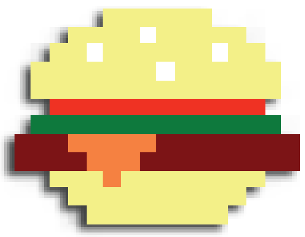
PAC-EDWARDS

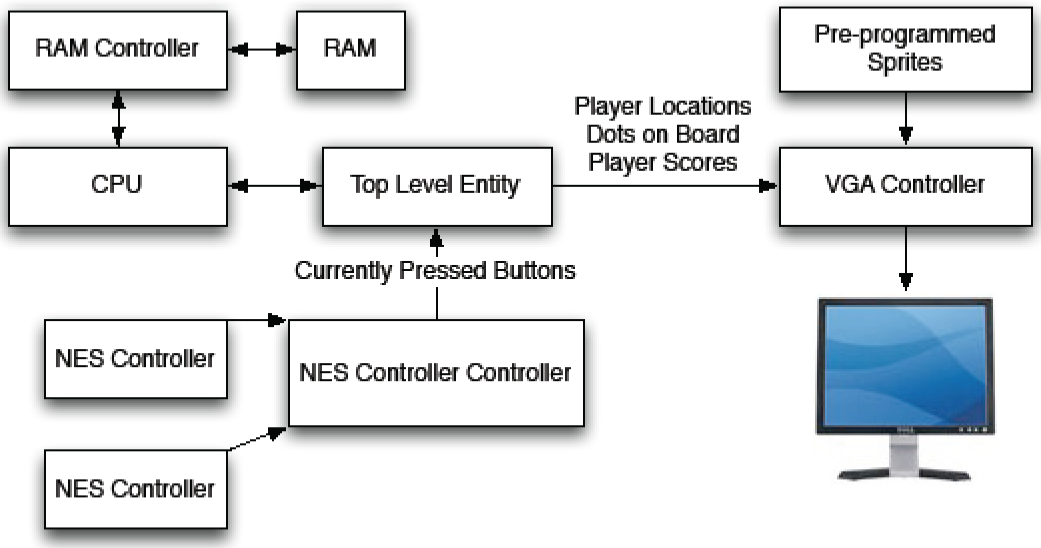
Varun Mehta - Mike Pierorazio - Jeffrey Cropsey



The *Pac-Edwards* Objective:

Originally code named TupacMan to keep our Professor Edwards and MacDonald's theme secret, Pac-Edwards was designed to be a two player Pac-Man style arcade game in which players competed to in a race to "eat" as many pellets as they could in a Pac-Man style maze. We used the VGA output of the Altera board to present the game's graphics, storing all images in hardware. A small RAM was created to store the graphics and act as a communication channel between hardware and software. Software was written for the NIOS II processor to track game state (including sprite movement and score) as well as respond to user input. Finally, to get user input, we used the arbitrary I/O ports on the DE2 board to connect NES controllers.

The *Pac-Edwards* Architecture:



Pac-Edwards Architecture In-Depth:

Game Logic

The game commences after both players press ‘Start’. Each characters sprite will move in the direction of the sprite’s mouth until direction is changed by the player using the directional pad. Only the directional pad is used during game play. Each time a player moves over a pellet it will be ‘eaten’, his score will increase and the pellet sprite will be removed from the game board. To restart after completion of the game both players should press ‘Start’.

NES Controler Input

The NES polls its controller every 60 Hz in order to determine the state of the eight buttons. It accomplishes this by sending a high signal to the ‘Latch’ pin; the controller then latches the state of the buttons. The ‘Latch’ signal is followed by eight high pulses of the ‘Pulse’ pin. By dividing the on-board 25MHz clock and coupling it with some counters and simple logic we were able to mimic the ‘Latch’ signal and send it to the nes-controllers through two of the general I/O pins (one for each controller).

```
                +----> Power
                |
5 +---------+   7
  | x  x  o   \
  | o  o  o  o |
4 +------------+ 1
    |  |  |  |
    |  |  |  +-> Ground
    |  |  +----> Pulse
    |  +-------> Latch
    +----------> Data
```

After the latch signal, ‘Data’ pin goes high for the duration of the eight high pulses of ‘Pulse’ before returning to low (if no buttons have been pressed). Each of the eight pulses relates to each of the eight buttons on the NES in this order: A, B, Select, Start, Up, Down, Left, Right. If one of these buttons was pressed when the ‘Latch’ signal was received then ‘Data’ will be low during that corresponding pulse of ‘Pulse’. Again by dividing the 25MHz clock we were able to mimic the ‘Pulse’ signals and send them through two of the general I/O pins. Another pair of general I/O pins were used to receive the data signal from the nes controller which had the 8-bit serial encoding of the current buttons pushed (as described above).

The 8-bit serial input from the controller is interpreted bit-by-bit by a state machine that progresses on subsequent latch and pulse signals. It records the data into two different vectors (each one corresponding to a different player). Those vectors are available to the CPU as signals and can be interpreted by the C program directly.

RAM

It seems that everything all of the images and game states will be able to be stored in the main chip on the Altera board either in our custom RAM or as signals.

The *Pac-Edwards* Video Controller:

Video (at VGA resolution) for Pac-Edwards posed a number of design challenges because of the number of concurrent tasks. There are 4 main components to video, which can be divided into 2 types of video. Tile graphics are displayed on 16-pixel boundaries on the screen, and are displayed in monochrome. On the other hand, sprite graphics are displayed at any arbitrary pixel location and can be poly-chromatic.

Component	Type	Requirements
Maze	Tile	Draw
Score	Tile	Update scores
Dots	Tile/Sprite	Read when to
Players	Sprite	Display at

Since two graphics cannot be expressed by the same pixel, a priority of layers is established as follows (lowest priority to highest): pellets, maze walls, scores, player 2, player 1. Game logic does not allow certain graphics to overlap, however if such overlap occurs hardware would draw according to the previously described priority.

Memory Addresses for the Video Controller Device

Address	Bit Range (MSB-	Description
0x00	31-16	Player 1's y-position
0x00	15-0	Player 1's x-position
0x01	31-16	Player 2's y-position
0x01	15-0	Player 2's x-position
0x02	31-12	Reserved
0x02	11-8	Player 1's score (100's place)
0x02	7-4	Player 1's score (10's place)
0x02	3-0	Player 1's score (1's place)
0x03	31-12	Reserved
0x03	11-8	Player 2's score (100's place)
0x03	7-4	Player 2's score (10's place)
0x03	3-0	Player 2's score (1's place)
0x04	31-26	Reserved
0x04	25-0	Pellet data for line 0
…	…	…
0x20	31-26	Reserved
0x20	25-0	Pellet data for line 27

Tiles

Tiles are always displayed on 16-pixel boundaries. Every time the pixel scanner (which determines what should be drawn in each pixel on screen) reaches a 16-pixel boundary, a process determines what tile graphic should be drawn. For each pixel within a particular 16x16 square, memory is accessed to determine if the pixel at the offset should be drawn as a color (which depends on the tile type) or black.

Maze Walls

Mazes are defined as an array of 30 40-bit values. Since each tile is 16x16 pixels, the maze definition covers the entire screen. Each bit defines whether that tile on the screen is an unpassable maze wall ('1') or a navigable maze path ('0'). For each maze tile, a hardware process checks its neighbors to determine which of the 6 possible maze graphics to use for that tile. From there the pixel output process (common to all graphics) will output the appropriate color (which is blue in this case).

The size types of maze wall are horizontal, vertical, and each of the 4 90° rotations of an L corner.

Close-up of Maze Wall

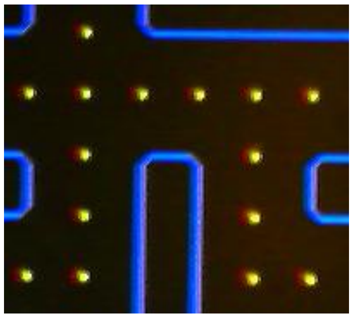
Close-up of Score

Score

By defining a simple custom font, which contains the digits from 0 to 9, it was possible to display the scores of each player on screen. The process which checks tile locations checks for certain locations on screen. By reading values from memory, Pac-Edwards software can update the scores.

Pellets

There is a RAM mapped to the pellets on screen, so the software can communicate which pellets are to be drawn. Each line of the game board has its own 32-bit word, which corresponds to the pellets on that line, with bit 0 being the right-most pellet. Like other tile graphics, the pixel scanning process will determine which pixel of the dot tile should be drawn, and will draw each pixel as either yellow or black, in order to form the shape of the small pellets.


Close-up of Pellets

For the Big Macs (the Power Pellets of Pac-Edwards), pixel placement is determined through the same tile process. However, drawing is done in a manner similar to player sprites.

Sprites

Since a sprite can be at any arbitrary position, some special machinery is required to determine pixel locations. A process exists which checks the location of each sprite and if within the 16-pixel block formed by the coordinates, calculates the displacement of the currently scanned pixel to determine a particular 4-bit value representing the color of that pixel.

These 4-bit values are then looked up by the pixel scanning process in a 16-color color table. Each sprite uses its own color-table, enabling each sprite to use a variety of colors.

Players

Player locations are read from memory, and controlled by software. The sprites for the players work exactly in the manner sprites are described.

The *Pac-Edwards* Team:

Varun Mehta: VGA Display and Game Logic
Mike Pierorazio: NES Input and Debugging
Jeffrey Cropsey: Documentation and Game Logic

A Note from Mike to Future Teams:

Getting the NES controllers to interface correctly with the board proved problematic. First of all I had to rely on information I found on the internet (that was often conflicting) with regards to the actual protocol. Another problem was that initially we tried to test the controller systems by running the software program and crossing our fingers. When it broke we had no idea where in the implementation it had broken down. By using a probe and an oscilloscope we were able to find and debug any issues with the current implementation (initially these actually included timing, pinning and voltage issues). Once all the kinks in data flow and hardware in general were ironed out everything quickly fell into place.

A Note from Jeffrey and Varun to Future Teams:

We were fortunate as our team started to work early on the project. This meant there was plenty of time when we ran into difficulties to seek help. It also meant that when our entire project was lost due to a system failure in the EE lab, we had time to rebuild the system from the beginning. This leads to our second lesson: use version control. Though we have used SVN for many other projects and even set up an SVN server for *Pac-Edwards*, we failed to commit much of the code. It could have saved us a lot of trouble had there been such a backup. Finally, when it comes to wiring hardware you didn't create yourself, the oscilloscope is your best friend. When trying to implement our hardware controller for the NES controller we encountered several false starts. By wiring the oscilloscope to the controllers we were able to see what signals the board and controller were producing and correct the errors. It is doubtful that we could have surmised the problem without its help as the glitch was a combination of wiring and faulty VHDL.

File Listings:

de2_sram_controler.vhd	//The SRAM Controller
hello_world.c	//The game logic
nes_ctlr.vhd	//The controller controller
tupac_video.vhd	//The VGA Controller
Tupacvhdl.vhd	//The top level entity

```
--DE2 SRAM
library IEEE;
use IEEE.std_logic_1164.all;

entity de2_sram_controller is

  port (
    signal avs_s1_clk,
         avs_s1_chipselect,
           avs_s1_write,
           avs_s1_read : in std_logic;
    signal avs_s1_address : in std_logic_vector(17 downto 0);
    signal avs_s1_readdata : out std_logic_vector(15 downto 0);
    signal avs_s1_writedata : in std_logic_vector(15 downto 0);
    signal avs_s1_byteenable : in std_logic_vector(1 downto 0);

    signal SRAM_DQ : inout std_logic_vector(15 downto 0);
    signal SRAM_ADDR : out std_logic_vector(17 downto 0);
    signal SRAM_UB_N,
           SRAM_LB_N,
           SRAM_WE_N,
           SRAM_CE_N,
           SRAM_OE_N : out std_logic
  );

end de2_sram_controller;

architecture datapath of de2_sram_controller is

begin

  SRAM_DQ <= avs_s1_writedata when avs_s1_write = '1' else
             (others => 'Z');
  avs_s1_readdata <= SRAM_DQ;
  SRAM_ADDR <= avs_s1_address;
  SRAM_UB_N <= not avs_s1_byteenable(1);
  SRAM_LB_N <= not avs_s1_byteenable(0);
  SRAM_WE_N <= not avs_s1_write;
  SRAM_CE_N <= not avs_s1_chipselect;
  SRAM_OE_N <= not avs_s1_read;

end datapath;
```

```
/*Hello_world.c - Game Logic*/

#include <io.h>
#include <stdio.h>
#include <system.h>
#include <sys/time.h>
#include <sys/alt_timestamp.h>

#define VIDEO_BASE 0x080A00
#define CONTROLLER_BASE 0x080808
#define POS_OFFSET 0
#define SCORE_OFFSET 8
#define DOTS_OFFSET 16
#define TOTAL_DOTS 100

#define SLOW_SPEED 1<<11
#define FAST_SPEED 1<<10
#define INIT_POWERUP_COUNTER 350
/*
 * "Hello World" example.
 *
 * This example prints 'Hello from Nios II' to the STDOUT stream. It
runs on
 * the Nios II 'standard', 'full_featured', 'fast', and 'low_cost'
example
 * designs. It runs with or without the MicroC/OS-II RTOS and requires
a STDOUT
 * device in your system's hardware.
 * The memory footprint of this hosted application is ~69 kbytes by
default
 * using the standard reference design.
 *
 * For a reduced footprint version of this template, and an
explanation of how
 * to reduce the memory footprint for a given application, see the
 * "small_hello_world" template.
 *
 */
 #define IOWR_POS(player, position) \
  IOWR_32DIRECT(VIDEO_BASE+POS_OFFSET, player*4, position)
 #define IOWR_SCORE(player, data) \
  IOWR_32DIRECT(VIDEO_BASE+SCORE_OFFSET, player*4, data)
 #define IOWR_DOTS(line, dots) \
  IOWR_32DIRECT(VIDEO_BASE+DOTS_OFFSET, line*4, dots)
 #define IO_READ_NES(player) \
  IORD_8DIRECT(CONTROLLER_BASE, player)

 #define LEFT 0
 #define UP 1
 #define RIGHT 2
 #define DOWN 3
 #define NONE 4

unsigned long maze[30] = {0x0FFFFFFF, 0x08006001, 0x0BDF6FBD,
                          0x0BDF6FBD, 0x08000001, 0x0BDBFDBD,
                          0x0BDBFDBD, 0x08186181, 0x0FDF6FBF,
                          0x0FDF6FBF, 0x0FD801BF, 0x0FDBFDBF,
                          0x0FDBFDBF, 0x0803FC01, 0x0FDBFDBF,
                          0x0FDBFDBF, 0x0FD801BF, 0x0FDBFDBF,
                          0x0FDBFDBF, 0x08006001, 0x0BDF6FBD,
                          0x0BDF6FBD, 0x08C00031, 0x0EDBFDB7,
                          0x0EDBFDB7, 0x08186181, 0x0BFF6FFD,
                          0x0BFF6FFD, 0x08000001, 0x0FFFFFFF};
unsigned long dots[30] = {0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF,
                          0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF,
                          0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF,
                          0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF,
                          0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF,
                          0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF,
                          0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF,
                          0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF,
                          0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF,
                          0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF};
```

```
int move(int player, int pos_x, int pos_y, int direction) {
    unsigned int maze_x, maze_y;
    unsigned int maze_x2, maze_y2;
    if (direction == LEFT) pos_x--;
    if (direction == UP) pos_y--;
    if (direction == RIGHT) pos_x++;
    if (direction == DOWN) pos_y++;
    maze_x = (pos_x >> 4) - 5;
    maze_y = (pos_y >> 4);
    maze_x2 = ((pos_x+15) >> 4) - 5;
    maze_y2 = ((pos_y+15) >> 4);

//    printf(" tile (%d,%d)\n",maze_x,maze_y);
    //printf("new spot: %xu", (maze[maze_y] & (1 << (28-maze_x))));
    if ((maze[maze_y] & (1 << (28-maze_x)))) //if there is a maze tile
where we want to go.
        return 0;
    if ((maze[maze_y] & (1 << (28-maze_x2))))
        return 0;
    if ((maze[maze_y2] & (1 << (28-maze_x)))) //if there is a maze
tile where we want to go.
        return 0;
    if ((maze[maze_y2] & (1 << (28-maze_x2))))
        return 0;
//    printf("moving to x:%d y%d\n", pos_x, pos_y);
    IOWR_POS(player, pos_x+(pos_y<<16));
    return 1;
}

int transform_board_to_screen_x(board_x) {return (board_x + 6) * 16;}
int transform_screen_to_board_x(screen_x) {return (screen_x / 16) -
6;}
int transform_board_to_screen_y(board_y) {return (board_y) *16;}
int transform_screen_to_board_y(screen_y) {return (screen_y / 16);}

int transform_screen_to_dots_x(screen_x) {return (screen_x/16 - 6);}
int transform_screen_to_dots_y(screen_y) {return (screen_y / 16) - 1;}

int read_direction(int player) {
    int status = IO_READ_NES(player);
    if (status & 0x80) return RIGHT;
    else if (status & 0x20) return DOWN;
    else if (status & 0x40) return LEFT;
    else if (status & 0x10) return UP;
    else return NONE;
}

int reset_condition() {
    int status1 = IO_READ_NES(0), status2 = IO_READ_NES(1);
    if ((status1 & 0x08) && (status2 & 0x08)) return 1;
    return 0;
}

int reset_game() {
    unsigned int x1 = transform_board_to_screen_x(1), y1 =
transform_board_to_screen_y(13),
                 x2 = transform_board_to_screen_x(26), y2 =
transform_board_to_screen_y(13), i;
    IOWR_SCORE(0, 0x00000000);
    IOWR_SCORE(1, 0x00000000);
    for (i = 0; i <= 27; i++) {
        IOWR_DOTS(i,0xFFFFFFFF);
        dots[i] = 0xFFFFFFFF;
    }
    IOWR_POS(0, x1+(y1<<16));
    IOWR_POS(1, x2+(y2<<16));
}

int main()
{
  srand();
  int i = 0;
  unsigned int x1 = transform_board_to_screen_x(1), y1 =
transform_board_to_screen_y(13),
               x2 = transform_board_to_screen_x(26), y2 =
transform_board_to_screen_y(13);
  unsigned int hitx, hity, hitdot;
  unsigned int p1_score = 0, p2_score = 0;
  unsigned int p1_counter = 0, p2_counter = 0, p1_threshold =
SLOW_SPEED, p2_threshold = SLOW_SPEED;
  unsigned int p1_powerup_counter = 0, p2_powerup_counter = 0;
  unsigned int last_direction_1 = NONE, last_direction_2 = NONE, temp;
  unsigned int next_direction_1 = NONE, next_direction_2 = NONE;
  int move_result = 0;
    IOWR_SCORE(0, 0x00000000);
    IOWR_SCORE(1, 0x00000000);
    for (i = 0; i <= 27; i++) {
        IOWR_DOTS(i,dots[i]);
    }
    IOWR_POS(0, x1+(y1<<16));
    while (1) {
        p1_counter++;
        p2_counter++;
        p1_counter %= p1_threshold;
        p2_counter %= p2_threshold;
        //write player_1 score
        if (p1_counter == 0) {
            printf("p1: %x\n", IO_READ_NES(0));
            if (p1_powerup_counter > 0) p1_powerup_counter--;
            else p1_threshold = SLOW_SPEED;
            IOWR_SCORE(0, p1_score%10 + (((p1_score/10)%10)<<4) +
(((p1_score/100)%10)<<8));
            if((dots[transform_screen_to_dots_y(y1+6)] &
(1<<transform_screen_to_dots_x(x1+8)))) {
               hitx = x1+8;
               hity = y1+6;
               hitdot = 1;
            }
            else if(dots[transform_screen_to_dots_y(y1+8)] &
(1<<transform_screen_to_dots_x(x1+6))) {
                hitx = x1+6;
                hity = y1+8;
                hitdot = 1;
            }
            else if(dots[transform_screen_to_dots_y(y1+10)] &
(1<<transform_screen_to_dots_x(x1+8))) {
                hitx = x1+8;
                hity = y1+10;
                hitdot = 1;
            }
            else if(dots[transform_screen_to_dots_y(y1+8)] &
(1<<transform_screen_to_dots_x(x1+10))) {
                hitx = x1+10;
                hity = y1+8;
                hitdot = 1;
            }
            else hitdot = 0;
//                printf("x1: %d, y1: %d, dots: %d,%d , maze_x: %d,
maze_y: %d\n", x1, y1, transform_screen_to_dots_x(x1),
transform_screen_to_dots_y(y1), transform_screen_to_board_x(x1),
transform_screen_to_board_y(y1));
            if (hitdot == 1) {
                if (transform_screen_to_dots_y(hity) == 1 &&
transform_screen_to_dots_x(hitx) == 26)
                    {p1_threshold = FAST_SPEED; p1_powerup_counter =
INIT_POWERUP_COUNTER;}
                if (transform_screen_to_dots_y(hity) == 24 &&
transform_screen_to_dots_x(hitx) == 26)
                    {p1_threshold = FAST_SPEED; p1_powerup_counter =
INIT_POWERUP_COUNTER;}
                if (transform_screen_to_dots_y(hity) == 1 &&
transform_screen_to_dots_x(hitx) == 1)
                    {p1_threshold = FAST_SPEED; p1_powerup_counter =
INIT_POWERUP_COUNTER;}
                if (transform_screen_to_dots_y(hity) == 24 &&
transform_screen_to_dots_x(hitx) == 1)
                    {p1_threshold = FAST_SPEED; p1_powerup_counter =
INIT_POWERUP_COUNTER;}

                dots[transform_screen_to_dots_y(hity)] =
dots[transform_screen_to_dots_y(hity)] &
~(1<<transform_screen_to_dots_x(hitx));
                IOWR_DOTS(transform_screen_to_dots_y(hity),
dots[transform_screen_to_dots_y(hity)]);
                p1_score++;
            }
            temp = read_direction(0);
            if (temp != NONE) next_direction_1 = temp;
            move_result = move(0, x1, y1, next_direction_1);
            if (move_result != 0) last_direction_1 = next_direction_1;
            else move_result = move(0, x1, y1, last_direction_1);
            if (move_result != 0) {
                switch (last_direction_1) {
                 case LEFT:
                   x1--;
                   break;
                 case UP:
                   y1--;
                   break;
                 case RIGHT:
                   x1++;
                   break;
                 case DOWN:
                   y1++;
                   break;
                }
            }
            if (reset_condition()) {
                last_direction_1 = NONE;
                last_direction_2 = NONE;
                next_direction_1 = NONE; next_direction_2 = NONE;
                x1 = transform_board_to_screen_x(1); y1 =
transform_board_to_screen_y(13);
                x2 = transform_board_to_screen_x(26); y2 =
transform_board_to_screen_y(13);
                p1_score = 0; p2_score = 0; p1_threshold = SLOW_SPEED;
p2_threshold = SLOW_SPEED;
                p1_powerup_counter = 0; p2_powerup_counter = 0;
                reset_game();
            }
        }
        if (p2_counter == 0) {
            printf("p2: %x\n", IO_READ_NES(1));
            //write player_2 score
            IOWR_SCORE(1, p2_score%10 + (((p2_score/10)%10)<<4) +
(((p2_score/100)%10)<<8));
            if (p2_powerup_counter > 0) p2_powerup_counter--;
            else p2_threshold = SLOW_SPEED;

            if((dots[transform_screen_to_dots_y(y2+6)] &
(1<<transform_screen_to_dots_x(x2+8)))) {
               hitx = x2+8;
               hity = y2+6;
               hitdot = 1;
            }
            else if(dots[transform_screen_to_dots_y(y2+8)] &
(1<<transform_screen_to_dots_x(x2+6))) {
                hitx = x2+6;
```

```
                hity = y2+8;
                hitdot = 1;
            }
            else if(dots[transform_screen_to_dots_y(y2+10)] &
(1<<transform_screen_to_dots_x(x2+8))) {
                hitx = x2+8;
                hity = y2+10;
                hitdot = 1;
            }
            else if(dots[transform_screen_to_dots_y(y2+8)] &
(1<<transform_screen_to_dots_x(x2+10))) {
                hitx = x2+10;
                hity = y2+8;
                hitdot = 1;
            }
            else hitdot = 0;
//                printf("x1: %d, y1: %d, dots: %d,%d , maze_x: %d,
maze_y: %d\n", x1, y1, transform_screen_to_dots_x(x1),
transform_screen_to_dots_y(y1), transform_screen_to_board_x(x1),
transform_screen_to_board_y(y1));
            if (hitdot == 1) {
                if (transform_screen_to_dots_y(y2) == 1 &&
transform_screen_to_dots_x(x2) == 26)
                    {p2_threshold = FAST_SPEED; p2_powerup_counter =
INIT_POWERUP_COUNTER;}
                if (transform_screen_to_dots_y(y2) == 24 &&
transform_screen_to_dots_x(x2) == 26)
                    {p2_threshold = FAST_SPEED; p2_powerup_counter =
INIT_POWERUP_COUNTER;}
                if (transform_screen_to_dots_y(y2) == 1 &&
transform_screen_to_dots_x(x2) == 1)
                    {p2_threshold = FAST_SPEED; p2_powerup_counter =
INIT_POWERUP_COUNTER;}
                if (transform_screen_to_dots_y(y2) == 24 &&
transform_screen_to_dots_x(x2) == 1)
                    {p2_threshold = FAST_SPEED; p2_powerup_counter =
INIT_POWERUP_COUNTER;}

                dots[transform_screen_to_dots_y(hity)] =
dots[transform_screen_to_dots_y(hity)] &
~(1<<transform_screen_to_dots_x(hitx));
```

```
                IOWR_DOTS(transform_screen_to_dots_y(hity),
dots[transform_screen_to_dots_y(hity)]);
                p2_score++;
            }
            temp = read_direction(1);
            if (temp != NONE) next_direction_2 = temp;
            move_result = move(1, x2, y2, next_direction_2);
            if (move_result != 0) last_direction_2 = next_direction_2;
            else move_result = move(1, x2, y2, last_direction_2);
            if (move_result != 0) {
                switch (last_direction_2) {
                 case LEFT:
                   x2--;
                   break;
                 case UP:
                   y2--;
                   break;
                 case RIGHT:
                   x2++;
                   break;
                 case DOWN:
                   y2++;
                   break;
                }
            }
        }
    }
    return 0;
}
```

```
--NES Controller Controller
--thanks to http://www.mit.edu/~tarvizo/nes-controller.html

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity nes_ctlr is
     port (
         avs_s1_clk        : in  std_logic;
```

```
        avs_s1_reset_n    : in  std_logic;
    avs_s1_read       : in  std_logic;
        avs_s1_write      : in  std_logic;
         avs_s1_chipselect : in  std_logic;
        avs_s1_address    : in  std_logic;
    avs_s1_readdata   : out std_logic_vector(7 downto 0);
        avs_s1_writedata  : in  std_logic_vector(7 downto 0);

         latch1: out std_logic;
         latch2: out std_logic;

         pulse1: out std_logic;
         pulse2: out std_logic;

         data1: in std_logic; --data
         data2: in std_logic --data

         );
end nes_ctlr;

architecture fsm of nes_ctlr is
    constant w : integer range 0 to 17 := 0;
   constant w0 : integer range 0 to 17 := 1;
   constant b0 : integer range 0 to 17 := 2;
   constant w1 : integer range 0 to 17 := 3;
   constant b1 : integer range 0 to 17 := 4;
   constant w2 : integer range 0 to 17 := 5;
   constant b2 : integer range 0 to 17 := 6;
   constant w3 : integer range 0 to 17 := 7;
   constant b3 : integer range 0 to 17 := 8;
   constant w4 : integer range 0 to 17 := 9;
   constant b4 : integer range 0 to 17 := 10;
   constant w5 : integer range 0 to 17 := 11;
   constant b5 : integer range 0 to 17 := 12;
   constant w6 : integer range 0 to 17 := 13;
   constant b6 : integer range 0 to 17 := 14;
   constant w7 : integer range 0 to 17 := 15;
   constant b7 : integer range 0 to 17 := 16;
   constant w8 : integer range 0 to 17 := 17;

    signal clk_60Hz: std_logic;
    signal prev_clk_60Hz: std_logic;
```

```
      signal clk_41KHz: std_logic;
      signal clk_83KHz: std_logic;
      signal counter_60Hz : integer range 0 to 8500000 := 0;
      signal counter_41KHz : integer range 0 to 300 := 0;
      signal counter_83KHz : integer range 0 to 600 := 0;
     signal p_s, n_s : integer range 0 to 17 := 0;
      signal player1: std_logic_vector (7 downto 0); --data
      signal player2: std_logic_vector (7 downto 0); --data
      signal latch_counter: integer range 0 to 300 := 0;
      signal pulse_counter: integer range 0 to 5400 := 0;
      signal latch_sig: std_logic;
      signal pulse_sig: std_logic;

begin

      Clock_60Hz:process(avs_s1_clk)
      begin
            if rising_edge(avs_s1_clk) then
                  prev_clk_60Hz <= clk_60Hz;
--                if (counter_60Hz = 15000) then
                  if (counter_60Hz = 416667) then
                        if (clk_60Hz = '1') then
                              clk_60Hz <= '0';
--                            prev_clk_60Hz <= '0';
--                            counter_60Hz <= counter_60Hz + 1;
                              counter_60Hz <= 0;
                        else
                              clk_60Hz <= '1';
                              counter_60Hz <= 0;
                        end if;
                  else
                        counter_60Hz <= counter_60Hz + 1;
                  end if;
            end if;
      end process Clock_60Hz;

      Clock_41KHz:process(avs_s1_clk)
      begin
            if rising_edge(avs_s1_clk) then
                  if (counter_41KHz = 150) then
                        if (clk_41KHz = '1') then
                              clk_41KHz <= '0';
                        counter_41KHz <= 0;
                    else
                        clk_41KHz <= '1';
                        counter_41KHz <= 0;
                    end if;
                else
                    counter_41KHz <= counter_41KHz + 1;
                end if;
            end if;
        end process Clock_41KHz;

        Clock_83KHz:process(avs_s1_clk)
        begin
            if rising_edge(avs_s1_clk) then
                if (counter_83KHz = 300) then
                    if (clk_83KHz = '1') then
                        clk_83KHz <= '0';
                        counter_83KHz <= 0;
                    else
                        clk_83KHz <= '1';
                        counter_83KHz <= 0;
                    end if;
                else
                    counter_83KHz <= counter_83KHz + 1;
                end if;
            end if;
        end process Clock_83KHz;

        Latch_Control:process (avs_s1_clk)
        begin
            if rising_edge (avs_s1_clk) then
                if prev_clk_60Hz = '0' and clk_60Hz = '1' and
    latch_counter < 300 then
                    latch1 <= '1';
                    latch2 <= '1';
                    latch_sig <= '1';
                    latch_counter <= latch_counter + 1;
                elsif clk_60Hz = '1' and latch_counter < 300 and
    latch_counter > 0 then
                    latch1 <= '1';
                    latch2 <= '1';
                    latch_sig <= '1';
```

```
                        latch_counter <= latch_counter + 1;
                    --elsif clk_60Hz = '0' and latch_counter <  then
                    --    latch1 <= '0';
                    --    latch2 <= '0';
                    --    latch_sig <= '0';
                    --    latch_counter <= latch_counter + 1;

                    else
                        latch1 <= '0';
                        latch2 <= '0';
                        latch_sig <= '0';
                        latch_counter <= 0;
                    end if;
                end if;
            end process Latch_Control;

            Pulse_Control:process (avs_s1_clk)
            begin
            if rising_edge (avs_s1_clk) then
                if prev_clk_60Hz = '0' and clk_60Hz = '1' and pulse_counter
        = 0 then
                    pulse_counter <= 1;
                    pulse1 <= '0';
                    pulse2 <= '0';
                    pulse_sig <= '0';
                elsif pulse_counter = 0 then
                    pulse_counter <= 0;
                    pulse1 <= '0';
                    pulse2 <= '0';
                    pulse_sig <= '0';
                elsif pulse_counter < 600 then
                    pulse_counter <= pulse_counter + 1;
                    pulse1 <= '0';
                    pulse2 <= '0';
                    pulse_sig <= '0';
                elsif pulse_counter < 900 then --1
                    pulse1 <= '1';
                    pulse2 <= '1';
                    pulse_sig <= '1';
                    pulse_counter <= pulse_counter + 1;
                elsif pulse_counter < 1200 then
                    pulse1 <= '0';
                    pulse2 <= '0';
                    pulse_sig <= '0';
                    pulse_counter <= pulse_counter + 1;
                elsif pulse_counter < 1500 then --2
                    pulse1 <= '1';
                    pulse2 <= '1';
                    pulse_sig <= '1';
                    pulse_counter <= pulse_counter + 1;
                elsif pulse_counter < 1800 then
                    pulse1 <= '0';
                    pulse2 <= '0';
                    pulse_sig <= '0';
                    pulse_counter <= pulse_counter + 1;
                elsif pulse_counter < 2100 then --3
                    pulse1 <= '1';
                    pulse2 <= '1';
                    pulse_sig <= '1';
                    pulse_counter <= pulse_counter + 1;
                elsif pulse_counter < 2400 then
                    pulse1 <= '0';
                    pulse2 <= '0';
                    pulse_sig <= '0';
                    pulse_counter <= pulse_counter + 1;
                elsif pulse_counter < 2700 then --4
                    pulse1 <= '1';
                    pulse2 <= '1';
                    pulse_sig <= '1';
                    pulse_counter <= pulse_counter + 1;
                elsif pulse_counter < 3000 then
                    pulse1 <= '0';
                    pulse2 <= '0';
                    pulse_sig <= '0';
                    pulse_counter <= pulse_counter + 1;
                elsif pulse_counter < 3300 then --5
                    pulse1 <= '1';
                    pulse2 <= '1';
                    pulse_sig <= '1';
                    pulse_counter <= pulse_counter + 1;
                elsif pulse_counter < 3600 then
                    pulse1 <= '0';
                    pulse2 <= '0';
                    pulse_sig <= '0';
```

```
                pulse_counter <= pulse_counter + 1;
            elsif pulse_counter < 3900 then --6
                pulse1 <= '1';
                pulse2 <= '1';
                pulse_sig <= '1';
                pulse_counter <= pulse_counter + 1;
            elsif pulse_counter < 4200 then
                pulse1 <= '0';
                pulse2 <= '0';
                pulse_sig <= '0';
                pulse_counter <= pulse_counter + 1;
            elsif pulse_counter < 4500 then --7
                pulse1 <= '1';
                pulse2 <= '1';
                pulse_sig <= '1';
                pulse_counter <= pulse_counter + 1;
--          elsif pulse_counter < 4800 then
--              pulse1 <= '0';
--              pulse2 <= '0';
--              pulse_sig <= '0';
--              pulse_counter <= pulse_counter + 1;
--          elsif pulse_counter < 5100 then --8
--              pulse1 <= '1';
--              pulse2 <= '1';
--              pulse_sig <= '1';
--              pulse_counter <= pulse_counter + 1;
            else --9
                pulse1 <= '0';
                pulse2 <= '0';
                pulse_sig <= '0';
                pulse_counter <= 0;
            end if;
        end if;
        end process Pulse_Control;

        fsm_process:process(p_s, latch_sig, pulse_sig)
        begin
          case p_s is
             when w =>
                if latch_sig = '1' then
                   n_s <= w0;
                else
            n_s <= w;
        end if;
    when w0 =>
        if latch_sig = '0' then
            n_s <= b0;
            player1(0) <= not data1;
            player2(0) <= not data2;
        else
            n_s <= w0;
        end if;
    when b0 =>
        if pulse_sig = '1' then
            n_s <= w1;
        else
            n_s <= b0;
        end if;
    when w1 =>
        if pulse_sig = '0' then
            n_s <= b1;
            player1(1) <= not data1;
            player2(1) <= not data2;
        else
            n_s <= w1;
        end if;
    when b1 =>
        if pulse_sig = '1' then
            n_s <= w2;
        else
            n_s <= b1;
        end if;
    when w2 =>
        if pulse_sig = '0' then
            n_s <= b2;
            player1(2) <= not data1;
            player2(2) <= not data2;
        else
            n_s <= w2;
        end if;
    when b2 =>
        if pulse_sig = '1' then
            n_s <= w3;
        else
```

```
                n_s <= b2;
            end if;
        when w3 =>
            if pulse_sig = '0' then
                n_s <= b3;
                player1(3) <= not data1;
                player2(3) <= not data2;
            else
                n_s <= w3;
            end if;
        when b3 =>
            if pulse_sig = '1' then
                n_s <= w4;
            else
                n_s <= b3;
            end if;
        when w4 =>
            if pulse_sig = '0' then
                n_s <= b4;
                player1(4) <= not data1;
                player2(4) <= not data2;
            else
                n_s <= w4;
            end if;
        when b4 =>
            if pulse_sig = '1' then
                n_s <= w5;
            else
                n_s <= b4;
            end if;
        when w5 =>
            if pulse_sig = '0' then
                n_s <= b5;
                player1(5) <= not data1;
                player2(5) <= not data2;
            else
                n_s <= w5;
            end if;
        when b5 =>
            if pulse_sig = '1' then
                n_s <= w6;
            else
```

```
                n_s <= b5;
            end if;
        when w6 =>
            if pulse_sig = '0' then
                n_s <= b6;
                player1(6) <= not data1;
                player2(6) <= not data2;
            else
                n_s <= w6;
            end if;
        when b6 =>
            if pulse_sig = '1' then
                n_s <= w7;
            else
                n_s <= b6;
            end if;
        when w7 =>
            if pulse_sig = '0' then
                n_s <= b7;
                player1(7) <= not data1;
                player2(7) <= not data2;
            else
                n_s <= w7;
            end if;
--          when b7 =>
--              if pulse_sig = '1' then
--                  n_s <= w8;
--              else
--                  n_s <= b7;
--              end if;
--          when w8 =>
--              if pulse_sig = '0' then
--                  n_s <= w;
--              else
--                  n_s <= w8;
--              end if;
        when others =>
            if latch_sig = '1' then
                n_s <= w0;
            end if;
    end case;
  end process fsm_process;
```

```
      state_transition:process (avs_s1_clk)
      begin
            if rising_edge(avs_s1_clk) then
                  p_s <= n_s;
            end if;
      end process state_transition;

      process (avs_s1_clk)
      begin
            if avs_s1_reset_n = '0' then
            avs_s1_readdata <= (others => '0');
            else
            if avs_s1_chipselect = '1' then
                  if avs_s1_read = '1' then
                              if avs_s1_address = '0' then
                              avs_s1_readdata <= player1;
                              else
                                    avs_s1_readdata <= player2;
                              end if;
                  end if;
            end if;
            end if;
      end process;

end fsm;
```

```
--tupac_video.vhd

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity avalon_video is
  port (
    avs_s1_clk        : in  std_logic;
    avs_s1_reset_n    : in  std_logic;
    avs_s1_read       : in  std_logic;
    avs_s1_write      : in  std_logic;
    avs_s1_chipselect : in  std_logic;
    avs_s1_address    : in  std_logic_vector(6 downto 0);
    avs_s1_readdata   : out std_logic_vector(31 downto 0);
    avs_s1_writedata  : in  std_logic_vector(31 downto 0);

     reset_n      : in std_logic;                             -- More
reset!
     CLK_25       : in std_logic;
    VGA_CLK   : out std_logic;                      -- Clock
    VGA_HS    : out std_logic;                      -- H_SYNC
    VGA_VS    : out std_logic;                      -- V_SYNC
    VGA_BLANK : out std_logic;                      -- BLANK
    VGA_SYNC  : out std_logic;                      -- SYNC
    VGA_R     : out std_logic_vector(9 downto 0); -- Red[9:0]
    VGA_G     : out std_logic_vector(9 downto 0); -- Green[9:0]
    VGA_B     : out std_logic_vector(9 downto 0)  -- Blue[9:0]
  );
end avalon_video;

architecture rtl of avalon_video is

signal clk : std_logic;

--Declare our various RAMs and ROMs
type maze_ram_line_type is array(40 downto -1) of std_logic;
type maze_ram_type is array(-1 to 30) of maze_ram_line_type;

type maze_sprite_ram_type is array(0 to 7) of std_logic_vector(255
downto 0);
type show_dot_ram_type is array(27 downto 0) of std_logic_vector(27
downto 0);
type color_ram_type is array (15 downto 0) of std_logic_vector(29
downto 0);

type player_sprite_ram_line_type is array(0 to 15) of
std_logic_vector(0 to 3);
type player_sprite_ram_type is array(0 to 15) of
player_sprite_ram_line_type;

type score_ram_line_type is array(2 downto 0) of std_logic_vector(3
downto 0);
type score_ram_type is array(1 downto 0) of score_ram_line_type;
type sprite_ram_line_type is array(0 to 15) of std_logic_vector(0 to
15);
type dot_sprite_ram_type is array(0 to 1) of sprite_ram_line_type;
type num_sprite_ram_type is array(0 to 10) of sprite_ram_line_type;

signal maze_sprite_ram: maze_sprite_ram_type;
signal dot_sprite_ram: dot_sprite_ram_type;
signal num_sprite_ram: num_sprite_ram_type;
signal score_ram: score_ram_type;
signal show_dot_ram: show_dot_ram_type;
signal RAM : maze_ram_type;
signal color_ram_1, color_ram_2, color_ram_dot: color_ram_type;

signal player_sprite_ram: player_sprite_ram_type;
signal player2_sprite_ram: player_sprite_ram_type;
signal bigdot_sprite_ram: player_sprite_ram_type;

signal ram_address, display_address : std_logic_vector(6 downto 0);
signal show_big_dot: std_logic;

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

constant VTOTAL       : integer := 525;
constant VSYNC        : integer := 2;
constant VBACK_PORCH  : integer := 33;
constant VACTIVE      : integer := 480;
constant VFRONT_PORCH : integer := 10;

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

signal vga_hblank, vga_hsync,
 vga_vblank, vga_vsync : std_logic;  -- Sync. signals
```

```
signal visible_xpos: std_logic_vector(9 downto 0);
signal visible_ypos: std_logic_vector(9 downto 0);

signal sprite_x: std_logic_vector(3 downto 0);
signal sprite_y: std_logic_vector(3 downto 0);

signal player_1_xpos, player_1_ypos, player_2_xpos, player_2_ypos:
std_logic_vector(9 downto 0);
signal draw_player_1, draw_player_2 : std_logic;

signal maze_x: std_logic_vector(5 downto 0);
signal maze_y: std_logic_vector(4 downto 0);
signal maze : std_logic;
signal print_char : std_logic;
signal maze_line_from_RAM : maze_ram_line_type;
signal left_maze, right_maze, up_maze, down_maze, downleft_maze,
upleft_maze, upright_maze, downright_maze: std_logic;

signal current_maze_sprite_from_RAM : std_logic_vector(255 downto 0);
signal current_dot_sprite_from_RAM : sprite_ram_line_type;
signal current_char_sprite_from_RAM : sprite_ram_line_type;

begin

  clk <= avs_s1_clk;
--  reset_n <= avs_s1_reset_n;
  ram_address <= avs_s1_address;
  visible_xpos <= Hcount - HSYNC - HBACK_PORCH;
  maze_x <= visible_xpos(9 downto 4);
  sprite_x <= visible_xpos(3 downto 0);
  left_maze <= RAM(conv_integer(maze_y))(conv_integer(maze_x)-1);
  right_maze <= RAM(conv_integer(maze_y))(conv_integer(maze_x)+1);
  up_maze <= RAM(conv_integer(maze_y)-1)(conv_integer(maze_x));
  down_maze <= RAM(conv_integer(maze_y)+1)(conv_integer(maze_x));
  downleft_maze <= RAM(conv_integer(maze_y)+1)(conv_integer(maze_x)-
1);
  downright_maze <=
RAM(conv_integer(maze_y)+1)(conv_integer(maze_x)+1);
  upleft_maze <= RAM(conv_integer(maze_y)-1)(conv_integer(maze_x)-1);
  upright_maze <= RAM(conv_integer(maze_y)-1)(conv_integer(maze_x)+1);

  CheckDrawPlayers: process (clk)
  begin
      if visible_xpos < player_1_xpos+16 and visible_xpos >=
player_1_xpos and visible_ypos < player_1_ypos+16 and visible_ypos >=
player_1_ypos then
           draw_player_1 <= '1';
           draw_player_2 <= '0';
      elsif visible_xpos < player_2_xpos+16 and visible_xpos >=
player_2_xpos and visible_ypos < player_2_ypos+16 and visible_ypos >=
player_2_ypos then
           draw_player_1 <= '0';
           draw_player_2 <= '1';
      else
           draw_player_1 <= '0';
           draw_player_2 <= '0';
      end if;
  end process;

  MazeSprite: process (clk)
  begin
       if up_maze = '1' and down_maze = '1' and left_maze = '0' and
right_maze = '1' then
           current_maze_sprite_from_RAM <= maze_sprite_ram(6);
       elsif up_maze = '1' and down_maze = '1' and left_maze = '1' and
right_maze = '0' then
           current_maze_sprite_from_RAM <= maze_sprite_ram(6);

       elsif up_maze = '1' and right_maze = '1' and upright_maze = '1'
and downleft_maze = '0' and down_maze = '0' and left_maze = '0' then
           current_maze_sprite_from_RAM <= maze_sprite_ram(1);
       elsif up_maze = '1' and right_maze = '1' and upright_maze = '0'
and downleft_maze = '1' and down_maze = '1' and left_maze = '1' then
           current_maze_sprite_from_RAM <= maze_sprite_ram(1);

       elsif down_maze = '1' and right_maze = '1' and downright_maze =
'1' and left_maze = '0' and upleft_maze = '0' then
           current_maze_sprite_from_RAM <= maze_sprite_ram(2);
       elsif down_maze = '1' and right_maze = '1' and downright_maze =
'0' and left_maze = '1' and upleft_maze = '1' then
           current_maze_sprite_from_RAM <= maze_sprite_ram(2);

       elsif up_maze = '1' and left_maze = '1' and upleft_maze = '1'
and downright_maze = '0' and down_maze = '0' and right_maze = '0' then
```

```
            current_maze_sprite_from_RAM <= maze_sprite_ram(3);
        elsif up_maze = '1' and left_maze = '1' and upleft_maze = '0'
and upleft_maze = '0' and downright_maze = '1' and down_maze = '1' and
downright_maze = '1' then
            current_maze_sprite_from_RAM <= maze_sprite_ram(3);

        elsif left_maze = '1' and down_maze = '1' and upright_maze =
'0' and right_maze = '0' and downleft_maze = '1' then
            current_maze_sprite_from_RAM <= maze_sprite_ram(4);
        elsif left_maze = '1' and down_maze = '1' and upright_maze =
'1' and right_maze = '1' and downleft_maze = '0' then
            current_maze_sprite_from_RAM <= maze_sprite_ram(4);

        elsif left_maze = '1' and right_maze = '1' and up_maze = '0'
and down_maze = '1' then
            current_maze_sprite_from_RAM <= maze_sprite_ram(5);
        elsif left_maze = '1' and right_maze = '1' and up_maze = '1'
and down_maze = '0' then
            current_maze_sprite_from_RAM <= maze_sprite_ram(5);

       else
            current_maze_sprite_from_RAM <= maze_sprite_ram(7);
        end if;


        if maze_x = "000010" and maze_y = "00010" then
            print_char <= '1';
            current_char_sprite_from_RAM <= num_sprite_ram(1);
        elsif maze_x = "100010" and maze_y = "00010" then
            print_char <= '1';
            current_char_sprite_from_RAM <= num_sprite_ram(2);
        elsif maze_x = "000010" and maze_y = "00011" then
            print_char <= '1';
            current_char_sprite_from_RAM <=
num_sprite_ram(conv_integer(score_ram(0)(0)));
        elsif maze_x = "000011" and maze_y = "00011" then
            print_char <= '1';
            current_char_sprite_from_RAM <=
num_sprite_ram(conv_integer(score_ram(0)(1)));
        elsif maze_x = "000100" and maze_y = "00011" then
            print_char <= '1';
            current_char_sprite_from_RAM <=
num_sprite_ram(conv_integer(score_ram(0)(2)));
        elsif maze_x = "100010" and maze_y = "00011" then
            print_char <= '1';
            current_char_sprite_from_RAM <=
num_sprite_ram(conv_integer(score_ram(1)(0)));
        elsif maze_x = "100011" and maze_y = "00011" then
            print_char <= '1';
            current_char_sprite_from_RAM <=
num_sprite_ram(conv_integer(score_ram(1)(1)));
        elsif maze_x = "100100" and maze_y = "00011" then
            print_char <= '1';
            current_char_sprite_from_RAM <=
num_sprite_ram(conv_integer(score_ram(1)(2)));
        else
            print_char <= '0';
        end if;

        if maze_x = 7 and maze_y = 2 then
            current_dot_sprite_from_RAM <= dot_sprite_ram(1);
            show_big_dot <= '1';
        elsif maze_x = 7 and maze_y = 25 then
            current_dot_sprite_from_RAM <= dot_sprite_ram(1);
            show_big_dot <= '1';
        elsif maze_x = 32 and maze_y = 2 then
            current_dot_sprite_from_RAM <= dot_sprite_ram(1);
            show_big_dot <= '1';
        elsif maze_x = 32 and maze_y = 25 then
            current_dot_sprite_from_RAM <= dot_sprite_ram(1);
            show_big_dot <= '1';
        else
            current_dot_sprite_from_RAM <= dot_sprite_ram(0);
            show_big_dot <= '0';
        end if;
  end process;

  process (clk)
  begin
      --all of these registers end up being reduced, so no worries :-)
I'll try to clean it up later, though.
    if clk'event and clk = '1' then
        RAM(-1)<=     "1111111111111111111111111111111111111111";
```

```
       RAM(0) <= "111111111111111111111111111111111111111111";
       RAM(1) <=     "111111110000000000001100000000000011111111";
       RAM(2) <= "111111110111101111101101111101111011111111";
       RAM(3) <= "111111110111101111101101111101111011111111";
       RAM(4) <= "111111110000000000000000000000000011111111";
       RAM(5) <= "111111110111101101111111101101111011111111";
       RAM(6) <=     "111111110111101101111111101101111011111111";
       RAM(7) <= "111111110000001100001100001100000011111111";
       RAM(8)<= "111111111111101111101101111101111111111111";
       RAM(9)<= "111111111111101111101101111101111111111111";
       RAM(10)<= "111111111111101100000000001101111111111111";
       RAM(11)<= "111111111111101101111111101101111111111111";
       RAM(12)<= "111111111111101101111111101101111111111111";
       RAM(13)<= "111111110000000001111111100000000011111111";
       RAM(14)<= "111111111111101101111111101101111111111111";
       RAM(15)<=     "111111111111101101111111101101111111111111";
       RAM(16)<=     "111111111111101100000000001101111111111111";
       RAM(17)<= "111111111111101101111111101101111111111111";
       RAM(18)<= "111111111111101101111111101101111111111111";
       RAM(19)<= "111111110000000000001100000000000011111111";
       RAM(20)<= "111111110111101111101101111101111011111111";
       RAM(21)<=     "111111110111101111101101111101111011111111";
       RAM(22)<= "111111110001100000000000000000011000111111111";
       RAM(23)<=     "111111111101101101111111101101101111111111";
       RAM(24)<= "111111111101101101111111101101101111111111";
       RAM(25)<= "111111110000001100001100001100000011111111";
       RAM(26)<= "111111110111111111101101111111111011111111";
       RAM(27)<= "111111110111111111101101111111111011111111";
       RAM(28)<=     "111111110000000000000000000000000011111111";
       RAM(29)<=     "111111111111111111111111111111111111111111";
       RAM(30)<=     "111111111111111111111111111111111111111111";

       maze_sprite_ram(0) <=
"111111111111111100000000000000001111111111111111000000000000000011111
1111111111100000000000000000111111111111111100000000000000001111111111
1111100000000000000001111111111111111000000000000000011111111111111110
000000000000000111111111111111100000000000000000";
       maze_sprite_ram(1) <=
"000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000111111000000000011111110000
0000000000001100000000000000110000000000000011000000000000001100000000
0000001100000000000000110000000000000011000000000";
```

```
        maze_sprite_ram(2) <=
"000000011000000000000001100000000000000110000000000000011000000000000
0011000000000000000110000000000000011000000011111110000000001111110000
0000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000";
       maze_sprite_ram(3) <=
"000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000011111100000000011
1111100000001100000000000000110000000000000011000000000000001100000000
000000110000000000000011000000000000001100000000";
        maze_sprite_ram(4) <=
"000000011000000000000001100000000000000110000000000000011000000000000
0011000000000000000110000000000000011000000000000000111111100000000001
1111100000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000";
        maze_sprite_ram(5) <=
"000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000111111111111111111111111111
1111100000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000";
        maze_sprite_ram(6) <=
"000000011000000000000001100000000000000110000000000000011000000000000
0011000000000000000110000000000000011000000000000000110000000000000001100
0000000000001100000000000000110000000000000011000000000000001100000000
000000110000000000000011000000000000001100000000";
        maze_sprite_ram(7) <=
"000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000";

        dot_sprite_ram(0)(0) <= "0000000000000000";
        dot_sprite_ram(0)(1) <= "0000000000000000";
        dot_sprite_ram(0)(2) <= "0000000000000000";
        dot_sprite_ram(0)(3) <= "0000000000000000";
        dot_sprite_ram(0)(4) <= "0000000000000000";
        dot_sprite_ram(0)(5) <= "0000000000000000";
        dot_sprite_ram(0)(6) <= "0000000000000000";
        dot_sprite_ram(0)(7) <= "0000000110000000";
        dot_sprite_ram(0)(8) <= "0000000110000000";
        dot_sprite_ram(0)(9) <= "0000000000000000";
        dot_sprite_ram(0)(10) <="0000000000000000";
        dot_sprite_ram(0)(11) <="0000000000000000";
        dot_sprite_ram(0)(12) <="0000000000000000";
        dot_sprite_ram(0)(13) <="0000000000000000";
        dot_sprite_ram(0)(14) <="0000000000000000";
        dot_sprite_ram(0)(15) <="0000000000000000";

        dot_sprite_ram(1)(0) <= "0000000110000000";
        dot_sprite_ram(1)(1) <= "0000011111100000";
        dot_sprite_ram(1)(2) <= "0000111111110000";
        dot_sprite_ram(1)(3) <= "0001111111111000";
        dot_sprite_ram(1)(4) <= "0001111111111000";
        dot_sprite_ram(1)(5) <= "0011111111111100";
        dot_sprite_ram(1)(6) <= "0011111111111100";
        dot_sprite_ram(1)(7) <= "0111111111111110";
        dot_sprite_ram(1)(8) <= "0111111111111110";
        dot_sprite_ram(1)(9) <= "0011111111111100";
        dot_sprite_ram(1)(10) <="0011111111111100";
        dot_sprite_ram(1)(11) <="0011111111111100";
        dot_sprite_ram(1)(12) <="0001111111111000";
        dot_sprite_ram(1)(13) <="0000111111110000";
        dot_sprite_ram(1)(14) <="0000011111100000";
        dot_sprite_ram(1)(15) <="0000000110000000";

        num_sprite_ram(0)(0) <= "0000000000000000";
        num_sprite_ram(0)(1) <= "0000011111100000";
        num_sprite_ram(0)(2) <= "0000011111100000";
        num_sprite_ram(0)(3) <= "0001100001111000";
        num_sprite_ram(0)(4) <= "0001100001111000";
        num_sprite_ram(0)(5) <= "0111100000011110";
        num_sprite_ram(0)(6) <= "0111100000011110";
        num_sprite_ram(0)(7) <= "0111100000011110";
        num_sprite_ram(0)(8) <= "0111100000011110";
        num_sprite_ram(0)(9) <= "0111100000011110";
        num_sprite_ram(0)(10) <="0111100000011110";
        num_sprite_ram(0)(11) <="0001111000011000";
        num_sprite_ram(0)(12) <="0001111000011000";
        num_sprite_ram(0)(13) <="0000011111100000";
        num_sprite_ram(0)(14) <="0000011111100000";
        num_sprite_ram(0)(15) <="0000000000000000";


        num_sprite_ram(1)(0) <=  "0000000000000000";
        num_sprite_ram(1)(1) <=  "0000001111000000";
        num_sprite_ram(1)(2) <=  "0000001111000000";
        num_sprite_ram(1)(3) <=  "0000111111000000";
        num_sprite_ram(1)(4) <=  "0000111111000000";
        num_sprite_ram(1)(5) <=  "0000001111000000";
        num_sprite_ram(1)(6) <=  "0000001111000000";
        num_sprite_ram(1)(7) <=  "0000001111000000";
        num_sprite_ram(1)(8) <=  "0000001111000000";
        num_sprite_ram(1)(9) <=  "0000001111000000";
        num_sprite_ram(1)(10) <= "0000001111000000";
        num_sprite_ram(1)(11) <= "0000001111000000";
        num_sprite_ram(1)(12) <= "0000001111000000";
        num_sprite_ram(1)(13) <= "0011111111111100";
        num_sprite_ram(1)(14) <= "0011111111111100";
        num_sprite_ram(1)(15) <= "0000000000000000";

        num_sprite_ram(2)(0) <=  "0000000000000000";
        num_sprite_ram(2)(1) <=  "0001111111111000";
        num_sprite_ram(2)(2) <=  "0001111111111000";
        num_sprite_ram(2)(3) <=  "0111100000011110";
        num_sprite_ram(2)(4) <=  "0111100000011110";
        num_sprite_ram(2)(5) <=  "0000000001111110";
        num_sprite_ram(2)(6) <=  "0000000001111110";
        num_sprite_ram(2)(7) <=  "0000011111111000";
        num_sprite_ram(2)(8) <=  "0000011111111000";
        num_sprite_ram(2)(9) <=  "0001111111100000";
        num_sprite_ram(2)(10) <= "0001111111100000";
        num_sprite_ram(2)(11) <= "0111111000000000";
        num_sprite_ram(2)(12) <= "0111111000000000";
        num_sprite_ram(2)(13) <= "0111111111111110";
        num_sprite_ram(2)(14) <= "0111111111111110";
        num_sprite_ram(2)(15) <= "0000000000000000";

        num_sprite_ram(3)(0) <= "0000000000000000";
        num_sprite_ram(3)(1) <= "0000011111111000";
        num_sprite_ram(3)(2) <= "0000011111111000";
        num_sprite_ram(3)(3) <= "0001100001111000";
        num_sprite_ram(3)(4) <= "0001100001111000";
        num_sprite_ram(3)(5) <= "0000000001111000";
        num_sprite_ram(3)(6) <= "0000000001111000";
        num_sprite_ram(3)(7) <= "0000000111100000";
        num_sprite_ram(3)(8) <= "0000000111100000";
        num_sprite_ram(3)(9) <= "0000000001111110";
        num_sprite_ram(3)(10) <="0000000001111110";
        num_sprite_ram(3)(11) <="0110000000011110";
        num_sprite_ram(3)(12) <="0110000000011110";
        num_sprite_ram(3)(13) <="0001111111111000";
        num_sprite_ram(3)(14) <="0001111111111000";
        num_sprite_ram(3)(15) <="0000000000000000";

        num_sprite_ram(4)(0) <= "0000000000000000";
        num_sprite_ram(4)(1) <= "0000000111111000";
        num_sprite_ram(4)(2) <= "0000000111111000";
        num_sprite_ram(4)(3) <= "0000011111111000";
        num_sprite_ram(4)(4) <= "0000011111111000";
        num_sprite_ram(4)(5) <= "0001111001111000";
        num_sprite_ram(4)(6) <= "0001111001111000";
        num_sprite_ram(4)(7) <= "0111100001111000";
        num_sprite_ram(4)(8) <= "0111100001111000";
        num_sprite_ram(4)(9) <= "0111111111111110";
        num_sprite_ram(4)(10) <="0000000001111000";
        num_sprite_ram(4)(11) <="0000000001111000";
        num_sprite_ram(4)(12) <="0000000001111000";
        num_sprite_ram(4)(13) <="0000000001111000";
        num_sprite_ram(4)(14) <="0000000001111000";
        num_sprite_ram(4)(15) <="0000000000000000";

        num_sprite_ram(5)(0) <= "0000000000000000";
        num_sprite_ram(5)(1) <= "0111111111111000";
        num_sprite_ram(5)(2) <= "0111111111111000";
        num_sprite_ram(5)(3) <= "0111100000000000";
        num_sprite_ram(5)(4) <= "0111100000000000";
        num_sprite_ram(5)(5) <= "0111111111111000";
        num_sprite_ram(5)(6) <= "0111111111111000";
        num_sprite_ram(5)(7) <= "0000000000011110";
        num_sprite_ram(5)(8) <= "0000000000011110";
        num_sprite_ram(5)(9) <= "0000000000011110";
        num_sprite_ram(5)(10) <="0000000000011110";
        num_sprite_ram(5)(11) <="0111100000011110";
        num_sprite_ram(5)(12) <="0111100000011110";
        num_sprite_ram(5)(13) <="0001111111111000";
        num_sprite_ram(5)(14) <="0001111111111000";
        num_sprite_ram(5)(15) <="0000000000000000";
        num_sprite_ram(6)(0) <= "0000000000000000";
        num_sprite_ram(6)(1) <= "0001111111111000";
        num_sprite_ram(6)(2) <= "0001111111111000";
        num_sprite_ram(6)(3) <= "0111100000000000";
        num_sprite_ram(6)(4) <= "0111100000000000";
        num_sprite_ram(6)(5) <= "0111111111111000";
        num_sprite_ram(6)(6) <= "0111111111111000";
        num_sprite_ram(6)(7) <= "0111100000011110";
        num_sprite_ram(6)(8) <= "0111100000011110";
        num_sprite_ram(6)(9) <= "0111100000011110";
        num_sprite_ram(6)(10) <="0111100000011110";
        num_sprite_ram(6)(11) <="0111100000011110";
        num_sprite_ram(6)(12) <="0111100000011110";
        num_sprite_ram(6)(13) <="0001111111111000";
        num_sprite_ram(6)(14) <="0001111111111000";
        num_sprite_ram(6)(15) <="0000000000000000";

        num_sprite_ram(7)(0) <= "0000000000000000";
        num_sprite_ram(7)(1) <= "0111111111111110";
        num_sprite_ram(7)(2) <= "0111111111111110";
        num_sprite_ram(7)(3) <= "0111100000011110";
        num_sprite_ram(7)(4) <= "0111100000011110";
        num_sprite_ram(7)(5) <= "0000000001111000";
        num_sprite_ram(7)(6) <= "0000000001111000";
        num_sprite_ram(7)(7) <= "0000000111100000";
        num_sprite_ram(7)(8) <= "0000000111100000";
        num_sprite_ram(7)(9) <= "0000011110000000";
        num_sprite_ram(7)(10) <="0000011110000000";
        num_sprite_ram(7)(11) <="0000011110000000";
        num_sprite_ram(7)(12) <="0000011110000000";
        num_sprite_ram(7)(13) <="0000011110000000";
        num_sprite_ram(7)(14) <="0000011110000000";
        num_sprite_ram(7)(15) <="0000000000000000";

        num_sprite_ram(8)(0) <= "0000000000000000";
        num_sprite_ram(8)(1) <= "0001111111100000";
        num_sprite_ram(8)(2) <= "0001111111100000";
        num_sprite_ram(8)(3) <= "0111100000011000";
        num_sprite_ram(8)(4) <= "0111100000011000";
        num_sprite_ram(8)(5) <= "0111111000011000";
        num_sprite_ram(8)(6) <= "0111111000011000";
        num_sprite_ram(8)(7) <= "0001111111100000";
        num_sprite_ram(8)(8) <= "0001111111100000";
        num_sprite_ram(8)(9) <= "0110000111111110";
        num_sprite_ram(8)(10) <="0110000111111110";
        num_sprite_ram(8)(11) <="0110000000011110";
        num_sprite_ram(8)(12) <="0110000000011110";
        num_sprite_ram(8)(13) <="0001111111111000";
        num_sprite_ram(8)(14) <="0001111111111000";
        num_sprite_ram(8)(15) <="0000000000000000";

        num_sprite_ram(9)(0) <= "0000000000000000";
        num_sprite_ram(9)(1) <= "0001111111111000";
        num_sprite_ram(9)(2) <= "0001111111111000";
        num_sprite_ram(9)(3) <= "0111100000011110";
        num_sprite_ram(9)(4) <= "0111100000011110";
        num_sprite_ram(9)(5) <= "0111100000011110";
        num_sprite_ram(9)(6) <= "0111100000011110";
        num_sprite_ram(9)(7) <= "0001111111111110";
        num_sprite_ram(9)(8) <= "0001111111111110";
        num_sprite_ram(9)(9) <= "0000000000011110";
        num_sprite_ram(9)(10) <="0000000000011110";
        num_sprite_ram(9)(11) <="0000000001111000";
        num_sprite_ram(9)(12) <="0000000001111000";
        num_sprite_ram(9)(13) <="0001111111100000";
        num_sprite_ram(9)(14) <="0001111111100000";
        num_sprite_ram(9)(15) <="0000000000000000";

        num_sprite_ram(10)(0) <= "0000000000000000";
        num_sprite_ram(10)(1) <= "0000000000000000";
        num_sprite_ram(10)(2) <= "0000000000000000";
        num_sprite_ram(10)(3) <= "0000000000000000";
        num_sprite_ram(10)(4) <= "0000000000000000";
        num_sprite_ram(10)(5) <= "0000000000000000";
        num_sprite_ram(10)(6) <= "0000000000000000";
        num_sprite_ram(10)(7) <= "0000000000000000";
        num_sprite_ram(10)(8) <= "0000000000000000";
        num_sprite_ram(10)(9) <= "0000000000000000";
        num_sprite_ram(10)(10) <="0000000000000000";
        num_sprite_ram(10)(11) <="0000000000000000";
        num_sprite_ram(10)(12) <="0000000000000000";
        num_sprite_ram(10)(13) <="0000000000000000";
        num_sprite_ram(10)(14) <="0000000000000000";
        num_sprite_ram(10)(15) <="0000000000000000";
```

```
        color_ram_1(0) <= "000000000000000000000000000000";
        color_ram_1(1) <= "010010000000111001000010111000";
        color_ram_1(2) <= "010110000000111100000010011100";
        color_ram_1(3) <= "111101110011101001001100100000";
        color_ram_1(4) <= "110000100010110111001001110100";
        color_ram_1(5) <= "110101000011001000001010110000";
        color_ram_1(6) <= "111010110011011110001011111000";
        color_ram_1(7) <= "111000110011010110001011011100";
        color_ram_1(8) <= "011011010011000001001110100100";
        color_ram_1(9) <= "111111111111111111111111111111";
        color_ram_1(10) <="111101110011100000001100100000";
        color_ram_1(11) <="111011100011011001001100000100";
        color_ram_1(12) <="111010000011001110001011101100";

        color_ram_2(0) <= "000000000000000000000000000000";
        color_ram_2(1) <= "110000000011000000001100000000";
        color_ram_2(2) <= "100000000010000000001000000000";
        color_ram_2(3) <= "111010110011011110001011111000";
        color_ram_2(4) <= "011011010011000001001110100100";
        color_ram_2(5) <= "111101110011101001001100100000";
        color_ram_2(6) <= "111111111111111111111000000000";
        color_ram_2(7) <= "111111111111111111111111111111";

        color_ram_dot(0) <= "000000000000000000000000000000";
        color_ram_dot(1) <= "111111111111111111111000000000";
        color_ram_dot(2) <= "111111111100000000000000000000";
        color_ram_dot(3) <= "000000000100000000000000000000";
        color_ram_dot(4) <= "111111111110000000000100000000";
        color_ram_dot(5) <= "100000000000000000000000000000";
        color_ram_dot(6) <= "111111111111111111111111111111";

        player_sprite_ram(0)(0)
<="0000";player_sprite_ram(0)(1)<="0000";player_sprite_ram(0)(2)<="000
0";player_sprite_ram(0)(3)<="0001";player_sprite_ram(0)(4)<="0010";pla
yer_sprite_ram(0)(5)<="0010";player_sprite_ram(0)(6)<="0001";player_sp
rite_ram(0)(7)<="0001";player_sprite_ram(0)(8)<="0010";player_sprite_r
am(0)(9)<="0010";player_sprite_ram(0)(10)<="0010";player_sprite_ram(0)
(11)<="0000";player_sprite_ram(0)(12)<="0000";player_sprite_ram(0)(13)
<="0000";player_sprite_ram(0)(14)<="0000";player_sprite_ram(0)(15)<="0
000";
```

```
        player_sprite_ram(1)(0)
<="0000";player_sprite_ram(1)(1)<="0000";player_sprite_ram(1)(2)<="000
0";player_sprite_ram(1)(3)<="0010";player_sprite_ram(1)(4)<="0001";pla
yer_sprite_ram(1)(5)<="0010";player_sprite_ram(1)(6)<="0010";player_sp
rite_ram(1)(7)<="0010";player_sprite_ram(1)(8)<="0001";player_sprite_r
am(1)(9)<="0001";player_sprite_ram(1)(10)<="0001";player_sprite_ram(1)
(11)<="0010";player_sprite_ram(1)(12)<="0000";player_sprite_ram(1)(13)
<="0000";player_sprite_ram(1)(14)<="0000";player_sprite_ram(1)(15)<="0
000";
        player_sprite_ram(2)(0)
<="0000";player_sprite_ram(2)(1)<="0000";player_sprite_ram(2)(2)<="001
0";player_sprite_ram(2)(3)<="0010";player_sprite_ram(2)(4)<="0010";pla
yer_sprite_ram(2)(5)<="0001";player_sprite_ram(2)(6)<="0010";player_sp
rite_ram(2)(7)<="0010";player_sprite_ram(2)(8)<="0010";player_sprite_r
am(2)(9)<="0010";player_sprite_ram(2)(10)<="0010";player_sprite_ram(2)
(11)<="0001";player_sprite_ram(2)(12)<="0010";player_sprite_ram(2)(13)
<="0000";player_sprite_ram(2)(14)<="0000";player_sprite_ram(2)(15)<="0
000";
        player_sprite_ram(3)(0)
<="0000";player_sprite_ram(3)(1)<="0000";player_sprite_ram(3)(2)<="000
1";player_sprite_ram(3)(3)<="0010";player_sprite_ram(3)(4)<="0010";pla
yer_sprite_ram(3)(5)<="0011";player_sprite_ram(3)(6)<="0011";player_sp
rite_ram(3)(7)<="0011";player_sprite_ram(3)(8)<="0011";player_sprite_r
am(3)(9)<="0011";player_sprite_ram(3)(10)<="0011";player_sprite_ram(3)
(11)<="0010";player_sprite_ram(3)(12)<="0010";player_sprite_ram(3)(13)
<="0000";player_sprite_ram(3)(14)<="0000";player_sprite_ram(3)(15)<="0
000";
        player_sprite_ram(4)(0)
<="0000";player_sprite_ram(4)(1)<="0000";player_sprite_ram(4)(2)<="001
0";player_sprite_ram(4)(3)<="0001";player_sprite_ram(4)(4)<="0011";pla
yer_sprite_ram(4)(5)<="0011";player_sprite_ram(4)(6)<="0011";player_sp
rite_ram(4)(7)<="0011";player_sprite_ram(4)(8)<="0011";player_sprite_r
am(4)(9)<="0011";player_sprite_ram(4)(10)<="0011";player_sprite_ram(4)
(11)<="0011";player_sprite_ram(4)(12)<="0001";player_sprite_ram(4)(13)
<="0000";player_sprite_ram(4)(14)<="0000";player_sprite_ram(4)(15)<="0
000";
        player_sprite_ram(5)(0)
<="0000";player_sprite_ram(5)(1)<="0000";player_sprite_ram(5)(2)<="001
0";player_sprite_ram(5)(3)<="0010";player_sprite_ram(5)(4)<="0011";pla
yer_sprite_ram(5)(5)<="0011";player_sprite_ram(5)(6)<="0011";player_sp
rite_ram(5)(7)<="0011";player_sprite_ram(5)(8)<="0011";player_sprite_r
am(5)(9)<="0011";player_sprite_ram(5)(10)<="0011";player_sprite_ram(5)
```

```
(11)<="0011";player_sprite_ram(5)(12)<="0010";player_sprite_ram(5)(13)
<="0000";player_sprite_ram(5)(14)<="0000";player_sprite_ram(5)(15)<="0
000";
        player_sprite_ram(6)(0)
<="0000";player_sprite_ram(6)(1)<="0000";player_sprite_ram(6)(2)<="001
0";player_sprite_ram(6)(3)<="0100";player_sprite_ram(6)(4)<="0100";pla
yer_sprite_ram(6)(5)<="0100";player_sprite_ram(6)(6)<="0100";player_sp
rite_ram(6)(7)<="0101";player_sprite_ram(6)(8)<="0110";player_sprite_r
am(6)(9)<="0110";player_sprite_ram(6)(10)<="0101";player_sprite_ram(6)
(11)<="0100";player_sprite_ram(6)(12)<="0100";player_sprite_ram(6)(13)
<="0100";player_sprite_ram(6)(14)<="0000";player_sprite_ram(6)(15)<="0
000";
        player_sprite_ram(7)(0)
<="0000";player_sprite_ram(7)(1)<="0000";player_sprite_ram(7)(2)<="001
1";player_sprite_ram(7)(3)<="0011";player_sprite_ram(7)(4)<="0011";pla
yer_sprite_ram(7)(5)<="0100";player_sprite_ram(7)(6)<="1001";player_sp
rite_ram(7)(7)<="1000";player_sprite_ram(7)(8)<="0100";player_sprite_r
am(7)(9)<="0100";player_sprite_ram(7)(10)<="1001";player_sprite_ram(7)
(11)<="1000";player_sprite_ram(7)(12)<="0100";player_sprite_ram(7)(13)
<="0000";player_sprite_ram(7)(14)<="0000";player_sprite_ram(7)(15)<="0
000";
        player_sprite_ram(8)(0)
<="0000";player_sprite_ram(8)(1)<="0000";player_sprite_ram(8)(2)<="001
1";player_sprite_ram(8)(3)<="0011";player_sprite_ram(8)(4)<="0011";pla
yer_sprite_ram(8)(5)<="0111";player_sprite_ram(8)(6)<="0100";player_sp
rite_ram(8)(7)<="0101";player_sprite_ram(8)(8)<="0110";player_sprite_r
am(8)(9)<="0110";player_sprite_ram(8)(10)<="0101";player_sprite_ram(8)
(11)<="0100";player_sprite_ram(8)(12)<="0111";player_sprite_ram(8)(13)
<="0000";player_sprite_ram(8)(14)<="0000";player_sprite_ram(8)(15)<="0
000";
        player_sprite_ram(9)(0)
<="0000";player_sprite_ram(9)(1)<="0000";player_sprite_ram(9)(2)<="001
1";player_sprite_ram(9)(3)<="0011";player_sprite_ram(9)(4)<="0011";pla
yer_sprite_ram(9)(5)<="0011";player_sprite_ram(9)(6)<="0011";player_sp
rite_ram(9)(7)<="0011";player_sprite_ram(9)(8)<="0011";player_sprite_r
am(9)(9)<="0011";player_sprite_ram(9)(10)<="0011";player_sprite_ram(9)
(11)<="0011";player_sprite_ram(9)(12)<="0011";player_sprite_ram(9)(13)
<="0000";player_sprite_ram(9)(14)<="0000";player_sprite_ram(9)(15)<="0
000";

player_sprite_ram(10)(0)<="0000";player_sprite_ram(10)(1)<="0000";play
er_sprite_ram(10)(2)<="0000";player_sprite_ram(10)(3)<="0011";player_s
```

```
prite_ram(10)(4)<="0011";player_sprite_ram(10)(5)<="0011";player_sprit
e_ram(10)(6)<="0011";player_sprite_ram(10)(7)<="0011";player_sprite_ra
m(10)(8)<="0110";player_sprite_ram(10)(9)<="0110";player_sprite_ram(10
)(10)<="0011";player_sprite_ram(10)(11)<="0011";player_sprite_ram(10)(
12)<="0000";player_sprite_ram(10)(13)<="0000";player_sprite_ram(10)(14
)<="0000";player_sprite_ram(10)(15)<="0000";

player_sprite_ram(11)(0)<="0000";player_sprite_ram(11)(1)<="0000";play
er_sprite_ram(11)(2)<="0000";player_sprite_ram(11)(3)<="0011";player_s
prite_ram(11)(4)<="0011";player_sprite_ram(11)(5)<="0011";player_sprit
e_ram(11)(6)<="0011";player_sprite_ram(11)(7)<="0011";player_sprite_ra
m(11)(8)<="0011";player_sprite_ram(11)(9)<="0011";player_sprite_ram(11
)(10)<="0011";player_sprite_ram(11)(11)<="0011";player_sprite_ram(11)(
12)<="0000";player_sprite_ram(11)(13)<="0000";player_sprite_ram(11)(14
)<="0000";player_sprite_ram(11)(15)<="0000";

player_sprite_ram(12)(0)<="0000";player_sprite_ram(12)(1)<="0000";play
er_sprite_ram(12)(2)<="0000";player_sprite_ram(12)(3)<="0011";player_s
prite_ram(12)(4)<="0011";player_sprite_ram(12)(5)<="0011";player_sprit
e_ram(12)(6)<="1010";player_sprite_ram(12)(7)<="0011";player_sprite_ra
m(12)(8)<="0011";player_sprite_ram(12)(9)<="0011";player_sprite_ram(12
)(10)<="0011";player_sprite_ram(12)(11)<="0011";player_sprite_ram(12)(
12)<="0000";player_sprite_ram(12)(13)<="0000";player_sprite_ram(12)(14
)<="0000";player_sprite_ram(12)(15)<="0000";

player_sprite_ram(13)(0)<="0000";player_sprite_ram(13)(1)<="0000";play
er_sprite_ram(13)(2)<="0000";player_sprite_ram(13)(3)<="0000";player_s
prite_ram(13)(4)<="0011";player_sprite_ram(13)(5)<="0011";player_sprit
e_ram(13)(6)<="0011";player_sprite_ram(13)(7)<="1011";player_sprite_ra
m(13)(8)<="1100";player_sprite_ram(13)(9)<="1100";player_sprite_ram(13
)(10)<="1011";player_sprite_ram(13)(11)<="0011";player_sprite_ram(13)(
12)<="0000";player_sprite_ram(13)(13)<="0000";player_sprite_ram(13)(14
)<="0000";player_sprite_ram(13)(15)<="0000";

player_sprite_ram(14)(0)<="0000";player_sprite_ram(14)(1)<="0000";play
er_sprite_ram(14)(2)<="0000";player_sprite_ram(14)(3)<="0000";player_s
prite_ram(14)(4)<="0000";player_sprite_ram(14)(5)<="0011";player_sprit
e_ram(14)(6)<="0011";player_sprite_ram(14)(7)<="0011";player_sprite_ra
m(14)(8)<="0011";player_sprite_ram(14)(9)<="0011";player_sprite_ram(14
)(10)<="0011";player_sprite_ram(14)(11)<="0011";player_sprite_ram(14)(
12)<="0000";player_sprite_ram(14)(13)<="0000";player_sprite_ram(14)(14
)<="0000";player_sprite_ram(14)(15)<="0000";
```

```
player_sprite_ram(15)(0)<="0000";player_sprite_ram(15)(1)<="0000";play
er_sprite_ram(15)(2)<="0000";player_sprite_ram(15)(3)<="0000";player_s
prite_ram(15)(4)<="0000";player_sprite_ram(15)(5)<="0000";player_sprit
e_ram(15)(6)<="0011";player_sprite_ram(15)(7)<="0011";player_sprite_ra
m(15)(8)<="0011";player_sprite_ram(15)(9)<="0011";player_sprite_ram(15
)(10)<="0011";player_sprite_ram(15)(11)<="0000";player_sprite_ram(15)(
12)<="0000";player_sprite_ram(15)(13)<="0000";player_sprite_ram(15)(14
)<="0000";player_sprite_ram(15)(15)<="0000";


player2_sprite_ram(0)(0)<="0000";player2_sprite_ram(0)(1)<="0000";play
er2_sprite_ram(0)(2)<="0000";player2_sprite_ram(0)(3)<="0000";player2_
sprite_ram(0)(4)<="0001";player2_sprite_ram(0)(5)<="0111";player2_spri
te_ram(0)(6)<="0111";player2_sprite_ram(0)(7)<="0111";player2_sprite_r
am(0)(8)<="0111";player2_sprite_ram(0)(9)<="0001";player2_sprite_ram(0
)(10)<="0111";player2_sprite_ram(0)(11)<="0111";player2_sprite_ram(0)(
12)<="0111";player2_sprite_ram(0)(13)<="0111";player2_sprite_ram(0)(14
)<="0000";player2_sprite_ram(0)(15)<="0000";

player2_sprite_ram(1)(0)<="0000";player2_sprite_ram(1)(1)<="0000";play
er2_sprite_ram(1)(2)<="0000";player2_sprite_ram(1)(3)<="0111";player2_
sprite_ram(1)(4)<="0111";player2_sprite_ram(1)(5)<="0000";player2_spri
te_ram(1)(6)<="0000";player2_sprite_ram(1)(7)<="0000";player2_sprite_r
am(1)(8)<="0000";player2_sprite_ram(1)(9)<="0000";player2_sprite_ram(1
)(10)<="0111";player2_sprite_ram(1)(11)<="0111";player2_sprite_ram(1)(
12)<="0001";player2_sprite_ram(1)(13)<="0111";player2_sprite_ram(1)(14
)<="0111";player2_sprite_ram(1)(15)<="0000";

player2_sprite_ram(2)(0)<="0000";player2_sprite_ram(2)(1)<="0000";play
er2_sprite_ram(2)(2)<="0111";player2_sprite_ram(2)(3)<="0001";player2_
sprite_ram(2)(4)<="0000";player2_sprite_ram(2)(5)<="0000";player2_spri
te_ram(2)(6)<="0110";player2_sprite_ram(2)(7)<="0000";player2_sprite_r
am(2)(8)<="0110";player2_sprite_ram(2)(9)<="0000";player2_sprite_ram(2
)(10)<="0000";player2_sprite_ram(2)(11)<="0111";player2_sprite_ram(2)(
12)<="0111";player2_sprite_ram(2)(13)<="0001";player2_sprite_ram(2)(14
)<="0111";player2_sprite_ram(2)(15)<="0000";

player2_sprite_ram(3)(0)<="0000";player2_sprite_ram(3)(1)<="0000";play
er2_sprite_ram(3)(2)<="0111";player2_sprite_ram(3)(3)<="0111";player2_
sprite_ram(3)(4)<="0000";player2_sprite_ram(3)(5)<="0110";player2_spri
te_ram(3)(6)<="0000";player2_sprite_ram(3)(7)<="0110";player2_sprite_r
```

```
am(3)(8)<="0000";player2_sprite_ram(3)(9)<="0110";player2_sprite_ram(3
)(10)<="0000";player2_sprite_ram(3)(11)<="0000";player2_sprite_ram(3)(
12)<="0111";player2_sprite_ram(3)(13)<="0111";player2_sprite_ram(3)(14
)<="0111";player2_sprite_ram(3)(15)<="0000";

player2_sprite_ram(4)(0)<="0000";player2_sprite_ram(4)(1)<="0111";play
er2_sprite_ram(4)(2)<="0111";player2_sprite_ram(4)(3)<="0001";player2_
sprite_ram(4)(4)<="0000";player2_sprite_ram(4)(5)<="0110";player2_spri
te_ram(4)(6)<="0000";player2_sprite_ram(4)(7)<="0000";player2_sprite_r
am(4)(8)<="0000";player2_sprite_ram(4)(9)<="0110";player2_sprite_ram(4
)(10)<="0000";player2_sprite_ram(4)(11)<="0000";player2_sprite_ram(4)(
12)<="0111";player2_sprite_ram(4)(13)<="0111";player2_sprite_ram(4)(14
)<="0111";player2_sprite_ram(4)(15)<="0001";

player2_sprite_ram(5)(0)<="0000";player2_sprite_ram(5)(1)<="0001";play
er2_sprite_ram(5)(2)<="0111";player2_sprite_ram(5)(3)<="0000";player2_
sprite_ram(5)(4)<="0000";player2_sprite_ram(5)(5)<="0000";player2_spri
te_ram(5)(6)<="0000";player2_sprite_ram(5)(7)<="0000";player2_sprite_r
am(5)(8)<="0000";player2_sprite_ram(5)(9)<="0000";player2_sprite_ram(5
)(10)<="0000";player2_sprite_ram(5)(11)<="0000";player2_sprite_ram(5)(
12)<="0111";player2_sprite_ram(5)(13)<="0001";player2_sprite_ram(5)(14
)<="0111";player2_sprite_ram(5)(15)<="0111";

player2_sprite_ram(6)(0)<="0000";player2_sprite_ram(6)(1)<="0111";play
er2_sprite_ram(6)(2)<="0111";player2_sprite_ram(6)(3)<="0000";player2_
sprite_ram(6)(4)<="0000";player2_sprite_ram(6)(5)<="0101";player2_spri
te_ram(6)(6)<="0101";player2_sprite_ram(6)(7)<="0101";player2_sprite_r
am(6)(8)<="0101";player2_sprite_ram(6)(9)<="0101";player2_sprite_ram(6
)(10)<="0000";player2_sprite_ram(6)(11)<="0000";player2_sprite_ram(6)(
12)<="0111";player2_sprite_ram(6)(13)<="0111";player2_sprite_ram(6)(14
)<="0111";player2_sprite_ram(6)(15)<="0111";

player2_sprite_ram(7)(0)<="0111";player2_sprite_ram(7)(1)<="0111";play
er2_sprite_ram(7)(2)<="0111";player2_sprite_ram(7)(3)<="0000";player2_
sprite_ram(7)(4)<="0101";player2_sprite_ram(7)(5)<="0101";player2_spri
te_ram(7)(6)<="0101";player2_sprite_ram(7)(7)<="0101";player2_sprite_r
am(7)(8)<="0101";player2_sprite_ram(7)(9)<="0101";player2_sprite_ram(7
)(10)<="0101";player2_sprite_ram(7)(11)<="0000";player2_sprite_ram(7)(
12)<="0111";player2_sprite_ram(7)(13)<="0111";player2_sprite_ram(7)(14
)<="0111";player2_sprite_ram(7)(15)<="0001";

player2_sprite_ram(8)(0)<="0001";player2_sprite_ram(8)(1)<="0111";play
```

```
er2_sprite_ram(8)(2)<="0001";player2_sprite_ram(8)(3)<="0101";player2_
sprite_ram(8)(4)<="0100";player2_sprite_ram(8)(5)<="0111";player2_spri
te_ram(8)(6)<="0101";player2_sprite_ram(8)(7)<="0101";player2_sprite_r
am(8)(8)<="0100";player2_sprite_ram(8)(9)<="0111";player2_sprite_ram(8
)(10)<="0101";player2_sprite_ram(8)(11)<="0101";player2_sprite_ram(8)(
12)<="0111";player2_sprite_ram(8)(13)<="0001";player2_sprite_ram(8)(14
)<="0111";player2_sprite_ram(8)(15)<="0111";

player2_sprite_ram(9)(0)<="0000";player2_sprite_ram(9)(1)<="0111";play
er2_sprite_ram(9)(2)<="0111";player2_sprite_ram(9)(3)<="0101";player2_
sprite_ram(9)(4)<="0101";player2_sprite_ram(9)(5)<="0101";player2_spri
te_ram(9)(6)<="0101";player2_sprite_ram(9)(7)<="0101";player2_sprite_r
am(9)(8)<="0101";player2_sprite_ram(9)(9)<="0101";player2_sprite_ram(9
)(10)<="0101";player2_sprite_ram(9)(11)<="0101";player2_sprite_ram(9)(
12)<="0001";player2_sprite_ram(9)(13)<="0111";player2_sprite_ram(9)(14
)<="0111";player2_sprite_ram(9)(15)<="0111";

player2_sprite_ram(10)(0)<="0000";player2_sprite_ram(10)(1)<="0001";pl
ayer2_sprite_ram(10)(2)<="0111";player2_sprite_ram(10)(3)<="0111";play
er2_sprite_ram(10)(4)<="0101";player2_sprite_ram(10)(5)<="0101";player
2_sprite_ram(10)(6)<="0101";player2_sprite_ram(10)(7)<="0101";player2_
sprite_ram(10)(8)<="0101";player2_sprite_ram(10)(9)<="0101";player2_sp
rite_ram(10)(10)<="0101";player2_sprite_ram(10)(11)<="0101";player2_sp
rite_ram(10)(12)<="0111";player2_sprite_ram(10)(13)<="0111";player2_sp
rite_ram(10)(14)<="0111";player2_sprite_ram(10)(15)<="0001";

player2_sprite_ram(11)(0)<="0000";player2_sprite_ram(11)(1)<="0000";pl
ayer2_sprite_ram(11)(2)<="0111";player2_sprite_ram(11)(3)<="0111";play
er2_sprite_ram(11)(4)<="0101";player2_sprite_ram(11)(5)<="0101";player
2_sprite_ram(11)(6)<="0011";player2_sprite_ram(11)(7)<="0011";player2_
sprite_ram(11)(8)<="0101";player2_sprite_ram(11)(9)<="0101";player2_sp
rite_ram(11)(10)<="0101";player2_sprite_ram(11)(11)<="0101";player2_sp
rite_ram(11)(12)<="0111";player2_sprite_ram(11)(13)<="0001";player2_sp
rite_ram(11)(14)<="0111";player2_sprite_ram(11)(15)<="0111";

player2_sprite_ram(12)(0)<="0000";player2_sprite_ram(12)(1)<="0000";pl
ayer2_sprite_ram(12)(2)<="0001";player2_sprite_ram(12)(3)<="0111";play
er2_sprite_ram(12)(4)<="0101";player2_sprite_ram(12)(5)<="0101";player
2_sprite_ram(12)(6)<="0101";player2_sprite_ram(12)(7)<="0101";player2_
sprite_ram(12)(8)<="0101";player2_sprite_ram(12)(9)<="0101";player2_sp
rite_ram(12)(10)<="0101";player2_sprite_ram(12)(11)<="0001";player2_sp
rite_ram(12)(12)<="0111";player2_sprite_ram(12)(13)<="0111";player2_sp
rite_ram(12)(14)<="0111";player2_sprite_ram(12)(15)<="0000";

player2_sprite_ram(13)(0)<="0000";player2_sprite_ram(13)(1)<="0000";pl
ayer2_sprite_ram(13)(2)<="0000";player2_sprite_ram(13)(3)<="0111";play
er2_sprite_ram(13)(4)<="0111";player2_sprite_ram(13)(5)<="0010";player
2_sprite_ram(13)(6)<="0010";player2_sprite_ram(13)(7)<="0010";player2_
sprite_ram(13)(8)<="0010";player2_sprite_ram(13)(9)<="0010";player2_sp
rite_ram(13)(10)<="0010";player2_sprite_ram(13)(11)<="0111";player2_sp
rite_ram(13)(12)<="0111";player2_sprite_ram(13)(13)<="0111";player2_sp
rite_ram(13)(14)<="0111";player2_sprite_ram(13)(15)<="0000";

player2_sprite_ram(14)(0)<="0000";player2_sprite_ram(14)(1)<="0000";pl
ayer2_sprite_ram(14)(2)<="0000";player2_sprite_ram(14)(3)<="0001";play
er2_sprite_ram(14)(4)<="0111";player2_sprite_ram(14)(5)<="0111";player
2_sprite_ram(14)(6)<="0010";player2_sprite_ram(14)(7)<="0010";player2_
sprite_ram(14)(8)<="0010";player2_sprite_ram(14)(9)<="0010";player2_sp
rite_ram(14)(10)<="0111";player2_sprite_ram(14)(11)<="0111";player2_sp
rite_ram(14)(12)<="0111";player2_sprite_ram(14)(13)<="0001";player2_sp
rite_ram(14)(14)<="0000";player2_sprite_ram(14)(15)<="0000";

player2_sprite_ram(15)(0)<="0000";player2_sprite_ram(15)(1)<="0000";pl
ayer2_sprite_ram(15)(2)<="0000";player2_sprite_ram(15)(3)<="0000";play
er2_sprite_ram(15)(4)<="0111";player2_sprite_ram(15)(5)<="0001";player
2_sprite_ram(15)(6)<="0111";player2_sprite_ram(15)(7)<="0111";player2_
sprite_ram(15)(8)<="0111";player2_sprite_ram(15)(9)<="0111";player2_sp
rite_ram(15)(10)<="0111";player2_sprite_ram(15)(11)<="0001";player2_sp
rite_ram(15)(12)<="0111";player2_sprite_ram(15)(13)<="0000";player2_sp
rite_ram(15)(14)<="0000";player2_sprite_ram(15)(15)<="0000";


bigdot_sprite_ram(0)(0)<="0000";bigdot_sprite_ram(0)(1)<="0000";bigdot
_sprite_ram(0)(2)<="0000";bigdot_sprite_ram(0)(3)<="0000";bigdot_sprit
e_ram(0)(4)<="0000";bigdot_sprite_ram(0)(5)<="0000";bigdot_sprite_ram(
0)(6)<="0000";bigdot_sprite_ram(0)(7)<="0000";bigdot_sprite_ram(0)(8)<
="0000";bigdot_sprite_ram(0)(9)<="0000";bigdot_sprite_ram(0)(10)<="000
0";bigdot_sprite_ram(0)(11)<="0000";bigdot_sprite_ram(0)(12)<="0000";b
igdot_sprite_ram(0)(13)<="0000";bigdot_sprite_ram(0)(14)<="0000";bigdo
t_sprite_ram(0)(15)<="0000";

bigdot_sprite_ram(1)(0)<="0000";bigdot_sprite_ram(1)(1)<="0000";bigdot
_sprite_ram(1)(2)<="0000";bigdot_sprite_ram(1)(3)<="0000";bigdot_sprit
```

```
e_ram(1)(4)<="0000";bigdot_sprite_ram(1)(5)<="0000";bigdot_sprite_ram(
1)(6)<="0000";bigdot_sprite_ram(1)(7)<="0000";bigdot_sprite_ram(1)(8)<
="0000";bigdot_sprite_ram(1)(9)<="0000";bigdot_sprite_ram(1)(10)<="000
0";bigdot_sprite_ram(1)(11)<="0000";bigdot_sprite_ram(1)(12)<="0000";b
igdot_sprite_ram(1)(13)<="0000";bigdot_sprite_ram(1)(14)<="0000";bigdo
t_sprite_ram(1)(15)<="0000";

bigdot_sprite_ram(2)(0)<="0000";bigdot_sprite_ram(2)(1)<="0000";bigdot
_sprite_ram(2)(2)<="0000";bigdot_sprite_ram(2)(3)<="0000";bigdot_sprit
e_ram(2)(4)<="0000";bigdot_sprite_ram(2)(5)<="0001";bigdot_sprite_ram(
2)(6)<="0001";bigdot_sprite_ram(2)(7)<="0001";bigdot_sprite_ram(2)(8)<
="0001";bigdot_sprite_ram(2)(9)<="0001";bigdot_sprite_ram(2)(10)<="000
1";bigdot_sprite_ram(2)(11)<="0000";bigdot_sprite_ram(2)(12)<="0000";b
igdot_sprite_ram(2)(13)<="0000";bigdot_sprite_ram(2)(14)<="0000";bigdo
t_sprite_ram(2)(15)<="0000";

bigdot_sprite_ram(3)(0)<="0000";bigdot_sprite_ram(3)(1)<="0000";bigdot
_sprite_ram(3)(2)<="0000";bigdot_sprite_ram(3)(3)<="0001";bigdot_sprit
e_ram(3)(4)<="0001";bigdot_sprite_ram(3)(5)<="0001";bigdot_sprite_ram(
3)(6)<="0001";bigdot_sprite_ram(3)(7)<="0110";bigdot_sprite_ram(3)(8)<
="0001";bigdot_sprite_ram(3)(9)<="0001";bigdot_sprite_ram(3)(10)<="000
1";bigdot_sprite_ram(3)(11)<="0001";bigdot_sprite_ram(3)(12)<="0001";b
igdot_sprite_ram(3)(13)<="0000";bigdot_sprite_ram(3)(14)<="0000";bigdo
t_sprite_ram(3)(15)<="0000";

bigdot_sprite_ram(4)(0)<="0000";bigdot_sprite_ram(4)(1)<="0000";bigdot
_sprite_ram(4)(2)<="0001";bigdot_sprite_ram(4)(3)<="0001";bigdot_sprit
e_ram(4)(4)<="0110";bigdot_sprite_ram(4)(5)<="0001";bigdot_sprite_ram(
4)(6)<="0001";bigdot_sprite_ram(4)(7)<="0001";bigdot_sprite_ram(4)(8)<
="0001";bigdot_sprite_ram(4)(9)<="0001";bigdot_sprite_ram(4)(10)<="000
1";bigdot_sprite_ram(4)(11)<="0110";bigdot_sprite_ram(4)(12)<="0001";b
igdot_sprite_ram(4)(13)<="0001";bigdot_sprite_ram(4)(14)<="0000";bigdo
t_sprite_ram(4)(15)<="0000";

bigdot_sprite_ram(5)(0)<="0000";bigdot_sprite_ram(5)(1)<="0001";bigdot
_sprite_ram(5)(2)<="0001";bigdot_sprite_ram(5)(3)<="0001";bigdot_sprit
e_ram(5)(4)<="0001";bigdot_sprite_ram(5)(5)<="0001";bigdot_sprite_ram(
5)(6)<="0001";bigdot_sprite_ram(5)(7)<="0001";bigdot_sprite_ram(5)(8)<
="0110";bigdot_sprite_ram(5)(9)<="0001";bigdot_sprite_ram(5)(10)<="000
1";bigdot_sprite_ram(5)(11)<="0001";bigdot_sprite_ram(5)(12)<="0001";b
igdot_sprite_ram(5)(13)<="0001";bigdot_sprite_ram(5)(14)<="0001";bigdo
t_sprite_ram(5)(15)<="0000";
```

```
bigdot_sprite_ram(6)(0)<="0000";bigdot_sprite_ram(6)(1)<="0001";bigdot
_sprite_ram(6)(2)<="0001";bigdot_sprite_ram(6)(3)<="0001";bigdot_sprit
e_ram(6)(4)<="0001";bigdot_sprite_ram(6)(5)<="0001";bigdot_sprite_ram(
6)(6)<="0001";bigdot_sprite_ram(6)(7)<="0001";bigdot_sprite_ram(6)(8)<
="0001";bigdot_sprite_ram(6)(9)<="0001";bigdot_sprite_ram(6)(10)<="000
1";bigdot_sprite_ram(6)(11)<="0001";bigdot_sprite_ram(6)(12)<="0001";b
igdot_sprite_ram(6)(13)<="0001";bigdot_sprite_ram(6)(14)<="0001";bigdo
t_sprite_ram(6)(15)<="0000";

bigdot_sprite_ram(7)(0)<="0000";bigdot_sprite_ram(7)(1)<="0000";bigdot
_sprite_ram(7)(2)<="0010";bigdot_sprite_ram(7)(3)<="0010";bigdot_sprit
e_ram(7)(4)<="0010";bigdot_sprite_ram(7)(5)<="0010";bigdot_sprite_ram(
7)(6)<="0010";bigdot_sprite_ram(7)(7)<="0010";bigdot_sprite_ram(7)(8)<
="0010";bigdot_sprite_ram(7)(9)<="0010";bigdot_sprite_ram(7)(10)<="001
0";bigdot_sprite_ram(7)(11)<="0010";bigdot_sprite_ram(7)(12)<="0010";b
igdot_sprite_ram(7)(13)<="0010";bigdot_sprite_ram(7)(14)<="0000";bigdo
t_sprite_ram(7)(15)<="0000";

bigdot_sprite_ram(8)(0)<="0000";bigdot_sprite_ram(8)(1)<="0011";bigdot
_sprite_ram(8)(2)<="0011";bigdot_sprite_ram(8)(3)<="0011";bigdot_sprit
e_ram(8)(4)<="0011";bigdot_sprite_ram(8)(5)<="0011";bigdot_sprite_ram(
8)(6)<="0011";bigdot_sprite_ram(8)(7)<="0011";bigdot_sprite_ram(8)(8)<
="0011";bigdot_sprite_ram(8)(9)<="0011";bigdot_sprite_ram(8)(10)<="001
1";bigdot_sprite_ram(8)(11)<="0011";bigdot_sprite_ram(8)(12)<="0011";b
igdot_sprite_ram(8)(13)<="0011";bigdot_sprite_ram(8)(14)<="0011";bigdo
t_sprite_ram(8)(15)<="0000";

bigdot_sprite_ram(9)(0)<="0101";bigdot_sprite_ram(9)(1)<="0101";bigdot
_sprite_ram(9)(2)<="0101";bigdot_sprite_ram(9)(3)<="0100";bigdot_sprit
e_ram(9)(4)<="0100";bigdot_sprite_ram(9)(5)<="0100";bigdot_sprite_ram(
9)(6)<="0100";bigdot_sprite_ram(9)(7)<="0100";bigdot_sprite_ram(9)(8)<
="0101";bigdot_sprite_ram(9)(9)<="0101";bigdot_sprite_ram(9)(10)<="010
1";bigdot_sprite_ram(9)(11)<="0101";bigdot_sprite_ram(9)(12)<="0101";b
igdot_sprite_ram(9)(13)<="0101";bigdot_sprite_ram(9)(14)<="0101";bigdo
t_sprite_ram(9)(15)<="0101";

bigdot_sprite_ram(10)(0)<="0101";bigdot_sprite_ram(10)(1)<="0101";bigd
ot_sprite_ram(10)(2)<="0101";bigdot_sprite_ram(10)(3)<="0101";bigdot_s
prite_ram(10)(4)<="0100";bigdot_sprite_ram(10)(5)<="0100";bigdot_sprit
e_ram(10)(6)<="0100";bigdot_sprite_ram(10)(7)<="0101";bigdot_sprite_ra
m(10)(8)<="0101";bigdot_sprite_ram(10)(9)<="0101";bigdot_sprite_ram(10
)(10)<="0101";bigdot_sprite_ram(10)(11)<="0101";bigdot_sprite_ram(10)(
12)<="0101";bigdot_sprite_ram(10)(13)<="0101";bigdot_sprite_ram(10)(14
)<="0101";bigdot_sprite_ram(10)(15)<="0101";

bigdot_sprite_ram(11)(0)<="0000";bigdot_sprite_ram(11)(1)<="0000";bigd
ot_sprite_ram(11)(2)<="0001";bigdot_sprite_ram(11)(3)<="0001";bigdot_s
prite_ram(11)(4)<="0001";bigdot_sprite_ram(11)(5)<="0100";bigdot_sprit
e_ram(11)(6)<="0001";bigdot_sprite_ram(11)(7)<="0001";bigdot_sprite_ra
m(11)(8)<="0001";bigdot_sprite_ram(11)(9)<="0001";bigdot_sprite_ram(11
)(10)<="0001";bigdot_sprite_ram(11)(11)<="0001";bigdot_sprite_ram(11)(
12)<="0001";bigdot_sprite_ram(11)(13)<="0001";bigdot_sprite_ram(11)(14
)<="0000";bigdot_sprite_ram(11)(15)<="0000";

bigdot_sprite_ram(12)(0)<="0000";bigdot_sprite_ram(12)(1)<="0000";bigd
ot_sprite_ram(12)(2)<="0000";bigdot_sprite_ram(12)(3)<="0001";bigdot_s
prite_ram(12)(4)<="0001";bigdot_sprite_ram(12)(5)<="0001";bigdot_sprit
e_ram(12)(6)<="0001";bigdot_sprite_ram(12)(7)<="0001";bigdot_sprite_ra
m(12)(8)<="0001";bigdot_sprite_ram(12)(9)<="0001";bigdot_sprite_ram(12
)(10)<="0001";bigdot_sprite_ram(12)(11)<="0001";bigdot_sprite_ram(12)(
12)<="0001";bigdot_sprite_ram(12)(13)<="0000";bigdot_sprite_ram(12)(14
)<="0000";bigdot_sprite_ram(12)(15)<="0000";

bigdot_sprite_ram(13)(0)<="0000";bigdot_sprite_ram(13)(1)<="0000";bigd
ot_sprite_ram(13)(2)<="0000";bigdot_sprite_ram(13)(3)<="0000";bigdot_s
prite_ram(13)(4)<="0001";bigdot_sprite_ram(13)(5)<="0001";bigdot_sprit
e_ram(13)(6)<="0001";bigdot_sprite_ram(13)(7)<="0001";bigdot_sprite_ra
m(13)(8)<="0001";bigdot_sprite_ram(13)(9)<="0001";bigdot_sprite_ram(13
)(10)<="0001";bigdot_sprite_ram(13)(11)<="0001";bigdot_sprite_ram(13)(
12)<="0000";bigdot_sprite_ram(13)(13)<="0000";bigdot_sprite_ram(13)(14
)<="0000";bigdot_sprite_ram(13)(15)<="0000";

bigdot_sprite_ram(14)(0)<="0000";bigdot_sprite_ram(14)(1)<="0000";bigd
ot_sprite_ram(14)(2)<="0000";bigdot_sprite_ram(14)(3)<="0000";bigdot_s
prite_ram(14)(4)<="0000";bigdot_sprite_ram(14)(5)<="0000";bigdot_sprit
e_ram(14)(6)<="0000";bigdot_sprite_ram(14)(7)<="0000";bigdot_sprite_ra
m(14)(8)<="0000";bigdot_sprite_ram(14)(9)<="0000";bigdot_sprite_ram(14
)(10)<="0000";bigdot_sprite_ram(14)(11)<="0000";bigdot_sprite_ram(14)(
12)<="0000";bigdot_sprite_ram(14)(13)<="0000";bigdot_sprite_ram(14)(14
)<="0000";bigdot_sprite_ram(14)(15)<="0000";

bigdot_sprite_ram(15)(0)<="0000";bigdot_sprite_ram(15)(1)<="0000";bigd
ot_sprite_ram(15)(2)<="0000";bigdot_sprite_ram(15)(3)<="0000";bigdot_s
```

```
prite_ram(15)(4)<="0000";bigdot_sprite_ram(15)(5)<="0000";bigdot_sprit
e_ram(15)(6)<="0000";bigdot_sprite_ram(15)(7)<="0000";bigdot_sprite_ra
m(15)(8)<="0000";bigdot_sprite_ram(15)(9)<="0000";bigdot_sprite_ram(15
)(10)<="0000";bigdot_sprite_ram(15)(11)<="0000";bigdot_sprite_ram(15)(
12)<="0000";bigdot_sprite_ram(15)(13)<="0000";bigdot_sprite_ram(15)(14
)<="0000";bigdot_sprite_ram(15)(15)<="0000";


       if avs_s1_reset_n = '0' then
        avs_s1_readdata <= (others => '0');
        display_address <= (others => '0');
--        counter <= (others => '0');
--        counter_delay <= (others => '1');
      else
        if avs_s1_chipselect = '1' then
          if avs_s1_read = '1' then
                avs_s1_readdata <= score_ram(0)(0) &
"0000000000000000000000000000";
          elsif avs_s1_write = '1' then
                if avs_s1_address = 0 then
                     player_1_xpos <= avs_s1_writedata(9 downto 0);
                     player_1_ypos <= avs_s1_writedata(25 downto 16);
                elsif avs_s1_address = 1 then
                     player_2_xpos <= avs_s1_writedata(9 downto 0);
                     player_2_ypos <= avs_s1_writedata(25 downto 16);
                elsif avs_s1_address = 2 then
                     score_ram(0)(2) <= avs_s1_writedata(3 downto 0);
                     score_ram(0)(1) <= avs_s1_writedata(7 downto 4);
                     score_ram(0)(0) <= avs_s1_writedata(11 downto
8);
                elsif avs_s1_address = 3 then
                     score_ram(1)(2) <= avs_s1_writedata(3 downto 0);
                     score_ram(1)(1) <= avs_s1_writedata(7 downto 4);
                     score_ram(1)(0) <= avs_s1_writedata(11 downto
8);
                else
                     show_dot_ram(conv_integer(avs_s1_address)-4) <=
avs_s1_writedata(27 downto 0);
                end if;
          end if;
        end if;
      end if;
```

```
    end if;
  end process;

      process (clk)
      begin
            if clk'event and clk = '1' then
--                maze <= maze_line_from_RAM(conv_integer(maze_x));
                  maze <=
RAM(conv_integer(maze_y))(conv_integer(maze_x));
            end if;
      end process;

  HCounter : process (clk_25, reset_n)
  begin
    if reset_n = '1' then
      Hcount <= (others => '0');
    elsif clk_25'event and clk_25 = '1' then
      if EndOfLine = '1' then
        Hcount <= (others => '0');
      else
        Hcount <= Hcount + 1;
      end if;
    end if;
  end process HCounter;

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

  VCounter: process (clk_25, reset_n)
  begin
    if reset_n = '1' then
      Vcount <= (others => '0');
    elsif clk_25'event and clk_25 = '1' then
      if EndOfLine = '1' then
        if EndOfField = '1' then
          Vcount <= (others => '0');
        else
          Vcount <= Vcount + 1;
           visible_ypos <= Vcount - VSYNC - VBACK_PORCH;
             sprite_y <= visible_ypos(3 downto 0);
             if maze_y /= visible_ypos(8 downto 4) then
                   maze_y <= visible_ypos(8 downto 4);
                   maze_line_from_RAM <= RAM(conv_integer(maze_y));
          end if;
        end if;
      end if;
    end if;
  end process VCounter;

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

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

  HSyncGen : process (clk_25, reset_n)
  begin
    if reset_n = '1' then
      vga_hsync <= '1';
    elsif clk_25'event and clk_25 = '1' then
      if EndOfLine = '1' then
        vga_hsync <= '1';
      elsif Hcount = HSYNC - 1 then
        vga_hsync <= '0';
      end if;
    end if;
  end process HSyncGen;

  HBlankGen : process (clk_25, reset_n)
  begin
    if reset_n = '1' then
      vga_hblank <= '1';
    elsif clk_25'event and clk_25 = '1' then
      if Hcount = HSYNC + HBACK_PORCH then
        vga_hblank <= '0';
      elsif Hcount = HSYNC + HBACK_PORCH + HACTIVE then
        vga_hblank <= '1';
      end if;
    end if;
  end process HBlankGen;

  VSyncGen : process (clk_25, reset_n)
  begin
    if reset_n = '1' then
      vga_vsync <= '1';
    elsif clk_25'event and clk_25 = '1' then
      if EndOfLine ='1' then
```

```
        if EndOfField = '1' then
          vga_vsync <= '1';
        elsif Vcount = VSYNC - 1 then
          vga_vsync <= '0';
        end if;
      end if;
    end if;
  end process VSyncGen;

  VBlankGen : process (clk_25, reset_n)
  begin
    if reset_n = '1' then
      vga_vblank <= '1';
    elsif clk_25'event and clk_25 = '1' then
      if EndOfLine = '1' then
        if Vcount = VSYNC + VBACK_PORCH - 1 then
          vga_vblank <= '0';
        elsif Vcount = VSYNC + VBACK_PORCH + VACTIVE - 1 then
          vga_vblank <= '1';
        end if;
      end if;
    end if;
  end process VBlankGen;

  -- Registered video signals going to the video DAC

  VideoOut: process (clk_25, reset_n)
  begin
    if reset_n = '1' then
      VGA_R <= "0000000000";
      VGA_G <= "0000000000";
      VGA_B <= "0000000000";
    elsif clk_25'event and clk_25 = '1' then     --it is here that we
determine which sprites take precedence over each other
                                                                    --
this is really important stuff!
       if draw_player_1 = '1' then
           VGA_R <=
color_ram_1(conv_integer(player_sprite_ram(conv_integer(visible_ypos)-
conv_integer(player_1_ypos))
```

```
                      (conv_integer(visible_xpos)-
conv_integer(player_1_xpos))

))(29 downto 20);
          VGA_G <=
color_ram_1(conv_integer(player_sprite_ram(conv_integer(visible_ypos)-
conv_integer(player_1_ypos))(conv_integer(visible_xpos)-
conv_integer(player_1_xpos))))(19 downto 10);
          VGA_B <=
color_ram_1(conv_integer(player_sprite_ram(conv_integer(visible_ypos)-
conv_integer(player_1_ypos))(conv_integer(visible_xpos)-
conv_integer(player_1_xpos))))(9 downto 0);
       elsif draw_player_2 = '1' then
          VGA_R <=
color_ram_2(conv_integer(player2_sprite_ram(conv_integer(visible_ypos)
-conv_integer(player_2_ypos))

                      (conv_integer(visible_xpos)-
conv_integer(player_2_xpos))

))(29 downto 20);
          VGA_G <=
color_ram_2(conv_integer(player2_sprite_ram(conv_integer(visible_ypos)
-conv_integer(player_2_ypos))(conv_integer(visible_xpos)-
conv_integer(player_2_xpos))))(19 downto 10);
          VGA_B <=
color_ram_2(conv_integer(player2_sprite_ram(conv_integer(visible_ypos)
-conv_integer(player_2_ypos))(conv_integer(visible_xpos)-
conv_integer(player_2_xpos))))(9 downto 0);
      elsif maze = '1' then
          if print_char = '1' then --if we should be printing some
number or text instead of the maze:
               VGA_R <= (others =>
current_char_sprite_from_ram(conv_integer(sprite_y))(conv_integer(spri
te_x)));
               VGA_G <= (others =>
current_char_sprite_from_ram(conv_integer(sprite_y))(conv_integer(spri
te_x)));
               VGA_B <= (others =>
current_char_sprite_from_ram(conv_integer(sprite_y))(conv_integer(spri
te_x)));
```

```
           else
              VGA_R <= "0000000000";
              VGA_G <= "0000000000";
              VGA_B <= (others =>
current_maze_sprite_from_ram(conv_integer(sprite_x) +
conv_integer(sprite_y&"0000")));
           end if;
      elsif vga_hblank = '0' and vga_vblank ='0' then
           if show_dot_ram(conv_integer(maze_y)-
1)(conv_integer(maze_x)-6) = '1' then
                if show_big_dot = '1' then
                     VGA_R <=
color_ram_dot(conv_integer(bigdot_sprite_ram(conv_integer(sprite_y))(c
onv_integer(sprite_x))))(29 downto 20);
                     VGA_G <=
color_ram_dot(conv_integer(bigdot_sprite_ram(conv_integer(sprite_y))(c
onv_integer(sprite_x))))(19 downto 10);
                     VGA_B <=
color_ram_dot(conv_integer(bigdot_sprite_ram(conv_integer(sprite_y))(c
onv_integer(sprite_x))))(9 downto 0);
                elsif show_big_dot = '0' then
                  VGA_R <= (others =>
current_dot_sprite_from_ram(conv_integer(sprite_y))(conv_integer(sprit
e_x)));
                  VGA_G <= (others =>
current_dot_sprite_from_ram(conv_integer(sprite_y))(conv_integer(sprit
e_x)));
                  VGA_B <= "0000000000";
                end if;
           else
              VGA_R <= "0000000000";
              VGA_G <= "0000000000";
          VGA_B <= "0000000000";
           end if;
      else
        VGA_R <= "0000000000";
        VGA_G <= "0000000000";
        VGA_B <= "0000000000";
      end if;
    end if;
  end process VideoOut;
  VGA_CLK <= clk_25;
  VGA_HS <= not vga_hsync;
  VGA_VS <= not vga_vsync;
  VGA_SYNC <= '0';
  VGA_BLANK <= not (vga_hsync or vga_vsync);
end rtl;
```

```
--tupacvhdl.vhd

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity tupacvhdl is

  port (
    signal CPU_RESET, CLOCK_50 : in std_logic;                -- 50
MHz clock
    VGA_CLK,                                            -- Clock
    VGA_HS,                                             -- H_SYNC
    VGA_VS,                                             -- V_SYNC
    VGA_BLANK,                                          -- BLANK
    VGA_SYNC : out std_logic;                           -- SYNC
    VGA_R,                                              -- Red[9:0]
    VGA_G,                                              -- Green[9:0]
    VGA_B : out std_logic_vector(9 downto 0);           -- Blue[9:0]


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

    SW : in std_logic_vector(17 downto 0);         -- DPDT switches
```

```
     GPIO_0 : inout std_logic_vector(35 downto 0)
     );

end tupacvhdl;

architecture rtl of tupacvhdl is

  component tupacsystem is
  port (
-- 1) global signals:
                 signal clk : IN STD_LOGIC;
                 signal reset_n : IN STD_LOGIC;

              -- the_de2_sram_controller_0
                 signal SRAM_ADDR_from_the_de2_sram_controller_0 : OUT
STD_LOGIC_VECTOR (17 DOWNTO 0);
                 signal SRAM_CE_N_from_the_de2_sram_controller_0 : OUT
STD_LOGIC;
                 signal SRAM_DQ_to_and_from_the_de2_sram_controller_0
: INOUT STD_LOGIC_VECTOR (15 DOWNTO 0);
                 signal SRAM_LB_N_from_the_de2_sram_controller_0 : OUT
STD_LOGIC;
                 signal SRAM_OE_N_from_the_de2_sram_controller_0 : OUT
STD_LOGIC;
                 signal SRAM_UB_N_from_the_de2_sram_controller_0 : OUT
STD_LOGIC;
                 signal SRAM_WE_N_from_the_de2_sram_controller_0 : OUT
STD_LOGIC;

              -- the_nes_controller
                 signal data1_to_the_nes_controller : IN STD_LOGIC;
                 signal data2_to_the_nes_controller : IN STD_LOGIC;
                 signal latch1_from_the_nes_controller : OUT
STD_LOGIC;
                 signal latch2_from_the_nes_controller : OUT
STD_LOGIC;
                 signal pulse1_from_the_nes_controller : OUT
STD_LOGIC;
                 signal pulse2_from_the_nes_controller : OUT
STD_LOGIC;

              -- the_video_controller
```

```
                 signal CLK_25_to_the_video_controller : IN STD_LOGIC;
                 signal VGA_BLANK_from_the_video_controller : OUT
STD_LOGIC;
                 signal VGA_B_from_the_video_controller : OUT
STD_LOGIC_VECTOR (9 DOWNTO 0);
                 signal VGA_CLK_from_the_video_controller : OUT
STD_LOGIC;
                 signal VGA_G_from_the_video_controller : OUT
STD_LOGIC_VECTOR (9 DOWNTO 0);
                 signal VGA_HS_from_the_video_controller : OUT
STD_LOGIC;
                 signal VGA_R_from_the_video_controller : OUT
STD_LOGIC_VECTOR (9 DOWNTO 0);
                 signal VGA_SYNC_from_the_video_controller : OUT
STD_LOGIC;
                 signal VGA_VS_from_the_video_controller : OUT
STD_LOGIC;
                 signal reset_n_to_the_video_controller : IN STD_LOGIC
  );
  end component;

  signal counter : std_logic_vector(15 downto 0);
  signal reset_n : std_logic;
  signal clk25 : std_logic := '0';

begin

  process (CLOCK_50)
  begin
    if CLOCK_50'event and CLOCK_50 = '1' then
      clk25 <= not clk25;
    end if;
  end process;

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

  tupac : tupacsystem port map (
    clk                         => CLOCK_50,
    reset_n                     => reset_n,
    SRAM_ADDR_from_the_de2_sram_controller_0      => SRAM_ADDR,
    SRAM_CE_N_from_the_de2_sram_controller_0      => SRAM_CE_N,
    SRAM_DQ_to_and_from_the_de2_sram_controller_0 => SRAM_DQ,
    SRAM_LB_N_from_the_de2_sram_controller_0      => SRAM_LB_N,
    SRAM_OE_N_from_the_de2_sram_controller_0      => SRAM_OE_N,
    SRAM_UB_N_from_the_de2_sram_controller_0      => SRAM_UB_N,
    SRAM_WE_N_from_the_de2_sram_controller_0      => SRAM_WE_N,

     data1_to_the_nes_controller => GPIO_0(5),
    data2_to_the_nes_controller => GPIO_0(15),
    latch1_from_the_nes_controller => GPIO_0(1),
    latch2_from_the_nes_controller => GPIO_0(11),
    pulse1_from_the_nes_controller => GPIO_0(3),
    pulse2_from_the_nes_controller => GPIO_0(13),

     CLK_25_to_the_video_controller                          =>
clk25,
     VGA_BLANK_from_the_video_controller                     =>
VGA_BLANK,
    VGA_B_from_the_video_controller                          =>
VGA_B,
    VGA_CLK_from_the_video_controller                   => VGA_CLK,
    VGA_G_from_the_video_controller                          =>
VGA_G,
    VGA_HS_from_the_video_controller                         =>
VGA_HS,
    VGA_R_from_the_video_controller                          =>
VGA_R,
    VGA_SYNC_from_the_video_controller                       =>
VGA_SYNC,
     VGA_VS_from_the_video_controller                             =>
VGA_VS,
     reset_n_to_the_video_controller                        => '0'
  );
  GPIO_0(0) <= CLOCK_50;
```

```
end rtl;
```