi_block;
                    writing_phase_nxt = 2'b00;
                    old_block_nxt = 0;
                    block_we = 0;
                end
            endcase
        end
        default: begin
            left_right_block = 0;
            position_changed_nxt = 0;
            up_direction = 0;
            write_block_nxt = 0;
            block_we = 0;
            modi_block_nxt = modi_block;
            special_nxt = BLOCKING;
            blocking_state_nxt = PREPARE;
            block_xpos_nxt = 0;
            block_ypos_nxt = 0;
            blocking_nxt = blocking;
            writing_phase_nxt = 2'b00;
            old_block_nxt = 0;
        end
    endcase
    end
    else begin
        if(position_changed1)
            position_changed_nxt = 1;
        else
            position_changed_nxt = 0;

        left_right_block = 0;
        up_direction = 0;
        write_block_nxt = 0;
        block_we = 0;
        modi_block_nxt = modi_block;
        special_nxt = BLOCKING;
        blocking_state_nxt = PREPARE;
        block_xpos_nxt = 0;
        block_ypos_nxt = 0;
        blocking_nxt = blocking;
        writing_phase_nxt = 2'b00;
        old_block_nxt = 0;
    end
end

//************************************
// End of logic for blocking the player
//************************************

//Breaking the locks and getting points
wire [5:0] new_block;
reg [5:0] old_block, old_block_nxt;
```

```
    reg up_direction, left_right_block;

    new_block my_new_block(
        .block_in(old_block),
        .up_direction(up_direction),
        .direction(left_right_block),
        .block_out(new_block),
        .relative_xpos(player_xpos + plane_xpos_ofs),
        .relative_ypos(player_ypos),
        .write_enable(new_block_we),
        .new_point(new_point)
    );

endmodule
`timescale 1 ns / 1 ps
//////////////////////////////////////////////////////////////////////////////////
///////
// Company:          AGH UST
// Engineer: Wojciech Gredel, Hubert Górowski
//
// Create Date:
// Design Name:
// Module Name:      Gameover
// Project Name:     DOS_Mario
// Target Devices:   Basys3
// Tool versions:    Vivado 2016.1
// Description:
//    This module displays GAME OVER text
//
// Dependencies:
//
// Revision:
// Revision 0.01 - Module created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
///////

module Gameover
(
    input wire gameover,
    input wire [9:0] hcount_in,
    input wire hsync_in,
    input wire [9:0] vcount_in,
    input wire vsync_in,
    input wire [23:0] rgb_in,
    input wire blnk_in,
    input wire clk,
    input wire rst,

    output reg [9:0] hcount_out,
    output reg hsync_out,
    output reg [9:0] vcount_out,
    output reg vsync_out,
    output reg [23:0] rgb_out,
    output reg blnk_out
    );

    localparam LETTER_XS = 11;
    localparam LETTER_YS = 10;
    //first letter
```

```
localparam X1_CENTER = 170;
localparam Y1_CENTER = 180;
localparam SIZE1 = 9;
localparam X1_OFFSET = X1_CENTER + LETTER_XS*SIZE1/2;
localparam Y1_OFFSET = Y1_CENTER + LETTER_YS*SIZE1/2;
//second letter
localparam X2_CENTER = 270;
localparam Y2_CENTER = 180;
localparam SIZE2 = 9;
localparam X2_OFFSET = X2_CENTER + LETTER_XS*SIZE2/2;
localparam Y2_OFFSET = Y2_CENTER + LETTER_YS*SIZE2/2;
//third letter
localparam X3_CENTER = 370;
localparam Y3_CENTER = 180;
localparam SIZE3 = 9;
localparam X3_OFFSET = X3_CENTER + LETTER_XS*SIZE3/2;
localparam Y3_OFFSET = Y3_CENTER + LETTER_YS*SIZE3/2;
//fourth letter
localparam X4_CENTER = 470;
localparam Y4_CENTER = 180;
localparam SIZE4 = 9;
localparam X4_OFFSET = X4_CENTER + LETTER_XS*SIZE4/2;
localparam Y4_OFFSET = Y4_CENTER + LETTER_YS*SIZE4/2;
//fifth letter
localparam X5_CENTER = 170;
localparam Y5_CENTER = 300;
localparam SIZE5 = 9;
localparam X5_OFFSET = X5_CENTER + LETTER_XS*SIZE5/2;
localparam Y5_OFFSET = Y5_CENTER + LETTER_YS*SIZE5/2;
//sixth letter
localparam X6_CENTER = 270;
localparam Y6_CENTER = 300;
localparam SIZE6 = 9;
localparam X6_OFFSET = X6_CENTER + LETTER_XS*SIZE6/2;
localparam Y6_OFFSET = Y6_CENTER + LETTER_YS*SIZE6/2;
//seventh letter
localparam X7_CENTER = 370;
localparam Y7_CENTER = 300;
localparam SIZE7 = 9;
localparam X7_OFFSET = X7_CENTER + LETTER_XS*SIZE7/2;
localparam Y7_OFFSET = Y7_CENTER + LETTER_YS*SIZE7/2;
//eighth letter
localparam X8_CENTER = 470;
localparam Y8_CENTER = 300;
localparam SIZE8 = 9;
localparam X8_OFFSET = X8_CENTER + LETTER_XS*SIZE8/2;
localparam Y8_OFFSET = Y8_CENTER + LETTER_YS*SIZE8/2;

//Letters
localparam reg [10:0] V_LETTER [9:0] ={
    11'b10000000001,
    11'b01000000010,
    11'b01000000010,
    11'b00100000100,
    11'b00100000100,
    11'b00010001000,
    11'b00010001000,
    11'b00001010000,
    11'b00001010000,
    11'b00000100000};
```

```
localparam reg [10:0] A_LETTER [9:0] ={
    11'b00111111100,
    11'b01000000010,
    11'b01000000010,
    11'b01000000010,
    11'b01111111110,
    11'b01000000010,
    11'b01000000010,
    11'b01000000010,
    11'b01000000010,
    11'b01000000010};

localparam reg [10:0] M_LETTER [9:0] ={
    11'b10000000001,
    11'b11000000011,
    11'b10100000101,
    11'b10010001001,
    11'b10001010001,
    11'b10000100001,
    11'b10000000001,
    11'b10000000001,
    11'b10000000001,
    11'b10000000001};

localparam reg [10:0] E_LETTER [9:0] ={
    11'b01111111110,
    11'b01000000000,
    11'b01000000000,
    11'b01000000000,
    11'b01111111110,
    11'b01000000000,
    11'b01000000000,
    11'b01000000000,
    11'b01000000000,
    11'b01111111110};

localparam reg [10:0] O_LETTER [9:0] ={
    11'b00111111100,
    11'b01000000010,
    11'b01000000010,
    11'b01000000010,
    11'b01000000010,
    11'b01000000010,
    11'b01000000010,
    11'b01000000010,
    11'b01000000010,
    11'b00111111100};

localparam reg [10:0] R_LETTER [9:0] ={
    11'b01111111100,
    11'b01000000010,
    11'b01000000010,
    11'b01000000010,
    11'b01111111100,
    11'b01000100000,
    11'b01000010000,
    11'b01000001000,
    11'b01000000100,
    11'b01000000010};

localparam reg [10:0] G_LETTER [9:0] ={
```

```
        11'b00011111000,
        11'b00100000100,
        11'b01000000010,
        11'b01000000010,
        11'b01000000000,
        11'b01000011110,
        11'b01000000010,
        11'b01000000010,
        11'b00100000100,
        11'b00011111000};

    localparam YRES = 480;
    //Colors
    localparam BLACK = 24'h00_00_00;
    localparam WHITE = 24'hff_ff_ff;

    reg [23:0] rgb_nxt = 0;

    always @(posedge clk or posedge rst) begin
        if(rst) begin
            hcount_out  <= #1 0;
            hsync_out   <= #1 0;
            vcount_out  <= #1 0;
            vsync_out   <= #1 0;
            rgb_out     <= #1 0;
            blnk_out    <= #1 0;
        end
        else begin
            hcount_out  <= #1 hcount_in;
            hsync_out   <= #1 hsync_in;
            vcount_out  <= #1 vcount_in;
            vsync_out   <= #1 vsync_in;
            rgb_out     <= #1 rgb_nxt;
            blnk_out    <= #1 blnk_in;
        end
    end

    always @* begin
        if(gameover) begin
             //G
            if(((Y1_OFFSET - vcount_in) <= LETTER_YS*SIZE1 - 1) &&
((Y1_OFFSET - vcount_in) >= 0) && ((X1_OFFSET - hcount_in) <= LETTER_XS *
SIZE1 - 1) && ((X1_OFFSET - hcount_in) >= 0)) begin
                if(G_LETTER[(Y1_OFFSET - vcount_in)/SIZE1][(X1_OFFSET -
hcount_in)/SIZE1] == 1)
                    begin
                        rgb_nxt = WHITE;
                    end
                else rgb_nxt = BLACK;
            end
            //A
            else if(((Y2_OFFSET - vcount_in) <= LETTER_YS*SIZE2 - 1) &&
((Y2_OFFSET - vcount_in) >= 0) && ((X2_OFFSET - hcount_in) <= LETTER_XS *
SIZE2 - 1) && ((X2_OFFSET - hcount_in) >= 0)) begin
                if(A_LETTER[(Y2_OFFSET - vcount_in)/SIZE2][(X2_OFFSET -
hcount_in)/SIZE2] == 1)
                    begin
                        rgb_nxt = WHITE;
                        end
                    else rgb_nxt = BLACK;
            end
```

```
            //M
            else if(((Y3_OFFSET - vcount_in) <= LETTER_YS*SIZE3 - 1) &&
((Y3_OFFSET - vcount_in) >= 0) && ((X3_OFFSET - hcount_in) <= LETTER_XS *
SIZE3 - 1) && ((X3_OFFSET - hcount_in) >= 0)) begin
                if(M_LETTER[(Y3_OFFSET - vcount_in)/SIZE3][(X3_OFFSET -
hcount_in)/SIZE3] == 1)
                    begin
                        rgb_nxt = WHITE;
                        end
                    else rgb_nxt = BLACK;
            end
            //E
            else if(((Y4_OFFSET - vcount_in) <= LETTER_YS*SIZE4 - 1) &&
((Y4_OFFSET - vcount_in) >= 0) && ((X4_OFFSET - hcount_in) <= LETTER_XS *
SIZE4 - 1) && ((X4_OFFSET - hcount_in) >= 0)) begin
                if(E_LETTER[(Y4_OFFSET - vcount_in)/SIZE4][(X4_OFFSET -
hcount_in)/SIZE4] == 1)
                    begin
                        rgb_nxt = WHITE;
                        end
                    else rgb_nxt = BLACK;
            end
            //O
            else if(((Y5_OFFSET - vcount_in) <= LETTER_YS*SIZE5 - 1) &&
((Y5_OFFSET - vcount_in) >= 0) && ((X5_OFFSET - hcount_in) <= LETTER_XS *
SIZE5 - 1) && ((X5_OFFSET - hcount_in) >= 0)) begin
                if(O_LETTER[(Y5_OFFSET - vcount_in)/SIZE5][(X5_OFFSET -
hcount_in)/SIZE5] == 1)
                   begin
                        rgb_nxt = WHITE;
                       end
                   else rgb_nxt = BLACK;
            end
            //V
            else if(((Y6_OFFSET - vcount_in) <= LETTER_YS*SIZE6 - 1) &&
((Y6_OFFSET - vcount_in) >= 0) && ((X6_OFFSET - hcount_in) <= LETTER_XS *
SIZE6 - 1) && ((X6_OFFSET - hcount_in) >= 0)) begin
                if(V_LETTER[(Y6_OFFSET - vcount_in)/SIZE6][(X6_OFFSET -
hcount_in)/SIZE6] == 1)
                    begin
                        rgb_nxt = WHITE;
                        end
                    else rgb_nxt = BLACK;
            end
            //E
            else if(((Y7_OFFSET - vcount_in) <= LETTER_YS*SIZE7 - 1) &&
((Y7_OFFSET - vcount_in) >= 0) && ((X7_OFFSET - hcount_in) <= LETTER_XS *
SIZE7 - 1) && ((X7_OFFSET - hcount_in) >= 0)) begin
                if(E_LETTER[(Y7_OFFSET - vcount_in)/SIZE7][(X7_OFFSET -
hcount_in)/SIZE7] == 1)
                    begin
                        rgb_nxt = WHITE;
                        end
                    else rgb_nxt = BLACK;
            end
            //R
            else if(((Y8_OFFSET - vcount_in) <= LETTER_YS*SIZE8 - 1) &&
((Y8_OFFSET - vcount_in) >= 0) && ((X8_OFFSET - hcount_in) <= LETTER_XS *
SIZE8 - 1) && ((X8_OFFSET - hcount_in) >= 0)) begin
                if(R_LETTER[(Y8_OFFSET - vcount_in)/SIZE8][(X8_OFFSET -
hcount_in)/SIZE8] == 1)
```

```
                    begin
                         rgb_nxt = WHITE;
                        end
                    else rgb_nxt = BLACK;
                end
                else begin
                    rgb_nxt = BLACK;
                end
            end
            else begin
                rgb_nxt = rgb_in;
            end
        end

endmodule
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
///////
// Company:          AGH UST
// Engineer: Montvydas Klumbys, modified by Wojciech Gredel
//
// Create Date:
// Design Name:
// Module Name:      Keyboard
// Project Name:     DOS_Mario
// Target Devices:   Basys3
// Tool versions:    Vivado 2016.1
// Description:
//    A module which is used to receive the DATA from PS2 type keyboard and
translate that data into sensible codeword.
//
// Dependencies:
//
// Revision:
// Revision 0.02 - Added keys differentiation
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
///////

module Keyboard(
      input wire CLK,          //board clock
      input wire PS2_CLK,      //keyboard clock and data signals
      input wire PS2_DATA,
      input wire rst,

      output reg L_ALT,
      output reg R_ALT,
      output reg L_CTRL,
      output reg R_CTRL,
      output reg SPACE,
      output reg L_SHIFT,
      output reg R_SHIFT,
      output reg ESC,
      output reg D_ARROW,
      output reg L_ARROW,
      output reg R_ARROW
      );

    localparam reg [7:0] ALT_CODE      = 8'h11;
    localparam reg [7:0] CTRL_CODE     = 8'h14;
```

```
      localparam reg [7:0] SPACE_CODE    = 8'h29;
    localparam reg [7:0] L_SHIFT_CODE  = 8'h12;
    localparam reg [7:0] R_SHIFT_CODE  = 8'h59;
    localparam reg [7:0] ESC_CODE      = 8'h76;
      localparam reg [7:0] D_ARROW_CODE  = 8'h72;
      localparam reg [7:0] L_ARROW_CODE  = 8'h6B;
      localparam reg [7:0] R_ARROW_CODE  = 8'h74;
    localparam reg [7:0] BREAK_CODE    = 8'hF0;
      localparam reg [7:0] EXTENDED_CODE = 8'hE0;
      localparam reg [7:0] L_GUI         = 8'h1F;
      localparam reg [7:0] R_GUI         = 8'h27;
      localparam reg [7:0] APPS          = 8'h2F;

    reg EXTENDED_nxt;
    reg BREAK_nxt;
      reg L_ALT_nxt;
    reg R_ALT_nxt;
    reg L_CTRL_nxt;
    reg R_CTRL_nxt;
    reg SPACE_nxt;
    reg L_SHIFT_nxt;
    reg R_SHIFT_nxt;
    reg ESC_nxt;
    reg D_ARROW_nxt;
    reg L_ARROW_nxt;
    reg R_ARROW_nxt;

      reg read;                         //this is 1 if still waits to receive
more bits
    reg [11:0] count_reading;  //this is used to detect how much time
passed since it received the previous codeword
      reg PREVIOUS_STATE;        //used to check the previous state of the
keyboard clock signal to know if it changed
      reg [10:0] scan_code;      //this stores 11 received bits
      reg scan_err;              //this tells about error
      reg [7:0] CODEWORD;        //this stores only the DATA codeword
      reg TRIG_ARR;                   //this is triggered when full 11 bits
are received
      reg [3:0]BIT_COUNTER;      //tells how many bits were received until
now (from 0 to 11)
      reg EXTENDED;
      reg BREAK;

      always @(posedge CLK or posedge rst) begin
        if(rst) begin
            count_reading <= #1 0;
        end
        else begin
            if (read)                                                  //if it
still waits to read full packet of 11 bits, then (read == 1)
                count_reading <= #1 count_reading + 1;      //and it counts
up this variable
            else                                                       //and
later if check to see how big this value is.
                count_reading <= #1 0;                                 //if it is
too big, then it resets the received data
        end
      end

      always @(posedge CLK or posedge rst) begin
        if(rst) begin
```

```
            PREVIOUS_STATE  <= #1 1;
            read            <= #1 0;
            scan_err        <= #1 0;
            scan_code       <= #1 11'b00000000000;
            BIT_COUNTER     <= #1 0;
            TRIG_ARR        <= #1 0;
        end
        else begin
            if (PS2_CLK != PREVIOUS_STATE) begin                    //if the
state of Clock pin changed from previous state
                if (!PS2_CLK) begin                     //and if the keyboard
clock is at falling edge
                    read <= #1 1;                       //mark down that it is
still reading for the next bit
                    scan_err <= #1 0;                           //no errors
                    scan_code[10:0] <= #1 {PS2_DATA, scan_code[10:1]};
      //add up the data received by shifting bits and adding one new bit
                    BIT_COUNTER <= #1 BIT_COUNTER + 1;                  //
                end
            end
            else if (BIT_COUNTER == 11) begin                       //if it
already received 11 bits
                BIT_COUNTER <= #1 0;
                read <= #1 0;                           //mark down that
reading stopped
                TRIG_ARR <= #1 1;                               //trigger out
that the full pack of 11bits was received
                //calculate scan_err using parity bit
                if (!scan_code[10] || scan_code[0] ||
!(scan_code[1]^scan_code[2]^scan_code[3]^scan_code[4]
                    ^scan_code[5]^scan_code[6]^scan_code[7]^scan_code[8]
                    ^scan_code[9]))
                    scan_err <= #1 1;
                else
                    scan_err <= #1 0;
            end
            else  begin                                 //if it yet not
received full pack of 11 bits
                TRIG_ARR <= #1 0;                               //tell that the
packet of 11bits was not received yet
                if (BIT_COUNTER < 11 && count_reading >= 4000) begin
      //and if after a certain time no more bits were received, then
                    BIT_COUNTER <= #1 0;                    //reset the
number of bits received
                    read <= #1 0;                       //and wait for the
next packet
                end
            end
            PREVIOUS_STATE <= #1 PS2_CLK;                           //mark
down the previous state of the keyboard clock
        end
      end


      always @(posedge CLK or posedge rst) begin
        if(rst) begin
            CODEWORD <= #1 8'd0;
        end
        else begin
            if (TRIG_ARR) begin                         //and if a full packet
of 11 bits was received
```

```
                if (scan_err) begin             //BUT if the packet was NOT
OK
                    CODEWORD <= #1 8'd0;        //then reset the codeword
register
                end
                else begin
                    CODEWORD <= scan_code[8:1];//else drop down the
unnecessary  bits and transport the 7 DATA bits to CODEWORD reg
                end                         //notice, that the codeword is
also reversed! This is because the first bit to received
            end                             //is supposed to be the last bit
in the codeword…
            else CODEWORD <= #1 8'd0;                       //not a full
packet received, thus reset codeword
        end
      end

      always @(posedge CLK or posedge rst) begin
        if(rst) begin
            EXTENDED<= #1 0;
            BREAK   <= #1 0;
            L_ALT   <= #1 0;
            R_ALT   <= #1 0;
            L_CTRL  <= #1 0;
            R_CTRL  <= #1 0;
            L_SHIFT <= #1 0;
            R_SHIFT <= #1 0;
            SPACE   <= #1 0;
            ESC     <= #1 0;
            D_ARROW <= #1 0;
            L_ARROW <= #1 0;
            R_ARROW <= #1 0;
        end
        else begin
            EXTENDED<= #1 EXTENDED_nxt;
            BREAK   <= #1 BREAK_nxt;
            L_ALT   <= #1 L_ALT_nxt;
            R_ALT   <= #1 R_ALT_nxt;
            L_CTRL  <= #1 L_CTRL_nxt;
            R_CTRL  <= #1 R_CTRL_nxt;
            L_SHIFT <= #1 L_SHIFT_nxt;
            R_SHIFT <= #1 R_SHIFT_nxt;
            SPACE   <= #1 SPACE_nxt;
            ESC     <= #1 ESC_nxt;
            D_ARROW <= #1 D_ARROW_nxt;
            L_ARROW <= #1 L_ARROW_nxt;
            R_ARROW <= #1 R_ARROW_nxt;
        end
      end

    always @* begin
        if(scan_err == 0) begin
            if(CODEWORD == EXTENDED_CODE) begin
                EXTENDED_nxt= 1;
                BREAK_nxt   = 0;
                L_ALT_nxt   = L_ALT;
                R_ALT_nxt   = R_ALT;
                L_CTRL_nxt  = L_CTRL;
                R_CTRL_nxt  = R_CTRL;
                L_SHIFT_nxt = L_SHIFT;
                R_SHIFT_nxt = R_SHIFT;
```

```
        SPACE_nxt    = SPACE;
        ESC_nxt      = ESC;
        D_ARROW_nxt = D_ARROW;
        L_ARROW_nxt = L_ARROW;
        R_ARROW_nxt = R_ARROW;
    end
    else if(CODEWORD == BREAK_CODE) begin
        EXTENDED_nxt= EXTENDED;
        BREAK_nxt     = 1;
        L_ALT_nxt     = L_ALT;
        R_ALT_nxt     = R_ALT;
        L_CTRL_nxt    = L_CTRL;
        R_CTRL_nxt    = R_CTRL;
        L_SHIFT_nxt = L_SHIFT;
        R_SHIFT_nxt = R_SHIFT;
        SPACE_nxt     = SPACE;
        ESC_nxt       = ESC;
        D_ARROW_nxt = D_ARROW;
        L_ARROW_nxt = L_ARROW;
        R_ARROW_nxt = R_ARROW;
    end
    else if(EXTENDED) begin
        if(CODEWORD == R_ARROW_CODE) begin
            if(BREAK) begin
                R_ARROW_nxt = 0;
            end
            else begin
                R_ARROW_nxt = 1;
            end
            EXTENDED_nxt= 0;
            BREAK_nxt       = 0;
            R_ALT_nxt       = R_ALT;
            R_CTRL_nxt      = R_CTRL;
        end
        else if(CODEWORD == ALT_CODE) begin
            if(BREAK) begin
               R_ALT_nxt = 0;
            end
            else begin
               R_ALT_nxt = 1;
            end
            BREAK_nxt       = 0;
            EXTENDED_nxt= 0;
            R_CTRL_nxt      = R_CTRL;
            R_ARROW_nxt = R_ARROW;
        end
        else if(CODEWORD == CTRL_CODE) begin
            if(BREAK) begin
                R_CTRL_nxt = 0;
            end
            else begin
                R_CTRL_nxt = 1;
            end
            BREAK_nxt       = 0;
            EXTENDED_nxt= 0;
            R_ALT_nxt       = R_ALT;
            R_ARROW_nxt = R_ARROW;
        end
        else begin
            EXTENDED_nxt= EXTENDED;
            BREAK_nxt       = BREAK;
```

```
            R_ALT_nxt       = R_ALT;
            R_CTRL_nxt      = R_CTRL;
            R_ARROW_nxt = R_ARROW;
        end
        L_ALT_nxt     = L_ALT;
        L_CTRL_nxt    = L_CTRL;
        L_SHIFT_nxt = L_SHIFT;
        R_SHIFT_nxt = R_SHIFT;
        SPACE_nxt     = SPACE;
        ESC_nxt       = ESC;
        D_ARROW_nxt = D_ARROW;
        L_ARROW_nxt = L_ARROW;
    end
    else if(CODEWORD == D_ARROW_CODE) begin
        if(BREAK) begin
            D_ARROW_nxt = 0;
        end
        else begin
            D_ARROW_nxt = 1;
        end
        EXTENDED_nxt= 0;
        BREAK_nxt     = 0;
        L_ALT_nxt     = L_ALT;
        R_ALT_nxt     = R_ALT;
        L_CTRL_nxt    = L_CTRL;
        R_CTRL_nxt    = R_CTRL;
        L_SHIFT_nxt = L_SHIFT;
        R_SHIFT_nxt = R_SHIFT;
        SPACE_nxt     = SPACE;
        ESC_nxt       = ESC;
        L_ARROW_nxt = L_ARROW;
        R_ARROW_nxt = R_ARROW;
    end
    else if(CODEWORD == L_ARROW_CODE) begin
        if(BREAK) begin
            L_ARROW_nxt = 0;
        end
        else begin
            L_ARROW_nxt = 1;
        end
        EXTENDED_nxt= 0;
        BREAK_nxt     = 0;
        L_ALT_nxt     = L_ALT;
        R_ALT_nxt     = R_ALT;
        L_CTRL_nxt    = L_CTRL;
        R_CTRL_nxt    = R_CTRL;
        L_SHIFT_nxt = L_SHIFT;
        R_SHIFT_nxt = R_SHIFT;
        SPACE_nxt     = SPACE;
        ESC_nxt       = ESC;
        D_ARROW_nxt = D_ARROW;
        R_ARROW_nxt = R_ARROW;
    end
    else if(CODEWORD == ALT_CODE) begin
        if(BREAK) begin
            L_ALT_nxt = 0;
        end
        else begin
            L_ALT_nxt = 1;
        end
        EXTENDED_nxt= 0;
```

```
        BREAK_nxt     = 0;
        R_ALT_nxt     = R_ALT;
        L_CTRL_nxt    = L_CTRL;
        R_CTRL_nxt    = R_CTRL;
        L_SHIFT_nxt = L_SHIFT;
        R_SHIFT_nxt = R_SHIFT;
        SPACE_nxt     = SPACE;
        ESC_nxt       = ESC;
        D_ARROW_nxt = D_ARROW;
        L_ARROW_nxt = L_ARROW;
        R_ARROW_nxt = R_ARROW;
    end
    else if(CODEWORD == CTRL_CODE) begin
        if(BREAK) begin
            L_CTRL_nxt = 0;
        end
        else begin
            L_CTRL_nxt = 1;
        end
        EXTENDED_nxt= 0;
        BREAK_nxt     = 0;
        L_ALT_nxt     = L_ALT;
        R_ALT_nxt     = R_ALT;
        R_CTRL_nxt    = R_CTRL;
        L_SHIFT_nxt = L_SHIFT;
        R_SHIFT_nxt = R_SHIFT;
        SPACE_nxt     = SPACE;
        ESC_nxt       = ESC;
        D_ARROW_nxt = D_ARROW;
        L_ARROW_nxt = L_ARROW;
        R_ARROW_nxt = R_ARROW;
    end
    else if(CODEWORD == L_SHIFT_CODE) begin
        if(BREAK) begin
            L_SHIFT_nxt = 0;
        end
        else begin
            L_SHIFT_nxt = 1;
        end
        EXTENDED_nxt= 0;
        BREAK_nxt     = 0;
        L_ALT_nxt     = L_ALT;
        R_ALT_nxt     = R_ALT;
        L_CTRL_nxt    = L_CTRL;
        R_CTRL_nxt    = R_CTRL;
        R_SHIFT_nxt = R_SHIFT;
        SPACE_nxt     = SPACE;
        ESC_nxt       = ESC;
        D_ARROW_nxt = D_ARROW;
        L_ARROW_nxt = L_ARROW;
        R_ARROW_nxt = R_ARROW;
    end
    else if(CODEWORD == R_SHIFT_CODE) begin
        if(BREAK) begin
            R_SHIFT_nxt = 0;
        end
        else begin
            R_SHIFT_nxt = 1;
        end
        EXTENDED_nxt= 0;
        BREAK_nxt     = 0;
```

```
        L_ALT_nxt     = L_ALT;
        R_ALT_nxt     = R_ALT;
        L_CTRL_nxt    = L_CTRL;
        R_CTRL_nxt    = R_CTRL;
        L_SHIFT_nxt = L_SHIFT;
        SPACE_nxt     = SPACE;
        ESC_nxt       = ESC;
        D_ARROW_nxt = D_ARROW;
        L_ARROW_nxt = L_ARROW;
        R_ARROW_nxt = R_ARROW;
    end
    else if(CODEWORD == SPACE_CODE) begin
        if(BREAK) begin
            SPACE_nxt = 0;
        end
        else begin
            SPACE_nxt = 1;
        end
        EXTENDED_nxt= 0;
        BREAK_nxt     = 0;
        L_ALT_nxt     = L_ALT;
        R_ALT_nxt     = R_ALT;
        L_CTRL_nxt    = L_CTRL;
        R_CTRL_nxt    = R_CTRL;
        L_SHIFT_nxt = L_SHIFT;
        R_SHIFT_nxt = R_SHIFT;
        ESC_nxt       = ESC;
        D_ARROW_nxt = D_ARROW;
        L_ARROW_nxt = L_ARROW;
        R_ARROW_nxt = R_ARROW;
    end
    else if(CODEWORD == ESC_CODE) begin
        if(BREAK) begin
            ESC_nxt = 0;
        end
        else begin
            ESC_nxt = 1;
        end
        EXTENDED_nxt= 0;
        BREAK_nxt     = 0;
        L_ALT_nxt     = L_ALT;
        R_ALT_nxt     = R_ALT;
        L_CTRL_nxt    = L_CTRL;
        R_CTRL_nxt    = R_CTRL;
        L_SHIFT_nxt = L_SHIFT;
        R_SHIFT_nxt = R_SHIFT;
        SPACE_nxt     = SPACE;
        D_ARROW_nxt = D_ARROW;
        L_ARROW_nxt = L_ARROW;
        R_ARROW_nxt = R_ARROW;
    end
    else begin
        EXTENDED_nxt= EXTENDED;
        BREAK_nxt     = BREAK;
        L_ALT_nxt     = L_ALT;
        R_ALT_nxt     = R_ALT;
        L_CTRL_nxt    = L_CTRL;
        R_CTRL_nxt    = R_CTRL;
        L_SHIFT_nxt = L_SHIFT;
        R_SHIFT_nxt = R_SHIFT;
        SPACE_nxt     = SPACE;
```

```
                ESC_nxt       = ESC;
                D_ARROW_nxt = D_ARROW;
                L_ARROW_nxt = L_ARROW;
                R_ARROW_nxt = R_ARROW;
            end
        end
        else begin
            EXTENDED_nxt= 0;
            BREAK_nxt   = 0;
            L_ALT_nxt   = L_ALT;
            R_ALT_nxt   = R_ALT;
            L_CTRL_nxt  = L_CTRL;
            R_CTRL_nxt  = R_CTRL;
            L_SHIFT_nxt = L_SHIFT;
            R_SHIFT_nxt = R_SHIFT;
            SPACE_nxt   = SPACE;
            ESC_nxt     = ESC;
            D_ARROW_nxt = D_ARROW;
            L_ARROW_nxt = L_ARROW;
            R_ARROW_nxt = R_ARROW;
        end
    end

endmodule
// File: main.v
// This is the top level design of DOS mario for fpga

// The `timescale directive specifies what the
// simulation time units are (1 ns here) and what
// the simulator time step should be (1 ps here).

`timescale 1 ns / 1 ps

// Declare the module and its ports. This is
// using Verilog-2001 syntax.

module main (
      input wire clk,
      input wire btnC,
      input wire PS2Clk,
      input wire PS2Data,

    output wire speaker,
      output wire [7:0] vga2Red,
      output wire [7:0] vga2Green,
      output wire [7:0] vga2Blue,
      output wire vga2Hsync,
      output wire vga2Vsync,
      output wire vga2Blank,
      output wire vga2Clk
      );

      assign vga2Clk = clk_25M;

      wire rst;
      assign rst = btnC;

    //Clocks generation
    wire clk_25M, clk_100M, clk_600, clk_100k, locked;

    clk_wiz_0 my_clk_wiz_0(
```

```
        .clk_in(clk),
        .reset(rst),

        .clk_25M(clk_25M),
        .clk_100M(clk_100M),
        .locked(locked)
    );

    clk_divider #(.FREQ(200000)) keyboard_clk_divider (
            .clk100MHz(clk_100M),
            .rst(rst),

            .clk_div(clk_100k)
        );

      clk_divider #(.FREQ(600)) game_clk_divider (
        .clk100MHz(clk_100M),
        .rst(rst),

        .clk_div(clk_600)
    );

    clk_divider #(.FREQ(50000000)) board_divider (
        .clk100MHz(clk_100M),
        .rst(rst),

        .clk_div(clk_50M)
    );

    wire [10:0] rom_addr;
    wire [23:0] rom_data;

     //End of clock generation

    wire [9:0] bcgr_xpos;

    wire [9:0] hcount, hcount_out_bg, hcount_out_brd, hcount_out_player,
hcount_out_monster_1, hcount_out_score, hcount_out;
    wire [9:0] vcount, vcount_out_bg, vcount_out_brd, vcount_out_player,
vcount_out_monster_1, vcount_out_score, vcount_out;
    wire hsync, hsync_out_bg, hsync_out_brd, hsync_out_player,
hsync_out_score, hsync_out_gameover;
    wire vsync, vsync_out_bg, vsync_out_brd, vsync_out_player,
vsync_out_score, vsync_out_gameover;
    wire blnk, blnk_out_bg, blnk_out_brd, blnk_out_player, blnk_out_score,
blnk_out_gameover, blnk_out;
    wire [23:0] rgb_out_bg, rgb_out_brd, rgb_out_player, rgb_out_monster_1,
rgb_out_score, rgb_out_gameover;

    wire [9:0] mario_x, monster_1_x;
    wire [8:0] mario_y, monster_1_y;

    wire [5:0] plane_xpos_ofs;
    wire [7:0]  plane_xpos;
    wire [3:0]  plane_ypos;
    wire [15:0] blocking_player;
    wire mario_dir, blocking, monster_1_dir;
    wire [5:0] block_display;

    wire [7:0] engine_xpos, block_display_xpos;
    wire [3:0] engine_ypos, block_display_ypos;
```

```
    wire [5:0] engine_write_block, engine_block;

      VgaTiming My_VgaTiming (
          .pclk(clk_25M),
          .rst(rst),

          .hcount(hcount),
          .hsync(hsync),
          .vcount(vcount),
          .vsync(vsync),
          .blnk(blnk)
      );

//*********BEGINING OF GRAPHICS********

      DrawBackground My_DrawBackground (
          .hcount_in(hcount),
          .hsync_in(hsync),
          .vcount_in(vcount),
          .vsync_in(vsync),
          .blnk_in(blnk),
          .xoffset(bcgr_xpos),
          .clk(clk_25M),
          .rst(rst),

          .hcount_out(hcount_out_bg),
          .hsync_out(hsync_out_bg),
          .vcount_out(vcount_out_bg),
          .vsync_out(vsync_out_bg),
          .rgb_out(rgb_out_bg),
          .blnk_out(blnk_out_bg)
      );

    Board My_Board (
        .hcount_in(hcount_out_bg),
        .hsync_in(hsync_out_bg),
        .vcount_in(vcount_out_bg),
        .vsync_in(vsync_out_bg),
        .rgb_in(rgb_out_bg),
        .blnk_in(blnk_out_bg),
        .plane_xpos(plane_xpos),
        .plane_xpos_ofs(plane_xpos_ofs),
        .block(block_display),
        .pclk(clk_25M),
        .clk(clk_50M),
        .rst(rst),

        .block_xpos(block_display_xpos),
        .block_ypos(block_display_ypos),
          .hcount_out(hcount_out_brd),
        .hsync_out(hsync_out_brd),
        .vcount_out(vcount_out_brd),
        .vsync_out(vsync_out_brd),
        .rgb_out(rgb_out_brd),
        .blnk_out(blnk_out_brd)
    );

    Player My_Player (
        .hcount_in(hcount_out_brd),
        .hsync_in(hsync_out_brd),
        .vcount_in(vcount_out_brd),
```

```
    .vsync_in(vsync_out_brd),
    .rgb_in(rgb_out_brd),
    .blnk_in(blnk_out_brd),
    .xpos(mario_x),
    .ypos(mario_y),
    .direction(mario_dir),
    .size(0),
    .fire(0),
    .clk(clk_25M),
    .rst(rst),
    .rom_data(rom_data),

    .rom_addr(rom_addr),
    .hcount_out(hcount_out_player),
    .hsync_out(hsync_out_player),
    .vcount_out(vcount_out_player),
    .vsync_out(vsync_out_player),
    .rgb_out(rgb_out_player),
    .blnk_out(blnk_out_player)
  );

DrawMarioScore My_DrawMarioScore(
    .clk(clk_25M),
    .rst(rst),
    .hcount_in(hcount_out_player),
    .hsync_in(hsync_out_player),
    .vcount_in(vcount_out_player),
    .vsync_in(vsync_out_player),
    .rgb_in(rgb_out_player),
    .blnk_in(blnk_out_player),
    .char_pixels(char_line_pixels),

    .hcount_out(hcount_out_score),
    .hsync_out(hsync_out_score),
    .vcount_out(vcount_out_score),
    .vsync_out(vsync_out_score),
    .rgb_out(rgb_out_score),
    .blnk_out(blnk_out_score),
    .char_xy(char_xy),
    .char_line(char_line)
);

Gameover My_Gameover (
    .hcount_in(hcount_out_score),
    .hsync_in(hsync_out_score),
    .vcount_in(vcount_out_score),
    .vsync_in(vsync_out_score),
    .rgb_in(rgb_out_score),
    .blnk_in(blnk_out_score),
    .gameover(gameover),
    .rst(rst),
    .clk(clk_25M),

    .hcount_out(hcount_out),
    .hsync_out(hsync_out_gameover),
    .vcount_out(vcount_out),
    .vsync_out(vsync_out_gameover),
    .rgb_out(rgb_out_gameover),
    .blnk_out(blnk_out_gameover)
);
```

```
    Change2Negedge My_Change2Negedge(
        .hsync_in(hsync_out_gameover),
        .vsync_in(vsync_out_gameover),
        .blnk_in(blnk_out_gameover),
        .rgb_in(rgb_out_gameover),
        .clk(clk_25M),
        .rst(rst),

        .hsync_out(vga2Hsync),
        .vsync_out(vga2Vsync),
        .blnk_out(vga2Blank),
        .rgb_out({vga2Red,vga2Green,vga2Blue})
    );


//**********END OF GRAPHICS**********


//**********GAME CONTROLL************

    dist_mem_gen_0 mario_picture(
        .a(rom_addr),
        .spo(rom_data)
    );

    wire [11:0] fs_ram_a, fs_ram_dpra;
    wire [5:0] fs_ram_d, fs_ram_spo, fs_ram_dpo;

    dist_mem_gen_1 first_stage_ram(
        .a(fs_ram_a),
        .d(fs_ram_d),
        .dpra(fs_ram_dpra),
        .clk(clk_100M),
        .we(fs_ram_we),
        .spo(fs_ram_spo),
        .dpo(fs_ram_dpo)
    );

   wire [11:0] fs_addr_c;
   wire [5:0] fs_data_c;

    dist_mem_gen_2 first_stage_rom(
        .a(fs_addr_c),
        .spo(fs_data_c)
    );

    wire [7:0] char_line_pixels;
    wire [7:0] char_code;
    wire [3:0] char_line;
    wire [7:0] char_xy;
    wire [3:0] mario_lifes;
    wire [11:0] player_points;
    wire [3:0] lvl_number;

    MarioFontRom My_MarioFontRom(
        .addr({char_code, char_line}),
        .char_line_pixels(char_line_pixels)
    );

    MarioScore24x1 My_MarioScore24x1(
        .char_xy(char_xy),
```

```
    .mario_lives(mario_lifes),
    .level(lvl_number),
    .coins(player_points),

    .char_code(char_code)
);

//Music
Music My_Music(
    .clk(clk_25M),
    .speaker(speaker)
);

FirstStage Engine_FirstStage(
    .plane_xpos(engine_xpos),
    .plane_ypos(engine_ypos),
    .ram_read_data(fs_ram_spo),
    .write_block(engine_write_block),
    .save_block(engine_we),
    .copy_backup(stage_restart),
    .copy_read_data(fs_data_c),
    .clk(clk_100M),
    .rst(rst),

    .blocking(engine_blocking),
    .block(engine_block),
    .ram_addr(fs_ram_a),
    .ram_write_data(fs_ram_d),
    .ram_we(fs_ram_we),
    .copy_addr(fs_addr_c),
    .backuped(stage_restarted)
);

FirstStage Display_FirstStage(
    .plane_xpos(block_display_xpos),
    .plane_ypos(block_display_ypos),
    .ram_read_data(fs_ram_dpo),
    .save_block(0),
    .copy_backup(0),

    .block(block_display),
    .ram_addr(fs_ram_dpra)
);

wire [1:0] player_speed;
GameEngine My_GameEngine(
    .clk(clk_25M),
    .game_clk(clk_600),
    .rst(rst),
    .L_ALT(l_alt),
    .R_ALT(r_alt),
    .L_CTRL(l_ctrl),
    .R_CTRL(r_ctrl),
    .SPACE(space),
    .L_SHIFT(l_shift),
    .R_SHIFT(r_shift),
    .ESC(esc),
    .D_ARROW(d_arrow),
    .L_ARROW(l_arrow),
    .R_ARROW(r_arrow),
    .blocking_in(engine_blocking),
```

```
        .block_in(engine_block),
        .gameover(gameover),
        .stage_restarted(stage_restarted),

        .block_xpos(engine_xpos),
        .block_ypos(engine_ypos),
        .player_xpos(mario_x),
        .player_ypos(mario_y),
        .bcgr_xpos(bcgr_xpos),
        .plane_xpos(plane_xpos),
        .plane_xpos_ofs(plane_xpos_ofs),
        .player_dir(mario_dir),
        .player_speed(player_speed),
        .player_life(mario_lifes),
        .player_points(player_points),
        .lvl_number(lvl_number),
        .write_block(engine_write_block),
        .block_we(engine_we),
        .restartgame(stage_restart)
    );
//******END OF GAME CONTROLL******

//*********PHERIPERALS************

    Keyboard my_Keyboard(
        .CLK(clk_100k),
        .rst(rst),
        .PS2_CLK(PS2Clk),
        .PS2_DATA(PS2Data),

        .L_ALT(l_alt),
        .R_ALT(r_alt),
        .L_CTRL(l_ctrl),
        .R_CTRL(r_ctrl),
        .SPACE(space),
        .L_SHIFT(l_shift),
        .R_SHIFT(r_shift),
        .ESC(esc),
        .D_ARROW(d_arrow),
        .L_ARROW(l_arrow),
        .R_ARROW(r_arrow)
    );

endmodule
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
///////
// Company:         AGH UST
// Engineer: Wojciech Gredel, Hubert Górowski
//
// Create Date:
// Design Name:
// Module Name:     MarioScore24x1
// Project Name:    DOS_Mario
// Target Devices:  Basys3
// Tool versions:   Vivado 2016.1
// Description:
//    This module is ROM with some fancy strings
//
// Dependencies:
//
```

```
// Revision:
// Revision 0.01 - Module created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
///////

module MarioScore24x1 (
        input wire  [7:0] char_xy,
            input wire  [3:0] mario_lives,
            input wire  [3:0] level,
            input wire  [11:0] coins,

        output wire [7:0] char_code
    );

    reg [7:0] char_code_nxt;
      reg [3:0] bcd0, bcd1, bcd2;
      reg [3:0] hex1, hex2, hex3, hex4, hex5;

      integer i;

    always @(coins)
    begin
        bcd0 = 0;
        bcd1 = 0;
        bcd2 = 0;

        for ( i = 11; i >= 0; i = i - 1 )
        begin
            if( bcd0 > 4 ) bcd0 = bcd0 + 3;
            if( bcd1 > 4 ) bcd1 = bcd1 + 3;
            if( bcd2 > 4 ) bcd2 = bcd2 + 3;
            { bcd2[3:0], bcd1[3:0], bcd0[3:0] } =
            { bcd2[2:0], bcd1[3:0], bcd0[3:0],coins[i] };
        end
    end

      always @(*)
      begin
            case(mario_lives)
                  0: hex1 = 8'h00;
                  1: hex1 = 8'h01;
                  2: hex1 = 8'h02;
                  3: hex1 = 8'h03;
                  4: hex1 = 8'h04;
                  5: hex1 = 8'h05;
                  6: hex1 = 8'h06;
                  7: hex1 = 8'h07;
                  8: hex1 = 8'h08;
                  9: hex1 = 8'h09;
                  default: hex1 = 8'h00;
            endcase

            case(level)
                  0: hex2 = 8'h00;
                  1: hex2 = 8'h01;
                  2: hex2 = 8'h02;
                  3: hex2 = 8'h03;
                  4: hex2 = 8'h04;
                  5: hex2 = 8'h05;
```

```
            6: hex2 = 8'h06;
            7: hex2 = 8'h07;
            8: hex2 = 8'h08;
            9: hex2 = 8'h09;
            default: hex2 = 8'h00;
        endcase

        case(bcd0)
            0: hex3 = 8'h00;
            1: hex3 = 8'h01;
            2: hex3 = 8'h02;
            3: hex3 = 8'h03;
            4: hex3 = 8'h04;
            5: hex3 = 8'h05;
            6: hex3 = 8'h06;
            7: hex3 = 8'h07;
            8: hex3 = 8'h08;
            9: hex3 = 8'h09;
            default: hex3 = 8'h00;
        endcase

        case(bcd1)
            0: hex4 = 8'h00;
            1: hex4 = 8'h01;
            2: hex4 = 8'h02;
            3: hex4 = 8'h03;
            4: hex4 = 8'h04;
            5: hex4 = 8'h05;
            6: hex4 = 8'h06;
            7: hex4 = 8'h07;
            8: hex4 = 8'h08;
            9: hex4 = 8'h09;
            default: hex4 = 8'h00;
        endcase

        case(bcd2)
            0: hex5 = 8'h00;
            1: hex5 = 8'h01;
            2: hex5 = 8'h02;
            3: hex5 = 8'h03;
            4: hex5 = 8'h04;
            5: hex5 = 8'h05;
            6: hex5 = 8'h06;
            7: hex5 = 8'h07;
            8: hex5 = 8'h08;
            9: hex5 = 8'h09;
            default: hex5 = 8'h00;
        endcase
    end


always @* begin
    case(char_xy)
        8'h00:      char_code_nxt = 8'h0A; // M
        8'h01:      char_code_nxt = 8'h0B; // A
        8'h02:  char_code_nxt = 8'h0C; // R
        8'h03:  char_code_nxt = 8'h0D; // I
        8'h04:  char_code_nxt = 8'h0E; // O
        8'h05:  char_code_nxt = 8'h0F; //
        8'h06:  char_code_nxt = 8'h10; // x
        8'h07:  char_code_nxt = hex1;  // liczba zyc
```

```
    8'h08:  char_code_nxt = 8'h0F; //
    8'h09:  char_code_nxt = 8'h0F; //
    8'h0a:  char_code_nxt = 8'h0F; //
    8'h0b:  char_code_nxt = 8'h0F; //
    8'h0c:  char_code_nxt = 8'h0F; //
    8'h0d:  char_code_nxt = 8'h0F; //
8'h0e:  char_code_nxt = 8'h0F; //
8'h0f:  char_code_nxt = 8'h0F; //
8'h10:  char_code_nxt = 8'h0F; //
8'h11:  char_code_nxt = 8'h0F; //
8'h12:  char_code_nxt = 8'h0F; //
8'h13:  char_code_nxt = 8'h0F; //
8'h14:  char_code_nxt = 8'h0F; //
8'h15:  char_code_nxt = 8'h0F; //
8'h16:  char_code_nxt = 8'h0F; //
8'h17:  char_code_nxt = 8'h0F; //
8'h18:  char_code_nxt = 8'h0F; //
8'h19:  char_code_nxt = 8'h0F; //
8'h1a:  char_code_nxt = 8'h0F; //
8'h1b:  char_code_nxt = 8'h0F; //
8'h1c:  char_code_nxt = 8'h0F; //
8'h1d:  char_code_nxt = 8'h0F; //

    8'h1e:  char_code_nxt = 8'h0F; //
    8'h1f:  char_code_nxt = 8'h0F; //
    8'h20:  char_code_nxt = 8'h11; // moneta
    8'h21:  char_code_nxt = 8'h0F; //
    8'h22:  char_code_nxt = 8'h10; // x
    8'h23:  char_code_nxt = hex5;  // liczba monet
    8'h24:  char_code_nxt = hex4;  // liczba monet
    8'h25:  char_code_nxt = hex3;  // liczba monet
    8'h26:  char_code_nxt = 8'h0F; //
8'h27:  char_code_nxt = 8'h0F; //
8'h28:  char_code_nxt = 8'h0F; //
8'h29:  char_code_nxt = 8'h0F; //
8'h2a:  char_code_nxt = 8'h0F; //
8'h2b:  char_code_nxt = 8'h0F; //
8'h2c:  char_code_nxt = 8'h0F; //
8'h2d:  char_code_nxt = 8'h0F; //
8'h2e:  char_code_nxt = 8'h0F; //
8'h2f:  char_code_nxt = 8'h0F; //
8'h30:  char_code_nxt = 8'h0F; //
8'h31:  char_code_nxt = 8'h0F; //
8'h32:  char_code_nxt = 8'h0F; //
8'h33:  char_code_nxt = 8'h0F; //
8'h34:  char_code_nxt = 8'h0F; //
8'h35:  char_code_nxt = 8'h0F; //
8'h36:  char_code_nxt = 8'h0F; //
8'h37:  char_code_nxt = 8'h0F; //
8'h38:  char_code_nxt = 8'h0F; //
8'h39:  char_code_nxt = 8'h0F; //
8'h3a:  char_code_nxt = 8'h0F; //
8'h3b:  char_code_nxt = 8'h0F; //

    8'h3c:  char_code_nxt = 8'h0F; //
    8'h3d:  char_code_nxt = 8'h0F; //
    8'h3e:  char_code_nxt = 8'h12; // L
    8'h3f:  char_code_nxt = 8'h13; // E
    8'h40:  char_code_nxt = 8'h14; // V
    8'h41:  char_code_nxt = 8'h13; // E
```

```
            8'h42:  char_code_nxt = 8'h12; // L
            8'h43:  char_code_nxt = 8'h0F; //
            8'h44:  char_code_nxt = hex2;  // numer levelu

            default: char_code_nxt = 8'hff;
            endcase
      end

    assign char_code = char_code_nxt;

endmodule
`timescale 1 ns / 1 ps
//////////////////////////////////////////////////////////////////////////////////
///////
// Company:          AGH UST
// Engineer: Wojciech Gredel, Hubert Górowski
//
// Create Date:
// Design Name:
// Module Name:      Music
// Project Name:     DOS_Mario
// Target Devices:   Basys3
// Tool versions:    Vivado 2016.1
// Description:
//    This module contains and generates music
//
// Dependencies:
//
// Revision:
// Revision 0.01 - Module created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
///////

module Music (
      input wire      clk,
      output reg      speaker
);

    wire [7:0]  fullnote;
    wire [2:0]  octave;
    wire [3:0]  note;
    reg  [30:0] tone;
    reg  [8:0]  clkdivider;
    reg  [8:0]  counter_note;
    reg  [7:0]  counter_octave;

    music_rom my_music_rom (
        .clk(clk),
        .address(tone[30:22]),
        .note(fullnote)
    );

    divide_by12 my_divide_by12 (
        .numerator(fullnote[5:0]),
        .quotient(octave),
        .remainder(note)
    );

    always @(posedge clk)
```

```
        tone    <= tone+31'd1;

    always @*
    case(note)
         0: clkdivider = 9'd511; //A
         1: clkdivider = 9'd482; // A#/Bb
         2: clkdivider = 9'd455; //B
         3: clkdivider = 9'd430; //C
         4: clkdivider = 9'd405; // C#/Db
         5: clkdivider = 9'd383; //D
         6: clkdivider = 9'd361; // D#/Eb
         7: clkdivider = 9'd341; //E
         8: clkdivider = 9'd322; //F
         9: clkdivider = 9'd303; // F#/Gb
        10: clkdivider = 9'd286; //G
        11: clkdivider = 9'd270; // G#/Ab
        default: clkdivider = 9'd0;
    endcase


    always @(posedge clk) counter_note <= (counter_note==0 ? clkdivider :
counter_note-9'd1);
    always @(posedge clk) if(counter_note==0) counter_octave <=
(counter_octave==0 ? 8'd255 >> octave : counter_octave-8'd1);
    always @(posedge clk) if(counter_note==0 && counter_octave==0 &&
fullnote!=0 && tone[21:18]!=0) speaker <= ~speaker;

    endmodule



    // This module is resposible for prividing divide by 12 operation.
    module divide_by12(
        input [5:0] numerator,  // value to be divided by 12
        output reg [2:0] quotient,
        output [3:0] remainder
    );

    reg [1:0] remainder3to2;
    always @(numerator[5:2])
    case(numerator[5:2])        // look-up table
         0: begin quotient=0; remainder3to2=0; end
         1: begin quotient=0; remainder3to2=1; end
         2: begin quotient=0; remainder3to2=2; end
         3: begin quotient=1; remainder3to2=0; end
         4: begin quotient=1; remainder3to2=1; end
         5: begin quotient=1; remainder3to2=2; end
         6: begin quotient=2; remainder3to2=0; end
         7: begin quotient=2; remainder3to2=1; end
         8: begin quotient=2; remainder3to2=2; end
         9: begin quotient=3; remainder3to2=0; end
        10: begin quotient=3; remainder3to2=1; end
        11: begin quotient=3; remainder3to2=2; end
        12: begin quotient=4; remainder3to2=0; end
        13: begin quotient=4; remainder3to2=1; end
        14: begin quotient=4; remainder3to2=2; end
        15: begin quotient=5; remainder3to2=0; end
    endcase

    assign remainder[1:0] = numerator[1:0];  // the first 2 bits are copied
```

```
through
    assign remainder[3:2] = remainder3to2;  // and the last 2 bits come
from the case statement

    endmodule
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
///////
// Company:
// Engineer:
//
// Create Date: 05/29/2016 09:39:31 PM
// Design Name:
// Module Name: new_block
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
///////


module new_block(
    input wire [5:0] block_in,
    input wire up_direction,
    input wire direction,
    input wire [9:0] relative_xpos,
    input wire [8:0] relative_ypos,
    output reg [5:0] block_out,
    output reg write_enable,
    output reg new_point
    );


    always @* begin
        if(block_in == GY) begin
            new_point = 1;
            block_out = B;
            write_enable = 1;
        end
        else if(((relative_xpos % 40) < 20) && (direction == 0)) begin
            if(block_in == D) begin
                if(up_direction) begin
                    new_point = 1;
                    block_out = DY;
                    write_enable = 1;
                end
                else begin
                    new_point = 0;
                    block_out = block_in;
                    write_enable = 0;
                end
            end
            else if(block_in == J) begin
```

```
                new_point = 0;
                if(up_direction) begin
                    block_out = B;
                    write_enable = 1;
                end
                else begin
                    block_out = block_in;
                    write_enable = 0;
                end
            end
            else begin
                new_point = 0;
                block_out = block_in;
                write_enable = 0;
            end
        end
        else if(((relative_xpos % 40) >= 20) && (direction == 1)) begin
            if(block_in == D) begin
                if(up_direction) begin
                    new_point = 1;
                    block_out = DY;
                    write_enable = 1;
                end
                else begin
                    new_point = 0;
                    block_out = block_in;
                    write_enable = 0;
                end
            end
            else if(block_in == J) begin
                new_point = 0;
                if(up_direction) begin
                    block_out = B;
                    write_enable = 1;
                end
                else begin
                    block_out = block_in;
                    write_enable = 0;
                end
            end
            else begin
                new_point = 0;
                block_out = block_in;
                write_enable = 0;
            end
        end
        else begin
            new_point = 0;
            block_out = block_in;
            write_enable = 0;
        end
    end


        localparam    A      =  1      ;
        localparam    B    =    0    ;
        localparam    C    =    2    ;
        localparam    D    =    3    ;
        localparam    E    =    4    ;
        localparam    F    =    5    ;
        localparam    G    =    6    ;
```

```
            localparam    H    =    7    ;
            localparam    I    =    8    ;
            localparam    J    =    9    ;
            localparam    K    =    10    ;
            localparam    L    =    11    ;
            localparam    M    =    12    ;
            localparam    N    =    13    ;
            localparam    O    =    14    ;
            localparam    P    =    15    ;
            localparam    Q    =    16    ;
            localparam    R    =    17    ;
            localparam    S    =    18    ;
            localparam    T    =    19    ;
            localparam    U    =    20    ;
            localparam    V    =    21    ;
            localparam    W    =    22    ;
            localparam    X    =    23    ;
            localparam    Y    =    24    ;
            localparam    Z    =    25    ;
            localparam    AY    =    26    ;
            localparam    IY    =    27    ;
            localparam    GY    =    28    ;
            localparam    KY    =    29    ;
            localparam    PY    =    30    ;
            localparam    TY    =    31    ;
            localparam    UY    =    32    ;
            localparam    WY    =    33    ;
            localparam    DY    =    34    ;
            localparam    BY    =    35    ;



    endmodule
`timescale 1 ns / 1 ps
//////////////////////////////////////////////////////////////////////////////////
///////
// Company:         AGH UST
// Engineer: Wojciech Gredel, Hubert Górowski
//
// Create Date:
// Design Name:
// Module Name:     Player
// Project Name:    DOS_Mario
// Target Devices:  Basys3
// Tool versions:   Vivado 2016.1
// Description:
//    This module displays player
//
// Dependencies:
//
// Revision:
// Revision 0.01 - Module created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
///////

module Player
#( parameter
    SMALL   = 40,
```

```
    BIG     = 80
)
(
    input wire [9:0] xpos,
    input wire [8:0] ypos,
    input wire direction,
    input wire size,
    input wire fire,
    input wire [9:0] hcount_in,
    input wire hsync_in,
    input wire [9:0] vcount_in,
    input wire vsync_in,
    input wire blnk_in,
    input wire rst,
    input wire clk,
    input wire [23:0] rgb_in,
    input wire [23:0] rom_data,

    output reg [10:0] rom_addr,
    output reg [9:0] hcount_out,
    output reg hsync_out,
    output reg [9:0] vcount_out,
    output reg vsync_out,
    output reg [23:0] rgb_out,
    output reg blnk_out
    );

    localparam ALFA_COLOR = 24'hA3_49_A4;
    localparam YRES = 480;
    localparam PLAYER_WIDTH = 40;
    reg [5:0] player_height;
    reg [5:0] player_height_nxt;
    reg [23:0] rgb_nxt;
    reg [10:0] rom_addr_nxt;

    always @(posedge clk or posedge rst) begin
        if(rst)
            rom_addr    <= #1 0;
        else
            rom_addr    <= #1 rom_addr_nxt;
    end

    always @(posedge clk or posedge rst) begin
        if(rst) begin
            rgb_out     <= #1 0;
            hcount_out  <= #1 0;
            hsync_out   <= #1 0;
            vcount_out  <= #1 0;
            vsync_out   <= #1 0;
            blnk_out    <= #1 0;
        end
        else begin
            rgb_out     <= #1 rgb_nxt;
            hcount_out  <= #1 hcount_in;
            hsync_out   <= #1 hsync_in;
            vcount_out  <= #1 vcount_in;
            vsync_out   <= #1 vsync_in;
            blnk_out    <= #1 blnk_in;
        end
    end
```

```
    always @* begin
        if(size)
            player_height_nxt = BIG;
        else
            player_height_nxt = SMALL;
    end

    always@(posedge clk or posedge rst) begin
        if(rst) begin
            player_height = SMALL;
        end
        else begin
            player_height = player_height_nxt;
        end
    end

    always @* begin
        if(direction) begin
            rom_addr_nxt = PLAYER_WIDTH*(vcount_in  -(YRES - ypos -
player_height)) + (PLAYER_WIDTH - 1 - (hcount_in - xpos + 1));
        end
        else begin
            rom_addr_nxt = PLAYER_WIDTH*(vcount_in  -(YRES - ypos -
player_height)) + ((hcount_in - xpos + 1));
        end
    end

    always @*
      begin
        if(((YRES - 1 - vcount_in) < (ypos + player_height)) && ((YRES - 1
- vcount_in) >= ypos) && (hcount_in < (xpos+PLAYER_WIDTH) ) && ((xpos) <=
hcount_in)) begin
            if(rom_data == ALFA_COLOR)
                rgb_nxt = rgb_in;
            else
                rgb_nxt = rom_data;
        end
        else begin
            rgb_nxt = rgb_in;
        end
    end

endmodule
`timescale 1 ns / 1 ps
//////////////////////////////////////////////////////////////////////////////////
///////
// Company:         AGH UST
// Engineer: Wojciech Gredel, Hubert Górowski
//
// Create Date:
// Design Name:
// Module Name:     VgaTiming
// Project Name:    DOS_Mario
// Target Devices:  Basys3
// Tool versions:   Vivado 2016.1
// Description:
//    This module creates timing for vga
//  VGA 640x480 60Hz 25.175MHz pixel clock, horizontal pulse - negative,
vertical pulse - negative
//
// Dependencies:
```

```
//
// Revision:
// Revision 0.01 - Module created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
///////

module VgaTiming (
      input  wire pclk,
      input  wire rst,

      output reg [9:0] hcount,
      output reg hsync,
      output reg [9:0] vcount,
      output reg vsync,
      output reg blnk
      );

//    localparam H_TOTAL_TIME = 1056;
//    //localparam H_ADDR_TIME     = 800;
//    localparam H_BLNK_START = 800;
//    //localparam H_BLNK_TIME    = 160;
//    localparam H_SYNC_START    = 840;
//    localparam H_SYNC_TIME     = 128;
//    localparam H_SYNC_POL    = 1;    //0 - negative, 1 - positive

//    localparam V_TOTAL_TIME = 628;
//    //localparam V_ADDR_TIME     = 600;
//    localparam V_BLNK_START = 600;
//    //localparam V_BLNK_TIME    = 28;
//    localparam V_SYNC_START    = 601;
//    localparam V_SYNC_TIME     = 4;
//    localparam V_SYNC_POL    = 1;     //0 - negative, 1 - positive

      localparam H_TOTAL_TIME = 800;
      //localparam H_ADDR_TIME      = 640;
      localparam H_BLNK_START = 640;
      //localparam H_BLNK_TIME      = 160;
      localparam H_SYNC_START = 656;
      localparam H_SYNC_TIME  = 96;
      localparam H_SYNC_POL   = 0;  //0 - negative, 1 - positive

      localparam V_TOTAL_TIME = 525;
      //localparam V_ADDR_TIME      = 480;
      localparam V_BLNK_START = 480;
      //localparam V_BLNK_TIME      = 45;
      localparam V_SYNC_START = 490;
      localparam V_SYNC_TIME  = 2;
      localparam V_SYNC_POL   = 0;  //0 - negative, 1 - positive

      reg [10:0] vcount_nxt;
      reg vsync_nxt;
      reg [10:0] hcount_nxt;
      reg hsync_nxt;
      reg vblnk, hblnk;
      reg vblnk_nxt, hblnk_nxt;

    always @(posedge pclk or posedge rst) begin
           if (rst) begin
                 hcount  <= #1 0;
```

```
        hsync   <= #1 0;
              vcount  <= #1 0;
          vsync   <= #1 0;
          hblnk   <= #1 0;
          vblnk   <= #1 0;
    end
        else begin
              hcount  <= #1 hcount_nxt;
        hsync   <= #1 hsync_nxt;
              vcount  <= #1 vcount_nxt;
        vsync   <= #1 vsync_nxt;
        hblnk   <= #1 hblnk_nxt;
        vblnk   <= #1 vblnk_nxt;
    end
end

always @* begin
    blnk = ~(hblnk | vblnk);
end

  always @* begin
  //horizontal
    if(hcount==H_BLNK_START - 1) begin
          hcount_nxt = hcount+1;
          if(H_SYNC_POL) hsync_nxt=0;
          else hsync_nxt = 1;
          hblnk_nxt=1;
    end
    else if(hcount==H_SYNC_START - 1) begin
          hcount_nxt = hcount+1;
          if(H_SYNC_POL) hsync_nxt=1;
          else hsync_nxt = 0;
        hblnk_nxt=1;
    end
    else if (hcount==H_SYNC_START + H_SYNC_TIME - 1) begin
          hcount_nxt = hcount+1;
          if(H_SYNC_POL) hsync_nxt=0;
          else hsync_nxt = 1;
        hblnk_nxt=1;
    end
    else if (hcount==H_TOTAL_TIME - 1) begin
          hcount_nxt = 0;
        if(H_SYNC_POL) hsync_nxt=0;
        else hsync_nxt = 1;
        hblnk_nxt=0;
    end
    else begin
          hcount_nxt = hcount+1;
        hsync_nxt = hsync;
        hblnk_nxt = hblnk;
        end
        //vertical
    if(hcount==H_TOTAL_TIME - 1) begin
              if(vcount==V_BLNK_START - 1) begin
                    vcount_nxt = vcount+1;
                    if(V_SYNC_POL) vsync_nxt=0;
              else vsync_nxt = 1;
              vblnk_nxt=1;
        end
              else if (vcount==V_SYNC_START - 1) begin
                    vcount_nxt = vcount+1;
```

```
                        if(V_SYNC_POL) vsync_nxt=1;
                  else vsync_nxt = 0;
                  vblnk_nxt = 1;
            end
                  else if (vcount==V_SYNC_START + V_SYNC_TIME - 1) begin
                        vcount_nxt = vcount+1;
                        if(V_SYNC_POL) vsync_nxt=0;
                  else vsync_nxt = 1;
                        vblnk_nxt = 1;
            end
                  else if (vcount==V_TOTAL_TIME - 1) begin
                        vcount_nxt = 0;
                        if(V_SYNC_POL) vsync_nxt=0;
                 else vsync_nxt = 1;
                        vblnk_nxt = 0;
            end
                  else begin
                        vcount_nxt = vcount+1;
                        vsync_nxt = vsync;
                        vblnk_nxt = vblnk;
            end
         end
         else begin
               vcount_nxt = vcount;
               vsync_nxt = vsync;
               vblnk_nxt = vblnk;
         end
      end

endmodule
```