Jumpers

Bernardo de Almeida Abreu, Henrique Pizzol Grando, Tomas Mantelato, Lucas Ikenaga Barros

***Abstract*—A design project of a game named "Jumpers" implemented on the FPGA SoCKit development board is here proposed. The purpose of this project is to develop a game called Jumpers using synthesizable System Verilog and C language on a FPGA board. The user input is obtained through a triple axis accelerometer. In the game, two players compete against each other playing simultaneously. The hardware developed in System Verilog will generate the image for the VGA and exchange information with the C program. The C program calculates the characters and platforms positions for sprites to be printed and apply the game rules.**

I. Introduction

The Jumpers game consists of a screen shared by two characters playing against each other. Both characters must jump through the platforms that are randomly distributed across the screen, in order to go to a higher position. If one character is touched by the bottom border of the screen, he then dies and loses the game. When the game ends the total height reached by the winner is showed in the screen. A image of the implemented game is depicted in the appendix.

II. Design Methodology

A block diagram of the project proposed is depicted in Figure 1.

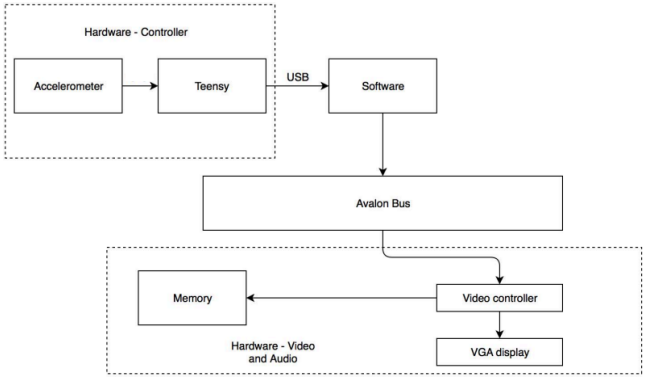

Fig. 1. Diagram of the system

A. Hardware

1) Hardware Peripheral - The ADXL335 (3 axis accelerometer): a ADXL335 accelerometer is used as the first stage of the system. The accelerometer works as the controller for the game. This piece of hardware controls the position of the player on the frame. The movement takes in consideration the position of the hand, trying to be as smoothly as possible and only one of the axis is used in this project, controlling only the horizontal displacement of the player. The accelerometer is plugged to the Teensy 2.0 development board, having a USB connection with the FPGA board. This allow us to work with the accelerometer as if it was a simple keyboard. Figure 2 below illustrates the triple axis accelerometer.

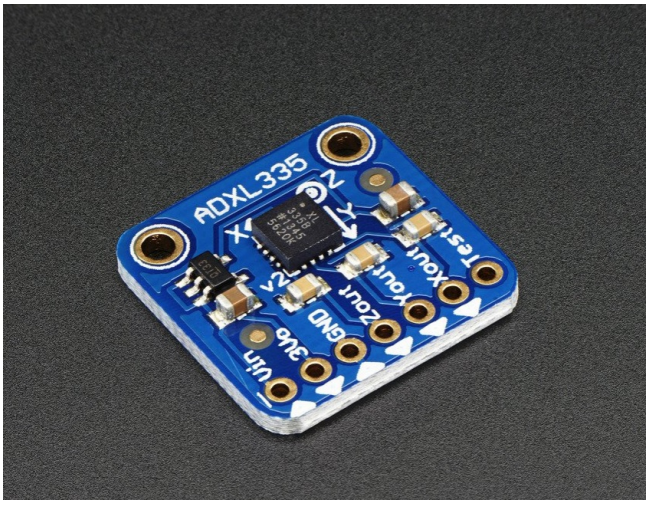

Fig. 2. ADXL335

2) Hardware Peripheral - The Teensy 2.0 (ATmega32u4 USB development board): the Teensy 2.0 is an Arduino compatible board that is equipped with an ATmega32u4 processor and uses USB to communicate with other devices.

This board reads the data from the accelerometer and sends it to the FPGA via USB. Figure 3 illustrates the Teensy pinout.

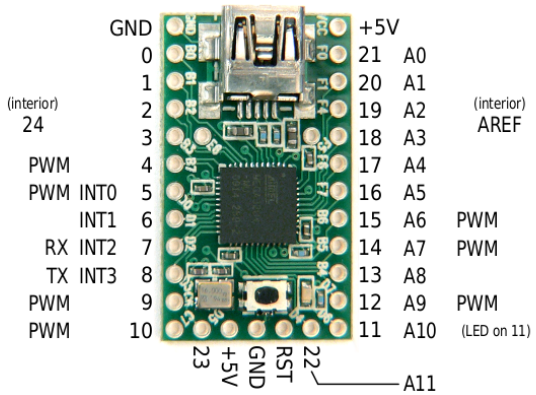

Fig. 3. Teensy 2.0

3) Hardware FPGA - SoCKit Board: The hardware in the FPGA was developed using the hardware description language System Verilog. It is also responsible for generating the background and printing all video content to the VGA display.

The VGA_LED_Emulator module is capable of showing on the screen up to 9 ground sprites and 8 general sprites. The printing is divided in four layers with different priorities. The background layer has the smallest priority, followed by the ground sprites, the regular sprites and the icons, which have the biggest priority. The information for each sprite is stored in registers. For the ground sprites there are registers for storing, for each sprite, the x and y position and the number of times a sprite is going to be sequentially printed. For the regular sprites, besides the registers that also exist for the ground sprites, there are registers to store which sprite to print and if the sprite is going to be mirrored or not.

There are control modules that, for the two sprite layers and for the icons, decide which of the possible sprites is going to be the shown on each pixel. These modules select the memory position to fetch the code of the pixel. This code goes through a color code decoder that outputs the RGB values for the pixel. The pixel for the position is printed according to the priority of the layers.

The sprites are stored in two different memories. There is a memory that store the color code for each pixel and a memory that store if a pixel is active, which means that it is to be printed.

The background is stored in memory and, for each pixel, two pixels are printed. All rows are also printed twice. This is done in order to double the size of the image.

- Sprites: there are seven different sprites used in the game. The sprites can be mirrored through the hardware. Two of them represent the number of life icons of both the characters, with a size of 16x16 and illustrated in Figures 4 and 5.

Fig. 4. Grandma icon

Fig. 5. Grandpa icon

Other four represent the characters in two states: either jumping or landing on the ground, with a size of 32x32. Figures 6 to 9.

Fig. 6. Grandma landed

Fig. 7. Grandpa landed

Fig. 8. Grandma jumping

Fig. 9. Grandpa jumping

The last sprite is the one used by the characters as ground and is also 32x32 (Figure 10).

Fig. 10. Ground

- Background: the background has a size of 320x240 and the hardware increases its size by a factor of 2. The color palettes of the background have only 4 colors. Figure 11 represents the background

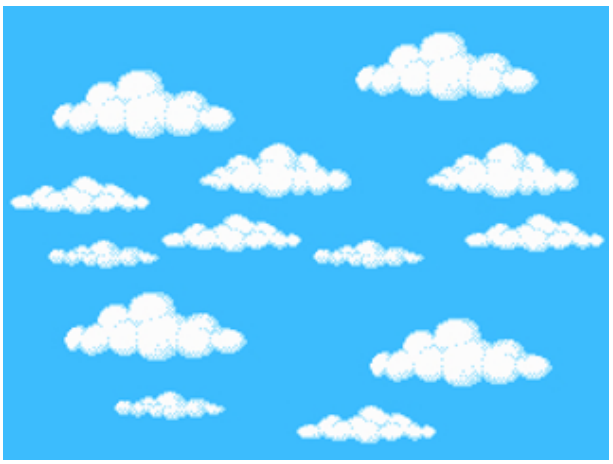

Fig. 11. Background

.

B. Software

The software part of this project is developed in C and is responsible for all the logic control in the game. We divided the software in three main different parts.

1) Main game logic : Resposible for applying all game rules: handles all collisions between characters, randomly generates grounds, decrements lives of the characters when they hit the bottom of the screen, emulates the game gravity

2) Communication with peripherals: Treats all the raw data received from the accelerometer (user input) converting voltage signals to characters and then the characters to a range of possible speeds depending on the accelerometer inclination. Connects with the accelerometer via USB through the libusb C library.

3) Communication with the hardware: The user C program exchanges information with the hardware through ioctl calls. Among the information exchanged are the sprites position and location in memory as well as the background information.

III. Tasks Division

THEgroup worked together in all tasks but we can highlight as the main tasks of each participant:

- Bernardo de Almeida Abreu: hardware on FPGA
- Henrique Pizzol Grando: software
- Lucas Ikenaga Barros: sprites and game design
- Tomas Mantelato: Peripherals

IV. Lessons Learned

THE main lessons learned during working on this projects were:

- Implementing a new peripheral can be a challenging task.
- Always have a good time management.

V. Advices for Future Projects

HERE are some advices for people working on future projects:

- Get the peripheral working, it is a nice feature to have while testing and it has real problems, dont let it be one of the last things.
- Maybe using the lab linux system works better, you wont need cross- compiling and it will definitely be easier.

References

[1] http://www.analog.com/media/en/technical-documentation/data-sheets/ADXL335.pdf
[2] https://www.pjrc.com/teensy/schematic.html
[3] http://www.instructables.com/id/USB-Game-Pad-With-Tilt-accelerometer-Mouse/?ALLSTEPS

Jumpers Source Code

May 12, 2016

Contents

1 Hardware/VGA_LED.sv

```
/*
 * Avalon memory-mapped peripheral for the VGA LED Emulator
 *
 * Stephen A. Edwards
 * Columbia University
 *
 *
 * Final Project
 *
 * Name: Bernardo de Almeida Abreu
 * UNI:  bd2440
 *
 * Name: Henrique Pizzol Grando
 * UNI:  hp2409
 *
 * Name: Lucas Ikenaga Barros
 * UNI:  li2176
 *
 * Name: Tomas Mantelato
 * UNI:  tm2779
 *
 */

module VGA_LED(input logic        clk,
               input logic        reset,
               input logic [31:0]  writedata,
               input logic        write,
               input              chipselect,
               input logic address,

               output logic [7:0] VGA_R, VGA_G, VGA_B,
               output logic       VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n,
               output logic       VGA_SYNC_n);


        logic [9:0]  posx,posy;
        logic [3:0] s_shape;
        logic s_invert;
        logic [2:0] s_count;
        logic [2:0] id;
        logic layer;
        logic [3:0] life_1, life_2;
        logic [7:0] bR,bG,bB;

   VGA_LED_Emulator led_emulator(.clk50(clk), .*);

   always_ff @(posedge clk)
     if (reset) begin
                        posx <= 10'b0010000000;
                        posy <= 10'b0010000000;
```

```
                        s_invert <= 1'd0;
                        s_count <= 3'd0;
                        id <= 3'd0;
                        s_shape <= 4'd0;
                        layer <= 1'd0;
                        life_1 <= 4'd0;
                        life_2 <= 4'd0;
                        bR <= 8'h35;
                        bG <= 8'hbc;
                        bB <= 8'hff;
      end else if (chipselect && write)
        case (address)
                        1'h0 :
                        begin
                                posx <= writedata[9:0];
                                posy <= writedata[19:10];
                                s_invert <= writedata[20];
                                s_count <= writedata[23:21];
                                id <= writedata[26:24];
                                s_shape <= writedata[30:27];
                                layer <= writedata[31];
                        end
                        1'h1 :
                        begin
                                life_1 <= writedata[3:0];
                                life_2 <= writedata[7:4];
                                bB <= writedata[15:8];
                                bG <= writedata[23:16];
                                bR <= writedata[31:24];
                        end
                endcase

endmodule
```

2 Hardware/VGA_LED_Emulator.sv

```
module VGA_LED_Emulator(
 input logic        clk50, reset,
 input logic [9:0]  posx,posy,
 input logic [3:0] s_shape,
 input logic s_invert,
 input logic [2:0] s_count,
 input logic [2:0] id,
 input logic layer,
 input logic [3:0] life_1, life_2,
 input logic [7:0] bR,bG,bB,
 output logic [7:0] VGA_R, VGA_G, VGA_B,
 output logic       VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n, VGA_SYNC_n);


        parameter N_SPRITES = 8;
        parameter N_GROUND = 9;

/*
 * 640 X 480 VGA timing for a 50 MHz clock: one pixel every other cycle
 *
 * HCOUNT 1599 0             1279       1599 0
 *             _______________              ________
 * ___________|    Video      |____________|  Video
 *
 *
 * |SYNC| BP |<-- HACTIVE -->|FP|SYNC| BP |<-- HACTIVE
 *       _______________________      _____________
 * |____|       VGA_HS          |____|
 */
   // Parameters for hcount
   parameter HACTIVE      = 11'd 1280,
             HFRONT_PORCH = 11'd 32,
             HSYNC        = 11'd 192,
             HBACK_PORCH  = 11'd 96,
             HTOTAL       = HACTIVE + HFRONT_PORCH + HSYNC + HBACK_PORCH; //
                 1600

   // Parameters for vcount
   parameter VACTIVE      = 10'd 480,
             VFRONT_PORCH = 10'd 10,
             VSYNC        = 10'd 2,
             VBACK_PORCH  = 10'd 33,
             VTOTAL       = VACTIVE + VFRONT_PORCH + VSYNC + VBACK_PORCH; //
                 525

   logic [10:0]                    hcount; // Horizontal counter
                                           // Hcount[10:1] indicates pixel
                                               column (0-639)
   logic                           endOfLine;
```

```
    always_ff @(posedge clk50 or posedge reset)
                if (reset) hcount <= 0;
                else if (endOfLine) hcount <= 0;
                else             hcount <= hcount + 11'd 1;

    assign endOfLine = hcount == HTOTAL - 1;

    // Vertical counter
    logic [9:0]                         vcount;
    logic                               endOfField;

         always_ff @(posedge clk50 or posedge reset)
                if (reset)
                        vcount <= 0;
                else if (endOfLine)
                        if (endOfField)
                                vcount <= 0;
                        else
                                vcount <= vcount + 10'd 1;



    assign endOfField = vcount == VTOTAL - 1;

    // Horizontal sync: from 0x520 to 0x5DF (0x57F)
    // 101 0010 0000 to 101 1101 1111
    assign VGA_HS = !( (hcount[10:8] == 3'b101) & !(hcount[7:5] == 3'b111));
    assign VGA_VS = !( vcount[9:1] == (VACTIVE + VFRONT_PORCH) / 2);

    assign VGA_SYNC_n = 1; // For adding sync to video signals; not used for
        VGA

    // Horizontal active: 0 to 1279     Vertical active: 0 to 479
    // 101 0000 0000  1280              01 1110 0000  480
    // 110 0011 1111  1599              10 0000 1100  524
    assign VGA_BLANK_n = !( hcount[10] & (hcount[9] | hcount[8]) ) &
                         !( vcount[9] | (vcount[8:5] == 4'b1111) );

    /* VGA_CLK is 25 MHz
     *             __    __    __
     * clk50    __|  |__|  |__|
     *
     *             _____       __
     * hcount[0]__|     |_____|
     */
    assign VGA_CLK = hcount[0]; // 25 MHz clock: pixel latched on rising edge


/*
    -------------------------------------------------------------------------------------------------
    */

        //Declarations
```

```
/*
    ************************************************************************
    */
        genvar i;

        //Sprites
        logic [2:0] count[N_SPRITES];    //count of sprite
        logic [3:0] shape[N_SPRITES];    //Shape of sprite
        logic [2:0] count_gnd[N_GROUND];         //count of sprite

        logic [7:0] Rpix[2];                                    //color red
        logic [7:0] Gpix[2];                                    //color green
        logic [7:0] Bpix[2];                                    //color blue
        logic [1:0] Apix;                                               //
            active bit
        logic inv[N_SPRITES];                   //invert sprite

        //Memory

        logic [4:0] x;
        logic [8:0] y;
        logic [4:0] gx, gy;
        logic invert;
        logic [4:0] code[2];

        //Sprite Control
        logic [9:0] cur_x, cur_y;
        logic [1:0] enable;             //enable to display sprite on screen

        //Icons
        logic [3:0] xc;                         //coordinate x
        logic [4:0] yc;                         //coordinate y
        logic enable_c;
        logic [2:0] life_p1,life_p2;
        logic icon_a;                           //output active bit
        logic [7:0] Ri,Gi,Bi;    //output colors


        //Position input data
        logic [9:0]px[N_SPRITES];
        logic [9:0]py[N_SPRITES];

        logic [9:0]px_gnd[N_GROUND];
        logic [9:0]py_gnd[N_GROUND];

/*
    ************************************************************************
    */

        //Input data
/*
    ************************************************************************
    */
```

```
    logic [3:0]id_ground;

    logic [N_SPRITES-1:0]reg_enable;
    logic [N_GROUND-1:0]reg_enb_gnd;

    assign id_ground[2:0] = id;
    assign id_ground[3] = s_shape[0];

    decoder #(N_SPRITES) (id, reg_enable);
    decoder #(N_GROUND) (id_ground, reg_enb_gnd);

    generate
            for(i = 0; i < N_GROUND; i++) begin: loop_gnd
                    register #(10) pos_x0(clk50,reset,reg_enb_gnd[i]&(~
                        layer),posx,px_gnd[i]);
                    register #(10) pos_y0(clk50,reset,reg_enb_gnd[i]&(~
                        layer),posy,py_gnd[i]);
                    register #(3) cnt0(clk50,reset,reg_enb_gnd[i]&(~layer
                        ),s_count,count_gnd[i]);
            end
    endgenerate

    generate
            for(i = 0; i < N_SPRITES; i++) begin: loop_modules
                    register #(10) pos_x1(clk50,reset,reg_enable[i]&layer
                        ,posx,px[i]);
                    register #(10) pos_y1(clk50,reset,reg_enable[i]&layer
                        ,posy,py[i]);
                    register #(3) cnt1(clk50,reset,reg_enable[i]&layer,
                        s_count,count[i]);
                    register #(1) inv1(clk50,reset,reg_enable[i]&layer,
                        s_invert,inv[i]);
                    register #(4) shp1(clk50,reset,reg_enable[i]&layer,
                        s_shape,shape[i]);
            end
    endgenerate

    assign life_p1 = life_1;
    assign life_p2 = life_2;
/*
    ******************************************************************************
    */

    //Modules instantiation
/*
    ******************************************************************************
    */
    logic [7:0] R,G,B;      //output colors

    assign cur_x = hcount[10:1];
    assign cur_y = vcount;

    groundcontrol gndctrl(cur_x, cur_y, px_gnd, py_gnd, count_gnd,
```

```
            gx, gy, enable[0]);

        spritecontrol spritectrl(cur_x, cur_y, px, py, shape, count, inv,
            x, y, enable[1], invert);

        icon_control ictrl(.*);

        icons picon(.*);

        color_gnd cmemgnd(gx,gy,code[0]);
        color_mem cmem(x,y,invert,code[1]);

        active_gnd amemgnd(gx,gy,Apix[0]);
        active_mem amem(x,y,invert,Apix[1]);

        color_code code_col_0(code[0],Rpix[0],Gpix[0],Bpix[0]);
        color_code code_col_1(code[1],Rpix[1],Gpix[1],Bpix[1]);


        background back(.*);
/*
    ************************************************************************
    */

        //Pixel Selection for printing
/*
    ************************************************************************
    */
        logic ivisible;
        logic [1:0] visible;
        logic [1:0] enb;

        assign visible = enable & Apix;
        assign ivisible = enable_c & icon_a;

        assign enb[0] = ivisible | (~visible[1])&visible[0];
        assign enb[1] = ivisible | visible[1];
/*
    ************************************************************************
    */

        //Pixel printing
/*
    ************************************************************************
    */
        always_comb begin
                case(enb)
                        2'b00: {VGA_R, VGA_G, VGA_B} = {R, G, B};
                                                    // Background
                        2'b01: {VGA_R, VGA_G, VGA_B} = {Rpix[0], Gpix[0],
                            Bpix[0]};      //Layer 0
                        2'b10: {VGA_R, VGA_G, VGA_B} = {Rpix[1], Gpix[1],
                            Bpix[1]};      //Layer 1
```

```
                        2'b11: {VGA_R, VGA_G, VGA_B} = {Ri, Gi, Bi};
                                                        //Icons layer
                endcase
        end

/*
    ************************************************************************
    */

/*
    ------------------------------------------------------------------------------------------------------
    */

endmodule // VGA_LED_Emulator



//Primitime modules
/*
    ************************************************************************
    */
module decoder #(parameter N = 3) (input [N-1:0] DataIn, output logic [(1<<N)
    -1:0] DataOut);
         always_comb begin
                 DataOut <= 1 << DataIn;
          end
endmodule


module register(clock,reset,enable,datain,dataout);
parameter size = 16;

input logic clock;
input logic reset;
input enable;
input logic [size-1:0] datain;
output logic [size-1:0] dataout;

        always_ff @(posedge clock) begin
                if (reset)
                        dataout <= 0;
                else if(enable)
                        dataout <= datain;
        end

endmodule

/*
    ************************************************************************
    */


//Image display control
```

```
/*
    ********************************************************************************
    */
module icon_control(
input logic [9:0] cur_x, cur_y,
input logic [3:0] life_1,life_2,
output logic [3:0] xc,
output logic [4:0] yc,
output logic enable_c);

        logic [1:0] inlife;
        logic in_x1, in_x2, in_y;
        logic [3:0] xc1, xc2;
        logic [4:0] yc1, yc2;

        assign enable_c =  inlife[0] | inlife[1];

        assign xc1 = cur_x - 10'd41;
        assign xc2 = cur_x - 10'd541;

        assign yc1 = cur_y - 10'd21;
        assign yc2 = cur_y - 10'd5;

        assign yc = (inlife[1]) ? yc2 : yc1;
        assign xc = (inlife[1]) ? xc2 : xc1;

        assign in_x1 = (cur_x > 10'd40)&
                                (cur_x < (10'd57 + 10'd16*(life_1-4'd1)));

        assign in_x2 = (cur_x > 10'd540)&
                                (cur_x < (10'd557 + 10'd16*(life_2-4'd1)))
                                    ;

        assign in_y = (cur_y > 10'd20) & (cur_y < 10'd37);


        assign inlife[0] = in_x1 & in_y & (life_1[0] | life_1[1] | life_1[2]
            | life_1[3]);
        assign inlife[1] = in_x2 & in_y & (life_2[0] | life_2[1] | life_2[2]
            | life_2[3]);

endmodule // iconcontrol


module spritecontrol(
input logic [9:0] cur_x, cur_y,
input logic [9:0] px[8],
input logic [9:0] py[8],
input logic [3:0] shape[8],
input logic [2:0] count[8],
input logic inv[8],
output logic [4:0] x,
output logic [8:0] y,
```

```
output logic enable,
output logic invert);

parameter N_SPRITES = 8;

        logic visible[N_SPRITES];               //visibility of sprite
        logic inSprite[N_SPRITES];
        logic [8:0]sh_shape[N_SPRITES];

        genvar i;


        assign invert =/**/
                                        (inSprite[7] & inv[7]) |
                                        (inSprite[6] & inv[6]) |
                                        (inSprite[5] & inv[5]) |
                                        (inSprite[4] & inv[4]) |
                                        (inSprite[3] & inv[3]) |
                                        (inSprite[2] & inv[2]) |
                                        (inSprite[1] & inv[1]) |
                                        (inSprite[0] & inv[0]);


        assign enable =/**/
                                        (inSprite[7] & visible[7]) |
                                        (inSprite[6] & visible[6]) |
                                        (inSprite[5] & visible[5]) |
                                        (inSprite[4] & visible[4]) |
                                        (inSprite[3] & visible[3]) |
                                        (inSprite[2] & visible[2]) |
                                        (inSprite[1] & visible[1]) |
                                        (inSprite[0] & visible[0]);

        logic [9:0] ax, ay;

        assign ay =  /**/
                                        (inSprite[7]) ? (py[7] - sh_shape[7])
                                             :
                                        (inSprite[6]) ? (py[6] - sh_shape[6])
                                             :
                                        (inSprite[5]) ? (py[5] - sh_shape[5])
                                             :
                                        (inSprite[4]) ? (py[4] - sh_shape[4])
                                             :
                                        (inSprite[3]) ? (py[3] - sh_shape[3])
                                             :
                                        (inSprite[2]) ? (py[2] - sh_shape[2])
                                             :
                                        (inSprite[1]) ? (py[1] - sh_shape[1])
                                             :
                                        (inSprite[0]) ? (py[0] - sh_shape[0])
                                             :
                                        8'd0;
```

```
        assign ax =  /**/
                                        (inSprite[7]) ? px[7] :
                                        (inSprite[6]) ? px[6] :
                                        (inSprite[5]) ? px[5] :
                                        (inSprite[4]) ? px[4] :
                                        (inSprite[3]) ? px[3] :
                                        (inSprite[2]) ? px[2] :
                                        (inSprite[1]) ? px[1] :
                                        (inSprite[0]) ? px[0] :
                                        4'd0;

        assign y = cur_y + 16 - ay;
        assign x = cur_x + 16 - ax;


        generate
                for(i = 0; i < N_SPRITES; i++) begin: loop_spr
                        assign inSprite[i] =((cur_x > (px[i] - 10'd17))&
                                                                (cur_x < (px[
                                                                    i] + 10'
                                                                    d16 + 10'
                                                                    d32*(
                                                                    count[i
                                                                    ]-1)))&
                                                                (cur_y > (py[
                                                                    i] - 10'
                                                                    d17))&
                                                                (cur_y < (py[
                                                                    i] + 10'
                                                                    d16)));
                end
        endgenerate

        generate
                for(i = 0; i < N_SPRITES; i++) begin: loop_sha
                        assign sh_shape[i][8:5] = shape[i];
                        assign sh_shape[i][4:0] = 5'd0;
                end
        endgenerate


        generate
                for(i = 0; i < N_SPRITES; i++) begin: loop_vis
                        assign visible[i] = (count[i][0] | count[i][1] |
                            count[i][2]);
                end
        endgenerate

endmodule // spritecontrol


module groundcontrol(cur_x, cur_y, px, py, count, x, y, enable);
```

```
parameter N_GROUND = 9;

input logic [9:0] cur_x, cur_y;
input logic [9:0] px[N_GROUND];
input logic [9:0] py[N_GROUND];
input logic [2:0] count[N_GROUND];
output logic [4:0] x;
output logic [4:0] y;
output logic enable;

        logic visible[N_GROUND];                //visibility of sprite
        logic inSprite[N_GROUND];
        logic [9:0] ax, ay;

        genvar i;

        assign enable =/**/
                                        (inSprite[8] & visible[8]) |
                                        (inSprite[7] & visible[7]) |
                                        (inSprite[6] & visible[6]) |
                                        (inSprite[5] & visible[5]) |
                                        (inSprite[4] & visible[4]) |
                                        (inSprite[3] & visible[3]) |
                                        (inSprite[2] & visible[2]) |
                                        (inSprite[1] & visible[1]) |
                                        (inSprite[0] & visible[0]);

        assign ay =/**/
                                        (inSprite[8]) ? py[8] :
                                        (inSprite[7]) ? py[7] :
                                        (inSprite[6]) ? py[6] :
                                        (inSprite[5]) ? py[5] :
                                        (inSprite[4]) ? py[4] :
                                        (inSprite[3]) ? py[3] :
                                        (inSprite[2]) ? py[2] :
                                        (inSprite[1]) ? py[1] :
                                        (inSprite[0]) ? py[0] :
                                        4'd0;

        assign ax =/**/
                                        (inSprite[8]) ? px[8] :
                                        (inSprite[7]) ? px[7] :
                                        (inSprite[6]) ? px[6] :
                                        (inSprite[5]) ? px[5] :
                                        (inSprite[4]) ? px[4] :
                                        (inSprite[3]) ? px[3] :
                                        (inSprite[2]) ? px[2] :
                                        (inSprite[1]) ? px[1] :
                                        (inSprite[0]) ? px[0] :
                                        4'd0;

        assign y = cur_y + 16 - ay;
        assign x = cur_x + 16 - ax;
```

```
        generate
                for(i = 0; i < N_GROUND; i++) begin: loop_spr
                        assign inSprite[i] =((cur_x > (px[i] - 10'd17))&
                                                                (cur_x < (px[
                                                                    i] + 10'
                                                                    d16 + 10'
                                                                    d32*(
                                                                    count[i
                                                                    ]-1)))&
                                                                (cur_y > (py[
                                                                    i] - 10'
                                                                    d17))&
                                                                (cur_y < (py[
                                                                    i] + 10'
                                                                    d16)));
                end
        endgenerate

        generate
                for(i = 0; i < N_GROUND; i++) begin: loop_vis
                        assign visible[i] = (count[i][0] | count[i][1] |
                            count[i][2]);
                end
        endgenerate

endmodule // spritecontrol
/*
    ********************************************************************************
    */


//icons
/*
    ********************************************************************************
    */
module icons(
input logic clk50,
input logic [3:0] xc,
input logic [4:0] yc,
output logic [7:0] Ri,Gi,Bi,
output logic icon_a);

        logic [15:0] a;
        logic [63:0] c;
        logic [4:0] icon_c;

        assign icon_c = c[xc*4 +: 4];
        assign icon_a = a[xc];

        active_icon act_ic(clk50, yc, a);

        colors_icon col_ic(clk50, yc, c);
```

```
        icon_code code(icon_c, Ri, Gi, Bi);

endmodule


module colors_icon(
input logic clk ,
input logic [4:0] address,
output logic [63:0] data_out);

        always_ff @(posedge clk)
        begin
                data_out <= mem[address];
        end

        logic [63:0] mem[0:31] = '{
                //grandma
                64'h11111000000,
                64'h111111110000,
                64'h123221111000,
                64'h23222121000,
                64'h222222221000,
                64'h24422211000,
                64'h22222211000,
                64'h15555511000,
                64'h555555551100,
                64'h5555555555100,
                64'h2255555522100,
                64'h2226666222000,
                64'h2266666622000,
                64'h66006600000,
                64'h422002240000,
                64'h4444004444000,
                //grandpa
                64'h1111100000,
                64'h11111110110,
                64'h27221111110,
                64'h227222121000,
                64'h2212221121000,
                64'h111222211000,
                64'h22222200000,
                64'h888898880000,
                64'h8888888888000,
                64'h88888898888800,
                64'h22888888882200,
                64'h222aaaaaa22200,
                64'h22aaaaaaaa2200,
                64'haaa00aaa0000,
                64'hbbb0000bbb000,
                64'hbbbb0000bbbb00};
endmodule
```

```
module active_icon(
input logic clk ,
input logic [4:0] address,
output logic [15:0] data_out);

        always_ff @(posedge clk)
        begin
                data_out <= mem[address];
        end

        logic [15:0] mem[0:31] = '{
                //grandma
                16'h7c0,
                16'hff0,
                16'hff8,
                16'h7f8,
                16'hff8,
                16'h7f8,
                16'h7f8,
                16'h7f8,
                16'hffc,
                16'h1ffc,
                16'h1ffc,
                16'h1ff8,
                16'h1ff8,
                16'h660,
                16'he70,
                16'h1e78,
                //grandpa
                16'h3e0,
                16'h7f6,
                16'h7fe,
                16'hff8,
                16'h1ff8,
                16'hff8,
                16'h7e0,
                16'hff0,
                16'h1ff8,
                16'h3ffc,
                16'h3ffc,
                16'h3ffc,
                16'h3ffc,
                16'he70,
                16'h1c38,
                16'h3c3c};

endmodule
/*
    ********************************************************************************
    */
```

```
//Sprites
/*
    ********************************************************************************
    */

//Top Level
/*---------------------------*/
//General Sprites Color
module color_mem(
input logic [4:0]x,
input logic [8:0]y,
input logic invert,
output logic [4:0] code);

        logic [7:0]posx;

        logic [6:0]s_x;

        logic [159:0] col;

        assign s_x[6:2] = x;
        assign s_x[1:0] = 2'd0;
        assign posx = (invert) ? (155 - (s_x + x)):(s_x + x);
        assign code = col[posx +: 5];

        twoport_color color(y,col);
endmodule

//Ground Sprites Color
module color_gnd(
input logic [4:0]gx,
input logic [4:0]gy,
output logic [4:0] code);

        logic [7:0]posx;

        logic [6:0]s_x;

        logic [159:0] col;

        assign s_x[6:2] = gx;
        assign s_x[1:0] = 2'd0;
        assign posx = s_x + gx;
        assign code = col[posx +: 5];

        ground_color color(gy,col);

endmodule

//General Sprites Active
module active_mem(
input logic [4:0]x,
input logic [8:0]y,
```

```
input invert,
output logic Apix);

        logic [4:0] xa;
        logic [31:0] ac;

        assign xa = (invert) ? (31-x) : (x);

        assign Apix = ac[xa];

        //active memory
        twoport_active active(y,ac);

endmodule

//Ground Sprites Active
module active_gnd(
input logic [4:0]gx,
input logic [4:0]gy,
output logic Agnd);

        logic [31:0] ac;

        assign Agnd = ac[gx];

        //active memory
        ground_active active(gy,ac);

endmodule
/*---------------------------*/


//Memory Modules
/*---------------------------*/
//General Sprites Color memory
module twoport_color(
input logic [8:0] aa,
output logic [159:0] qa);

        assign qa = mem[aa];

        //Sprite memory
        logic [159:0] mem[0:191] = '{
                //ground
                160'h0,
                160'h0,
                160'h0,
                160'h0,
                160'h842100000000000000000000000000,
                160'h210842108400000000840000000008000,
                160'h842108421084421042108421084210842108421,
                160'h842108421084421084210842108421084108421,
                160'h8421084421086318c6318c6318c421086210823,
```

```
160'h18c6318c6310c6318c6318c6318c4210c6318c63,
160'h18c6318c6310c6318c6318c6318c421084318c43,
160'h18c6318c4218c6318c6318c4218c6318c6318842,
160'h18c631884318c6318c6318c4318c6318c6318c63,
160'h18c631086318c6318c6318c4318c6318c6318c63,
160'h108421086318c6318c6318c421086310c4210842,
160'h188421086318c6318c6318c42108421084210842,
160'h18c6318c631086318c6318c621084318c6318c63,
160'h18c6318c6318c42108421084210843l8c6318c63,
160'h18c6318c6318c6318c6318c631086318c6318c63,
160'h18c6318c6318c6318c6318c631086318c6318c63,
160'h2108421084210842108421084210842108421084,
160'h2108421084210842108421084210842108421084,
160'h2108421084210842108421084210842108421084,
160'h2108421084210842108421084210842108421084,
160'h2108421084210842108421084210842108421084,
160'h2108421084210842108421084210842108421084,
160'h2108421084210842108421084210842108421084,
160'h2108421084210842108421084210842108421084,
160'h2108421084210842108421084210842108421084,
160'h2108421084210842108421084210842108421084,
160'h2108421084210842108421084210842108421084,
160'h2108421084210842108421084210842108421084,
//grandma
160'h2108421084210842108421084210842108421084,
160'h2108421084210842108421084210842108421084,
160'h2108421084210842108421084210842108421084,
160'h2108421084210842108429484210842108421084,
160'h2108421084210842108421085318a4210842108421084,
160'h2108421084210842108421084210a6318c5210842108421084,
160'h2108421084210842108421085318a4210842108421084,
160'h2108421084210842108421085294a63148421084210842108421084,
160'h2108421084210a6318c6318a4210842108421084,
160'h2108421084214c6318c6318c5210842108421084,
160'h2108421084214c739ce739cc5210842108421084,
160'h2108421084214e7420e7420e7290842108421084,
160'h2108421084214e7420e7420e7290842108421084,
160'h2108421084214e739ce739ce7290842108421084,
160'h2108421084214e739ce739ce7290842108421084,
160'h2108421084214a739d2939ce5414842108421084,
160'h21084210852a10539ce739ca8420a42108421084,
160'h21084210a842108294a529508421052108421084,
160'h21084210a84210842108421084210521084210842,
160'h21084210a739508421084210829ce52108421084,
160'h21084210a7395084210842108829ce52108421084,
160'h21084210a739508421084210529ce52108421084,
160'h21084210a739ca8421084210539ce52108421084,
160'h21084210a73954a5294a5294529ce52108421084,
160'h21084210852a94a5294a52945214a42108421084,
160'h21084210842a94a294a5529452108421084210842,
160'h21084210842a94a2908552945210842108421084,
160'h21084210842a94a2908552945210842108421084,
160'h2108421085549ce72908539ce9290842108421084,
```

```
160'h21084214a949ce72908539ce9494a42108421084,
160'h210842a5294a529290854a5294a5252108421084,
160'h21085294a5294a529085294a5294a52908421084,
//grandma jump
160'h2108421084210842108429484210842108421084,
160'h210842108421084210851318a4210842108421084,
160'h21084210842108421084210a6318c5210842108421084,
160'h210842948421084210851318a421084294a421084,
160'h2108539ca421085294a6314842108539ce521084,
160'h210a739ce5210a6318c6318a4210a739ce729084,
160'h210a739ce5214c6318c6318c5210a739ce729084,
160'h2108539ce7294c739ce739cc5214e739ce521084,
160'h2108429ce7394e7420e7420e729ce739ca421084,
160'h21084214e7414e7420e7420e729ce73948421084,
160'h21084210a8414e739ce739ce72a0e72908421084,
160'h2108421085414e739ce739ce72a1052108421084,
160'h21084210842a0a739d2939ce5420a42108421084,
160'h21084210842150539ce739ca8414842108421084,
160'h21084210842150829 4a529508414842108421084,
160'h2108421084210a8421084210829084210842108421084,
160'h2108421084210a8421084210829084210842108421084,
160'h2108421084214a8421084210829084210842108421084,
160'h21084210842a1084210842108290842108421084,
160'h210842108542108421084210829084210842108421084,
160'h21084210aa5294a5294a5294a290842108421084,
160'h21084210aa5294a5294a5294a290842108421084,
160'h21084210a939ce5294a55294a290842108421084,
160'h2108421529 39ce5210855294a290842108421084,
160'h210842a5294a525210855294a290842108421084,
160'h210842a5294a4a5210a939ce5210842108421084,
160'h21084214a529484210a939ce5210842108421084,
160'h21084210842108421084210a94a525210842108421084,
160'h21084210842108421084210a94a525210842108421084,
160'h21084210842108421084210a94a525210842108421084,
160'h21084210842108421084210a94a525210842108421084,
160'h210842108421084210851294a4210842108421084,
//grandpa
160'h2108421084210842108421084210842108421084,
160'h2108421084210842108421084210842108421084,
160'h2108421084210842108421084210842108421084,
160'h2108421084210842948421084210842108421084,
160'h210842108421085318a42108421084210842108421084,
160'h210842108421085318c52108421084210842108421084,
160'h21084210842108429 8c52908421084210842108421084,
160'h210842108421085318c63148421084210842108421084,
160'h2108421084210a6318c6318a4210842108421084,
160'h2108421084214c6318c6318c5210842108421084,
160'h2108421084214c739ce739cc5210842108421084,
160'h2108421084214e75ace75ace7290842108421084,
160'h2108421084214e75ace75ace7290842108421084,
160'h2108421084214e739ce739ce7290842108421084,
160'h2108421084214e739cc639ce7290842108421084,
160'h2108421084214a7398c631ce5614842108421084,
```

```
160'h21084210852b18539ce739cac630a42108421084,
160'h21084210ac6318c294a52958c631852108421084,
160'h21084210ac6318c6318c6318c631852108421084,
160'h21084210a73958c6358c6b18c29ce52108421084,
160'h21084210a73958c6318c6318c29ce52108421084,
160'h21084210a7395ce739ce739c529ce52108421084,
160'h21084210a739cae739ce739c539ce52108421084,
160'h21084210a7395ce739ce739c529ce52108421084,
160'h21084210852b9ce739ce739c5214a42108421084,
160'h21084210842b9ce294a5739c5210842108421084,
160'h21084210842b9ce29085739c5210842108421084,
160'h21084210842b9ce29085739c5210842108421084,
160'h21084210857bdef290857bdef290842108421084,
160'h21084214a57bdef290857bdef294a42108421084,
160'h210842bdef7bdef290857bdef7bde52108421084,
160'h21085294a5294a529085294a5294a52908421084,
//grandpa jump
160'h2108421084210842948421084210842108421084,
160'h2108421084210853184210842108421084210842108421084,
160'h2108421084210853184c5210842108421084210842108421084,
160'h2108429484210842984c52908421084294a421084,
160'h2108539ca421085318c6314842108539ce521084,
160'h210a739ce5210a6318c6318a4210a739ce729084,
160'h210a739ce5214c6318c6318c5210a739ce729084,
160'h2108539ce7294c739ce739cc5214e739ce521084,
160'h2108429ce7394e75ace75ace729ce739ca421084,
160'h21084214e7614e75ace75ace729ce73948421084,
160'h21084210ac614e739ce739ce72b0e72908421084,
160'h2108421085614e739cc639ce72b1852108421084,
160'h21084210842b0a7398c631ce5630a42108421084,
160'h2108421084215853 9ce739cac614842108421084,
160'h2108421084215 8c294a52958c614842108421084,
160'h2108421084210ac6318c6318c290842108421084,
160'h2108421084210ac6358c6b18c290842108421084,
160'h2108421084214ac6318c6318c290842108421084,
160'h21084210842b9ce739ce739ce290842108421084,
160'h2108421085739ce739ce739ce290842108421084,
160'h21084210af7bdce739ce739ce290842108421084,
160'h21084210af7bdee739ce739ce290842108421084,
160'h21084210af7bdef294a5739ce290842108421084,
160'h21084215ef7bde521085739ce290842108421084,
160'h210842bdef7bde521085739ce290842108421084,
160'h210842bdef7bca5210af7bde5210842108421084,
160'h21084214a529484210af7bde5210842108421084,
160'h2108421084210842 10af7bde5210842108421084,
160'h2108421084210842 10af7bde5210842108421084,
160'h2108421084210842 10af7bde5210842108421084,
160'h2108421084210842 10af7bde5210842108421084,
160'h2108421084210842108421085294a4210842108421084,
//wheelchair
160'h210842108421084210842108421084210842108421084,
160'h210842108421084210842108421084210842108421084,
160'h210842108421084210842108421084210842108421084,
```

```
                160'h2108421084210842108421231810842108421084,
                160'h2108421084210842108424 6739ca318423194084,
                160'h2108421084210842108424a739ce739ce4421084,
                160'h2108421084210842108424e739ce739ce2421084,
                160'h2108421084210842421024e739ce739ce2421084,
                160'h2108421084242539ce739ce739ce739ce0421084,
                160'h2108421084242308408424e739ca5294a0421084,
                160'h2108421084810842108424e3294e739ca0421084,
                160'h21084210848c23194e739ca739ce739ca0421084,
                160'h21084210848ce739ce739ca739ce739ce3121084,
                160'h21084210848ce739ce739ce739ce739ce539c484,
                160'h21084210848ce739ce739ce319ce7394e3024a04,
                160'h21084210848ce739ce739ce739ce739ce3024644,
                160'h21084210848c2339ce739ce739ce739ce3089250,
                160'h21084210848ca738c2139ce728ca719465091211,
                160'h2108421211942712423 39c09094e528c67321211,
                160'h2108421210942442125394 6339ca2424a7184611,
                160'h2108424204892642461 18ca739ca5084e7281211,
                160'h2108424490912508921 0946538c62424e7121230,
                160'h2108424490946522123 19425284a248ce7089250,
                160'h2108484092940938425 18ca7194a3184e3024664,
                160'h210848c0939ce109c610946308c65024a3124624,
                160'h21084890939c6448ca739c49084a518c61084884,
                160'h24210812139ca4424a108ca308ca508909194084,
                160'h24e738109394e04844848ce72212729121381084,
                160'h2121194611944848908494e32212129ce5021084,
                160'h2108421232946319108494a52210842108421084,
                160'h2108421084846318908494651210842108421084,
                160'h2108421084210842108424a44210842108421084};

endmodule

//Ground Color memory
module ground_color(
input logic [4:0] aa,
output logic [159:0] qa);

        assign qa = mem[aa];

        //Sprite memory
        logic [159:0] mem[0:31] = '{
                //ground
                160'h0,
                160'h0,
                160'h0,
                160'h0,
                160'h842100000000000000000000000000,
                160'h2108421084000000008400000000 8000,
                160'h8421084210844210421084210842108421,
                160'h84210842108442108421084210842108421,
                160'h8421084421086318c6318c6318c421086210823,
                160'h18c6318c6310c6318c6318c6318c4210c6318c63,
                160'h18c6318c6310c6318c6318c6318c421084318c43,
```

```
                160'h18c6318c4218c6318c6318c4218c6318c6318842,
                160'h18c631884318c6318c6318c4318c6318c6318c63,
                160'h18c631086318c6318c6318c4318c6318c6318c63,
                160'h108421086318c6318c6318c421086310c4210842,
                160'h188421086318c6318c6318c42108421084210842,
                160'h18c6318c631086318c6318c621084318c6318c63,
                160'h18c6318c6318c4210842108421084318c6318c63,
                160'h18c6318c6318c6318c6318c631086318c6318c63,
                160'h18c6318c6318c6318c6318c631086318c6318c63,
                160'h2108421084210842108421084210842108421084,
                160'h2108421084210842108421084210842108421084,
                160'h2108421084210842108421084210842108421084,
                160'h2108421084210842108421084210842108421084,
                160'h2108421084210842108421084210842108421084,
                160'h2108421084210842108421084210842108421084,
                160'h2108421084210842108421084210842108421084,
                160'h2108421084210842108421084210842108421084,
                160'h2108421084210842108421084210842108421084,
                160'h2108421084210842108421084210842108421084,
                160'h2108421084210842108421084210842108421084,
                160'h2108421084210842108421084210842108421084};

endmodule

//General Sprites Active memory
module twoport_active(
input logic [7:0] aa,
output logic [31:0] qa);

        assign qa = mem[aa];

        logic [31:0] mem[0:191] = '{
                //ground
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
//grandma
32'h0,
32'h0,
32'h0,
32'hc000,
32'h1e000,
32'h3f000,
32'h1e000,
32'h1fc000,
32'h3fe000,
32'h7ff000,
32'h7ff000,
32'h7ff800,
32'h7ff800,
32'h7ff800,
32'h7ff800,
32'h7ffc00,
32'h1fffe00,
32'h3ffff00,
32'h3ffff00,
32'h3ffff00,
32'h3ffff00,
32'h3ffff00,
32'h3ffff00,
32'h3ffff00,
32'h1fff600,
32'hfff000,
32'hf9f000,
32'hf9f000,
32'h1f9f800,
32'h7f9fe00,
32'hff9ff00,
32'h1ff9ff80,
//grandma jump
32'hc000,
32'h1e000,
32'h3f000,
32'hc01e0e0,
32'h1e1fc1f0,
32'h3f3fe3f8,
32'h3f7ff3f8,
```

```
32'h1ffff7f0,
32'hffffffe0,
32'h7ffffc0,
32'h3ffff80,
32'h1ffff00,
32'hfffe00,
32'h7ffc00,
32'h7ffc00,
32'h3ff800,
32'h3ff800,
32'h7ff800,
32'hfff800,
32'h1fff800,
32'h3fff800,
32'h3fff800,
32'h3fff800,
32'h7f1f800,
32'hff1f800,
32'hff3f000,
32'h7c3f000,
32'h3f000,
32'h3f000,
32'h3f000,
32'h3f000,
32'h1e000,
//grandpa
32'h0,
32'h0,
32'h0,
32'hc0000,
32'h1e0000,
32'h1f0000,
32'hf8000,
32'h1fc000,
32'h3fe000,
32'h7ff000,
32'h7ff000,
32'h7ff800,
32'h7ff800,
32'h7ff800,
32'h7ff800,
32'h7ffc00,
32'h1fffe00,
32'h3ffff00,
32'h3ffff00,
32'h3ffff00,
32'h3ffff00,
32'h3ffff00,
32'h3ffff00,
32'h3ffff00,
32'h1fff600,
32'hfff000,
32'hf9f000,
```

```
32'hf9f000,
32'h1f9f800,
32'h7f9fe00,
32'hff9ff00,
32'h1ff9ff80,
//grandpa jump
32'hc0000,
32'h1e0000,
32'h1f0000,
32'hc0f80e0,
32'h1e1fc1f0,
32'h3f3fe3f8,
32'h3f7ff3f8,
32'h1ffff7f0,
32'hffffffe0,
32'h7ffffc0,
32'h3ffff80,
32'h1ffff00,
32'hfffe00,
32'h7ffc00,
32'h7ffc00,
32'h3ff800,
32'h3ff800,
32'h7ff800,
32'hfff800,
32'h1fff800,
32'h3fff800,
32'h3fff800,
32'h3fff800,
32'h7f1f800,
32'hff1f800,
32'hff3f000,
32'h7c3f000,
32'h3f000,
32'h3f000,
32'h3f000,
32'h3f000,
32'h1e000,
//wheelchair
32'h0,
32'h0,
32'h0,
32'h3800,
32'h7ffc,
32'h7fe0,
32'h7fe0,
32'h77fe0,
32'h7fffe0,
32'h7c7fe0,
32'h807fe0,
32'hfffffe0,
32'hffffff0,
32'hffffffc,
```

```
                32'hffffff6,
                32'hffffff6,
                32'hffffffb,
                32'hffffffb,
                32'h3f7dff3,
                32'h3e3fe7f,
                32'h6a7fffb,
                32'h5bbfe73,
                32'h5f3fefb,
                32'hddffff6,
                32'hdffff76,
                32'h9efdffc,
                32'h7be7ff9c,
                32'h79ecf3b8,
                32'h3fc8f3f0,
                32'h3f8f000,
                32'hf8f000,
                32'h6000};
endmodule

//Ground Sprites Active memory
module ground_active(
input logic [4:0] aa,
output logic [31:0] qa);

        assign qa = mem[aa];

        logic [31:0] mem[0:31] = '{
                //ground
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'hffffffff,
                32'h0,
                32'h0,
                32'h0,
                32'h0,
```

```
                32'h0,
                32'h0,
                32'h0,
                32'h0,
                32'h0,
                32'h0,
                32'h0,
                32'h0};
endmodule
/*--------------------------*/

/*
    ********************************************************************************
    */


//Code Converters
/*
    ********************************************************************************
    */
module icon_code(
input [4:0]code,
output logic [7:0] R, G, B);


        always_comb begin
                case(code)
                        5'h0:
                        begin
                                R = 8'd195;
                                G = 8'd255;
                                B = 8'd255;
                        end
                        5'h1:
                        begin
                                R = 8'd167;
                                G = 8'd167;
                                B = 8'd167;
                        end
                        5'h2:
                        begin
                                R = 8'd255;
                                G = 8'd215;
                                B = 8'd177;
                        end
                        5'h3:
                        begin
                                R = 8'd122;
                                G = 8'd255;
                                B = 8'd170;
                        end
                        5'h4:
                        begin
```

```
                R = 8'd255;
                G = 8'd0;
                B = 8'd0;
        end
        5'h5:
        begin
                R = 8'd62;
                G = 8'd226;
                B = 8'd136;
        end
        5'h6:
        begin
                R = 8'd20;
                G = 8'd77;
                B = 8'd255;
        end
        5'h7:
        begin
                R = 8'd51;
                G = 8'd226;
                B = 8'd255;
        end
        5'h8:
        begin
                R = 8'd255;
                G = 8'd255;
                B = 8'd255;
        end
        5'h9:
        begin
                R = 8'd255;
                G = 8'd255;
                B = 8'd0;
        end
        5'hA:
        begin
                R = 8'd138;
                G = 8'd87;
                B = 8'd23;
        end
        5'hB:
        begin
                R = 8'd0;
                G = 8'd0;
                B = 8'd0;
        end
        default:
        begin
                R = 8'd0;
                G = 8'd0;
                B = 8'd0;
        end
endcase
```

```
        end

endmodule

module color_code(
input [4:0]code,
output logic [7:0] R, G, B);

        always_comb begin
                case(code)
                        5'h0:
                        begin
                                R = 8'd148;
                                G = 8'd206;
                                B = 8'd0;
                        end
                        5'h1:
                        begin
                                R = 8'd49;
                                G = 8'd92;
                                B = 8'd5;
                        end
                        5'h2:
                        begin
                                R = 8'd69;
                                G = 8'd36;
                                B = 8'd1;
                        end
                        5'h3:
                        begin
                                R = 8'd120;
                                G = 8'd70;
                                B = 8'd14;
                        end
                        5'h4:
                        begin
                                R = 8'd195;
                                G = 8'd254;
                                B = 8'd252;
                        end
                        5'h5:
                        begin
                                R = 8'd40;
                                G = 8'd40;
                                B = 8'd40;
                        end
                        5'h6:
                        begin
                                R = 8'd167;
                                G = 8'd167;
                                B = 8'd167;
                        end
                        5'h7:
```

```
begin
        R = 8'd255;
        G = 8'd215;
        B = 8'd117;
end
5'h8:
begin
        R = 8'd62;
        G = 8'd226;
        B = 8'd136;
end
5'h9:
begin
        R = 8'd255;
        G = 8'd0;
        B = 8'd0;
end
5'hA:
begin
        R = 8'd20;
        G = 8'd77;
        B = 8'd255;
end
5'hB:
begin
        R = 8'd65;
        G = 8'd226;
        B = 8'd253;
end
5'hC:
begin
        R = 8'd255;
        G = 8'd255;
        B = 8'd255;
end
5'hD:
begin
        R = 8'd255;
        G = 8'd253;
        B = 8'd56;
end
5'hE:
begin
        R = 8'd138;
        G = 8'd87;
        B = 8'd23;
end
5'hF:
begin
        R = 8'd0;
        G = 8'd0;
        B = 8'd0;
end
```

```
                        5'h10:
                        begin
                                R = 8'd165;
                                G = 8'd206;
                                B = 8'd207;
                        end
                        5'h11:
                        begin
                                R = 8'd136;
                                G = 8'd156;
                                B = 8'd155;
                        end
                        5'h12:
                        begin
                                R = 8'd100;
                                G = 8'd109;
                                B = 8'd104;
                        end
                        5'h13:
                        begin
                                R = 8'd61;
                                G = 8'd63;
                                B = 8'd60;
                        end
                        default:
                        begin
                                R = 8'd0;
                                G = 8'd0;
                                B = 8'd0;
                        end
                endcase
        end

endmodule
/*
    ********************************************************************************
    */

module background(
input logic [9:0] cur_x,
input logic [9:0] cur_y,
output logic [7:0] R, G, B);

        logic [4:0] posx;
        logic [12:0] posy;

        logic [31:0] col;
        logic [1:0] code;


        assign posx[4:1] = cur_x[4:1];
        assign posx[0] = 1'b0;
```

```
        assign posy = cur_y[9:1]*20 + cur_x[9:5];
        assign code = col[posx +: 2];

        mem_back back(posy,col);

        back_code bcode(.*);

endmodule

module back_code(
input [1:0]code,
output logic [7:0] R, G, B);

        always_comb begin
                case(code)
                        2'b00:
                        begin
                                R = 8'h35;
                                G = 8'hbc;
                                B = 8'hff;
                        end
                        2'b01:
                        begin
                                R = 8'ha7;
                                G = 8'he3;
                                B = 8'hfc;
                        end
                        2'b10:
                        begin
                                R = 8'ha4;
                                G = 8'he4;
                                B = 8'hff;
                        end
                        2'b11:
                        begin
                                R = 8'hfb;
                                G = 8'hfc;
                                B = 8'hf9;
                        end
                        default:
                        begin
                                R = 8'd0;
                                G = 8'd0;
                                B = 8'd0;
                        end
                endcase // code
        end
endmodule

module mem_back(
input logic [12:0] aa,
output logic [31:0] qa);
```

```
assign qa = mem[aa];

logic [31:0] mem [0:4799] = '{
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
                                        32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h2aaaa000,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'haafffa00,
32'h2,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hefffffa0,
32'ha,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hbfffffff8,
32'h2b,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffffffff8,
32'h2d,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'haa90000,
32'hfffffffe,
32'h2b,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h2bffa000,
32'hfffffffe,
32'hae,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'hbffff800,
32'hfffffffe,
32'hbb,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hbffffe00,
32'hffffffff,
32'hae,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffffff00,
32'hbfffffff,
32'h7a800abb,
32'hb,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffffff80,
32'hfbffffff,
32'heee8adef,
32'haf,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h7fffffc0,
32'hfffbfffe,
32'hffbabbff,
32'h2bb,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'heeffffc0,
32'hfffffbb7,
32'hfffeeeff,
```

```
32'haef,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffffffa8,
32'hffffffeef,
32'hffffffbff,
32'haff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h82a00000,
32'hffeffefe,
32'hfffffffbf,
32'hffffffff,
32'h2abf,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'heaff0000,
32'hffffffff,
32'hffffffff,
32'hefffbff,
32'h2aaefe,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h2aaaa000,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfbffc000,
32'hffffffff,
32'hffffffff,
32'hffffffff,
32'hbff7abb,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'haafffa00,
32'h2,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfefff000,
32'hffffffff,
32'hfffffbf,
32'hffffdff,
32'hbfffdfaf,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'hefffffa0,
32'ha,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffffff000,
32'hfffffffff,
32'hfffffffff,
32'hfffffffff,
32'hbfffffff,
32'h2,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hbffffff8,
32'h2b,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffffffc00,
32'hfffffbfff,
32'hfffffffff,
32'hfffffffff,
32'hfffffffef,
32'h2,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffffffff8,
32'h2d,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffffffc00,
32'hfffffffff,
32'hffffffbf,
```

```
32'hffffdfe,
32'hffffffff,
32'ha,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'haa90000,
32'hfffffffe,
32'h2b,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffbff800,
32'hffffbfff,
32'hffffffff,
32'hffffffff,
32'hbffffffef,
32'ha,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h2bffa000,
32'hfffffffe,
32'hae,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffaff000,
32'hffffaeff,
32'hfeffffbf,
32'hffffefb,
32'hbbffffbb,
32'h2,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hbffff800,
32'hfffffffe,
32'hbb,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'hfeabe000,
32'hfffeab7b,
32'hfffffff7b,
32'hffffeeee,
32'haeeefaab,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hbfffffe00,
32'hffffffff,
32'hae,
32'h0,
32'h0,
32'h0,
32'h0,
32'h7a098000,
32'heee82aef,
32'h7b7ffeae,
32'hfffffb9ab,
32'h2bbba8ae,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffffff00,
32'hbffffffff,
32'h7a800abb,
32'hb,
32'h0,
32'h0,
32'h0,
32'he8000000,
32'hbba002aa,
32'hefffe2b,
32'hbbbb98ae,
32'h2aa802b,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'hffffff80,
32'hfbffffff,
32'heee8adef,
32'haf,
32'h0,
32'h0,
32'h0,
32'h80000000,
32'haa00000a,
32'hbbbbb802,
32'hadeaa0a9,
32'ha,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h7fffffc0,
32'hfffbfffe,
32'hffbabbff,
32'h2bb,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h6eeea000,
32'h2aaa000a,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'heeffffc0,
32'hffffbb7,
32'hfffeeeff,
32'haef,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'haa9a0000,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfffffffa8,
32'hffffffeef,
32'hffffffbff,
32'haff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h82a00000,
32'hffeffefe,
32'hffffffbf,
32'hfffffffff,
32'h2abf,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'heaff0000,
32'hffffffff,
32'hffffffff,
```

```
32'hefffbff,
32'h2aaefe,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfbffc000,
32'hffffffff,
32'hffffffff,
32'hffffffff,
32'hbff7abb,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfefff000,
32'hffffffff,
32'hfffffffbf,
32'hffffffdff,
32'hbfffdfaf,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'hffffff000,
32'hfffffffff,
32'hfffffffff,
32'hfffffffff,
32'hbfffffff,
32'h2,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffffffc00,
32'hfffffbfff,
32'hfffffffff,
32'hfffffffff,
32'hfffffffef,
32'h2,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffffffc00,
32'hfffffffff,
32'hfffffffbf,
32'hffffffdfe,
32'hfffffffff,
32'ha,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffbff800,
32'hfffbfff,
32'hffffffff,
32'hffffffff,
32'hbffffef,
32'ha,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffaff000,
32'hfffaeff,
32'hfeffffbf,
32'hffffefb,
32'hbbfffbb,
32'h2,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfeabe000,
32'hfffeab7b,
```

```
32'hfffffff7b,
32'hffffeeee,
32'haeeefaab,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h7a098000,
32'heee82aef,
32'h7b7ffeae,
32'hfffffb9ab,
32'h2bbba8ae,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'he8000000,
32'hbba002aa,
32'hefffe2b,
32'hbbbb98ae,
32'h2aa802b,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'ha0000000,
```

```
32'h2b,
32'h0,
32'h0,
32'h0,
32'h0,
32'h80000000,
32'haa00000a,
32'hbbbbb802,
32'hadeaa0a9,
32'ha,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfe000000,
32'h27f,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h6eeea000,
32'h2aaa000a,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffe00000,
32'hbff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'haa9a0000,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfff80000,
32'h2eff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfffd0000,
32'h290bbff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'he0000000,
32'hfffe002b,
32'h2bf9bbff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfa000000,
32'hfffe82bf,
32'hbffeefff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hff800000,
32'hfbbaebff,
32'hfffebbff,
32'h2,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hff800000,
```

```
32'hfffeafbb,
32'hfffbafff,
32'h2,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffe00000,
32'hfffff7fff,
32'hfffebbbf,
32'ha,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'heebb9000,
32'hfffeffff,
32'hffebaeff,
32'hab,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfbdff800,
32'hffffffff,
32'hffbfbfff,
32'h2a82bd,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfefffe00,
32'hfffeffff,
32'hffffffff,
32'h2afeaff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffffeaa0,
32'hffffffff,
32'hffffffff,
32'hafffbff,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'he8000000,
32'ha,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfffffbffa,
32'hfffffffff,
32'hfffffffff,
32'hbfffffff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfe800000,
32'haf,
32'h0,
32'h0,
32'h0,
32'h80000000,
32'hfffffffff,
32'hfffffffff,
32'hfffffefff,
32'h2bfffffff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffa00000,
32'h2ff,
32'h0,
32'h0,
32'h0,
32'h80000000,
```

```
32'hfffffffe,
32'hffffffff,
32'hfeffffff,
32'h2afffff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffe00000,
32'haa2bff,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfebbfbaa,
32'hfffef7ff,
32'hffffefff,
32'h1bbffbff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hf8000000,
32'hffaa0009,
32'hbfebfff,
32'h0,
32'h0,
32'h0,
32'h0,
32'hf8aee900,
32'hffefbefe,
32'hfaef7bff,
32'haefabbf,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'hfe800000,
32'hfeffa02f,
32'hbfffffff,
32'h0,
32'h0,
32'h0,
32'h0,
32'ha02aa000,
32'hffbaabbb,
32'heabbeaff,
32'ha68aee,
32'h0,
32'h0,
32'h0,
32'h0,
32'h2ba,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffe00000,
32'hbbfffaab,
32'hffffefff,
32'h2,
32'h0,
32'h0,
32'h0,
32'h0,
32'hedea0aaa,
32'ha0aa52ae,
32'h29a,
32'h0,
32'h0,
32'h0,
32'ha0000000,
32'h2bff,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfff80000,
32'hffffeffe,
32'hffebbbfb,
32'h2a,
32'h0,
32'h0,
32'h0,
32'h0,
32'hbba80000,
32'h802a80ab,
32'h2a,
32'h0,
32'h0,
```

```
32'h0,
32'he8000000,
32'hbfff,
32'h0,
32'h0,
32'h0,
32'h0,
32'hbffebe80,
32'hffffffff,
32'hbf7ffaaf,
32'ha0af,
32'h0,
32'h0,
32'h0,
32'h0,
32'haa800000,
32'ha,
32'h0,
32'h0,
32'h0,
32'h0,
32'hf8000000,
32'h2a8affff,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfffbfff8,
32'hfffffbfff,
32'hffffffbf,
32'hbfaff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hea80027e,
32'hffaffffff,
32'h2,
32'h0,
32'h0,
32'h0,
32'hfffffffe,
32'hfffffffff,
32'hfffffffef,
32'h2fffff,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'ha0000000,
32'hbfe80bff,
32'hffffffff,
32'h2f,
32'h0,
32'h0,
32'h0,
32'hffbbfffe,
32'hffffffff,
32'hfbffffff,
32'h2fffff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hf8000000,
32'hfffeaaff,
32'hfffbffee,
32'hbf,
32'h0,
32'h0,
32'h0,
32'hfeab7ff8,
32'hfffbeeff,
32'haaffffbb,
32'hbf6ff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfe000000,
32'hfffbffbf,
32'hfaeefeff,
32'habf,
32'h0,
32'h0,
```

```
32'h0,
32'haa0aaaa0,
32'hbfeeabbb,
32'h80aefaaf,
32'h2a2a6,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffafa000,
32'hfffffffef,
32'hdffeabff,
32'h282bef,
32'h0,
32'h0,
32'h0,
32'h80000000,
32'hef6802aa,
32'haa02a,
32'h28,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfefffe00,
32'hffeffffff,
32'hfffffefff,
32'h2febfff,
32'h0,
32'h0,
32'h0,
32'h0,
32'haa800000,
32'ha,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'hffffff80,
32'hffffffff,
32'hffffffbff,
32'hbffffff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'heeffff80,
32'hffffffff,
32'hffffffff,
32'hbffffe,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'haadffe00,
32'hfefbbfff,
32'hbfffeeff,
32'h2fdbfea,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h82aaa800,
32'hfbaaeeea,
32'h2bbeabef,
32'ha8a9a0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hda00aaa0,
32'h2a80abb,
32'ha00,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'ha0000000,
32'h2aa,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h80000000,
32'hae,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'he8000000,
32'haff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfa000000,
32'h2fff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfe000000,
32'haa2bfff,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hba000000,
32'h2,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h80000000,
32'hfaa0009f,
32'hbfebffff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffa00000,
32'h2b,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'he8000000,
32'heffa02ff,
32'hffffffff,
32'hb,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfff82900,
32'hae,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'hfe000000,
32'hbfffaabf,
32'hfffefffb,
32'h2f,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfffebfa0,
32'h2aa82ab,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hff800000,
32'hfffeffef,
32'hfebbbfbf,
32'h2af,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hbbc0000,
32'hfffbffea,
32'h2aeeeadf,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffebe800,
32'hffffffb,
32'hf7ffaaff,
32'ha0afb,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'ha7ff0000,
32'hffe6eb9f,
32'hbbbffb77,
32'ha,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffbfff80,
32'hffffbffff,
32'hffffffbff,
32'hbfafff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hefff8000,
32'heeffbfff,
32'hfeeffeef,
32'h2af,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfffffffe0,
32'hffffffff,
32'hfffffeff,
32'h2ffffff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hbb7b8000,
32'hffbffbbf,
32'hfa7fffff,
32'haff,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfbbfffe0,
32'hfffffffff,
32'hbffffffff,
32'h2ffffff,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'heaee0000,
32'hffbbfaae,
32'heaefbbbb,
32'h2bb,
32'h0,
32'h0,
32'h0,
32'h0,
32'heab7ff80,
32'hffbeefff,
32'hafffbbf,
32'hbf6ffa,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'ha2a80000,
32'hefaee0aa,
32'h82bae9ee,
32'h2a,
32'h0,
32'h0,
32'h0,
32'h0,
32'ha0aaaa00,
32'hfeeabbba,
32'haefaafb,
32'h2a2a68,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hba29800a,
32'haa82ab,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'hf6802aa8,
32'haa02ae,
32'h280,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'ha8000000,
32'ha,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'ha8000000,
32'haa,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h2aaaa000,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h2aaaa00,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'haafffa00,
32'h2,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h2aafffa0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h80000000,
32'hae,
32'h0,
32'h0,
32'h0,
32'h0,
32'heffffffa0,
32'ha,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'haeffffffa,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'he8000000,
32'haff,
32'h0,
32'h0,
32'h0,
32'h0,
32'hbfffffff8,
32'h2b,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h80000000,
32'hbbfffffff,
32'h2,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfe0a4000,
32'h2bbf,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'hffffffff8,
32'h2d,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h80000000,
32'hdfffffff,
32'h2,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffafe800,
32'haaa0aaff,
32'h0,
32'h0,
32'h0,
32'haa90000,
32'hfffffffe,
32'h2b,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'he0aa9000,
32'hbfffffff,
32'h2,
32'h0,
32'h0,
32'h0,
32'hef000000,
32'hfefffa82,
32'hbbbab7ff,
32'ha,
32'h0,
32'h0,
32'h2bffa000,
32'hfffffffe,
32'hae,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'he2bffa00,
32'hefffffff,
32'ha,
32'h0,
```

```
32'h0,
32'h0,
32'hffc00000,
32'hf9bae7e9,
32'heffeddff,
32'h2ae,
32'h0,
32'h0,
32'hbffff800,
32'hfffffffe,
32'hbb,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hebffff80,
32'hbfffffff,
32'hb,
32'h0,
32'h0,
32'h0,
32'hffe00000,
32'hbfeffffb,
32'hbbffbbfb,
32'habff,
32'h0,
32'h0,
32'hbffffe00,
32'hffffffff,
32'hae,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfbffffe0,
32'heffffffff,
32'ha,
32'h0,
32'h0,
32'h0,
32'hdee00000,
32'heffeefee,
32'h9fffffff,
32'h2bffe,
32'h0,
32'h0,
32'hffffff00,
32'hbfffffff,
32'h7a800abb,
32'hb,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'hfffffff0,
32'hbbffffff,
32'hb7a800ab,
32'h0,
32'h0,
32'h0,
32'hbb800000,
32'heefeabba,
32'hbbeeeeff,
32'haefa,
32'h0,
32'h0,
32'hffffff80,
32'hfbffffff,
32'heee8adef,
32'haf,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffffffff8,
32'hffbfffff,
32'hfeee8ade,
32'ha,
32'h0,
32'h0,
32'haa000000,
32'hebb82aa8,
32'haeba7bbb,
32'haa0,
32'h0,
32'h0,
32'h7fffffc0,
32'hfffbfffe,
32'hffbabbff,
32'h2bb,
32'h0,
32'h0,
32'h0,
32'h0,
32'he7fffffc,
32'hfffffbfff,
32'hbffbabbf,
32'h2b,
32'h0,
32'h0,
32'h0,
32'h8a600280,
32'h2aa0aaee,
32'h0,
```

```
32'h0,
32'h0,
32'heeffffc0,
32'hffffffbb7,
32'hfffeeeff,
32'haef,
32'h0,
32'h0,
32'h0,
32'h0,
32'h7eeffffc,
32'hffffffbb,
32'hffffeeef,
32'hae,
32'h0,
32'h0,
32'h0,
32'h0,
32'h2aa,
32'h0,
32'h0,
32'h0,
32'hfffffffa8,
32'hffffffeef,
32'hffffffbff,
32'haff,
32'h0,
32'h0,
32'h0,
32'h80000000,
32'hffffffffa,
32'hffffffee,
32'hffffffbf,
32'haf,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h82a00000,
32'hffeffefe,
32'hffffffbf,
32'hfffffffff,
32'h2abf,
32'h0,
32'h0,
32'h0,
32'he82a0000,
32'hfffeffef,
32'hffffffffb,
32'hfffffffff,
```

```
32'h2ab,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'heaff0000,
32'hffffffff,
32'hffffffff,
32'hefffbff,
32'h2aaefe,
32'h0,
32'h0,
32'h0,
32'hfeaff000,
32'hffffffff,
32'hffffffff,
32'heeffffbf,
32'h2aaef,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfbffc000,
32'hffffffff,
32'hffffffff,
32'hffffffff,
32'hbff7abb,
32'h0,
32'h0,
32'h0,
32'hffbffc00,
32'hffffffff,
32'hffffffff,
32'hbffffffff,
32'hbff7ab,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfefff000,
32'hffffffff,
32'hfffffbf,
32'hffffdff,
32'hbfffdfaf,
```

```
32'h0,
32'h0,
32'h0,
32'hffefff00,
32'hffffffff,
32'hfffffffb,
32'hfffffdf,
32'hbfffdfa,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffffff000,
32'hffffffff,
32'hffffffff,
32'hffffffff,
32'hbfffffff,
32'h2,
32'h0,
32'h0,
32'hfffffff00,
32'hffffffff,
32'hffffffff,
32'hffffffff,
32'h2bfffffff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffffffc00,
32'hfffffbfff,
32'hffffffff,
32'hffffffff,
32'hfffffffef,
32'h2,
32'h0,
32'h0,
32'hffffffc0,
32'hffffffbff,
32'hffffffff,
32'hffffffff,
32'h2ffffffe,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'hffffffc00,
32'hffffffff,
32'hfffffffbf,
32'hffffffdfe,
32'hffffffff,
32'ha,
32'h0,
32'h0,
32'hfffffffc0,
32'hffffffff,
32'hefffffffb,
32'hfffffffdf,
32'hafffffff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffbff800,
32'hfffffbfff,
32'hffffffff,
32'hffffffff,
32'hbffffffef,
32'ha,
32'h0,
32'h0,
32'hfffbff80,
32'hffffffbff,
32'hffffffff,
32'hffffffff,
32'habffffffe,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffaff000,
32'hffffaeff,
32'hfeffffbf,
32'hffffffefb,
32'hbbffffbb,
32'h2,
32'h0,
32'h0,
32'hfffaff00,
32'hffffffaef,
32'hbfeffffb,
```

```
32'hbfffffef,
32'h2bbffffb,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfeabe000,
32'hfffeab7b,
32'hfffffff7b,
32'hffffeeee,
32'haeeefaab,
32'h0,
32'h0,
32'h0,
32'hbfeabe00,
32'hbfffeab7,
32'hefffffff7,
32'hbffffeee,
32'haeeefaa,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h7a098000,
32'heee82aef,
32'h7b7ffeae,
32'hffffb9ab,
32'h2bbba8ae,
32'h0,
32'h0,
32'h0,
32'hf7a09800,
32'heeee82ae,
32'hb7b7ffea,
32'hefffb9a,
32'h2bbba8a,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'he8000000,
32'hbba002aa,
32'hefffe2b,
32'hbbbb98ae,
```

```
32'h2aa802b,
32'h0,
32'h0,
32'h0,
32'hae800000,
32'hbbba002a,
32'heeffffe2,
32'hbbbbb98a,
32'h2aa802,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h80000000,
32'haa00000a,
32'hbbbbb802,
32'hadeaa0a9,
32'ha,
32'h0,
32'h0,
32'h0,
32'ha8000000,
32'h2aa00000,
32'h9bbbbb80,
32'haadeaa0a,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h6eeea000,
32'h2aaa000a,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'ha6eeea00,
32'h2aaa000,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'haa9a0000,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'haa9a000,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'he8000000,
32'ha,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hff800000,
32'h9f,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfff80000,
32'h2ff,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfffe0000,
32'hbbf,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffff4000,
32'ha42eff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'hf8000000,
32'hffff800a,
32'hafe6eff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfe800000,
32'hffffa0af,
32'h2fffbbff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffe00000,
32'hfeeebaff,
32'hbfffaeff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffe00000,
32'hffffabee,
32'hbffeebff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h2ba0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfff80000,
32'hffffdfff,
32'hbfffaeef,
32'h2,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h27ffe,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfbaee400,
32'hffffbfff,
32'hfffaebbf,
32'h2a,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'he0000000,
32'hbffff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfef7fe00,
32'hffffffff,
32'h7fefefff,
32'haa0af,
32'h0,
32'h0,
32'h0,
32'h0,
32'hf8000000,
32'h2effff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffbfff80,
32'hfffbfff,
32'hffffffff,
32'habfabf,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfd000000,
32'h90bbffff,
32'h2,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'hfffffaa8,
32'hffffffff,
32'hffffffff,
32'h2bffeff,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfe002be0,
32'hf9bbffff,
32'h2b,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h80000000,
32'hfffffeffe,
32'hffffffff,
32'hffffffff,
32'h2ffffff,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfe82bffa,
32'hfeeffffff,
32'hbf,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'he0000000,
32'hfffffffff,
32'hfffffffff,
32'hffffffbff,
32'haffffff,
32'h0,
32'h0,
32'h0,
32'h80000000,
32'hbaebffff,
32'hfebbfffb,
32'h2ff,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'ha0000000,
32'hffffffff,
32'hffffffff,
32'hffbfffff,
32'habffffff,
32'h0,
32'h0,
32'h0,
32'h80000000,
32'hfeafbbff,
32'hfbaffffff,
32'h2ff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h80000000,
32'hffaefeea,
32'hfffffbdff,
32'hffffffbff,
32'h6effeff,
32'h0,
32'h0,
32'h0,
32'he0000000,
32'hff7fffff,
32'hfebbbfff,
32'haff,
32'h0,
32'h0,
32'h0,
32'h0,
32'hae80000,
32'h0,
32'h0,
32'h0,
32'h0,
32'hbe2bba40,
32'hfffbefbf,
32'hfebbdeff,
32'h2bbeaef,
32'h0,
```

```
32'h0,
32'h0,
32'hbb900000,
32'hfeffffee,
32'hebaeffff,
32'habff,
32'h0,
32'h0,
32'h0,
32'h0,
32'haffe8000,
32'h0,
32'h0,
32'h0,
32'h0,
32'he80aa800,
32'hffeeaaee,
32'hbaaefabf,
32'h29a2bb,
32'h0,
32'h0,
32'h0,
32'hdff80000,
32'hfffffffb,
32'hbfbfffff,
32'h2a82bdff,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffffa000,
32'h2,
32'h0,
32'h0,
32'h0,
32'h80000000,
32'hbb7a82aa,
32'ha82a94ab,
32'ha6,
32'h0,
32'h0,
32'h0,
32'hfffe0000,
32'hfeffffe,
32'hffffffff,
32'hafeaffff,
32'h2,
32'h0,
32'h0,
32'h0,
32'hffffe000,
32'haa2b,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'heeea0000,
32'ha00aa02a,
32'ha,
32'h0,
32'h0,
32'h0,
32'hffeaa000,
32'hffffffff,
32'hffffffff,
32'hfffbffff,
32'ha,
32'h0,
32'h0,
32'h9f80000,
32'hffffaa00,
32'hbfebf,
32'h0,
32'h0,
32'h0,
32'h0,
32'haaa00000,
32'h2,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffbffa00,
32'hffffffff,
32'hffffffff,
32'hffffffff,
32'hb,
32'h0,
32'h0,
32'h2ffe8000,
32'hfffeffa0,
32'hbffffff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfffffff80,
32'hffffffff,
32'hffeffffff,
32'hffffffff,
```

```
32'h2b,
32'h0,
32'h0,
32'habffe000,
32'hffbbfffa,
32'h2ffffef,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffffffe80,
32'hfffffffff,
32'hfffffffff,
32'hfffffffe,
32'h2a,
32'h0,
32'h0,
32'hfefff800,
32'hfbffffef,
32'h2affebbb,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hbbfbaa00,
32'hfef7fffe,
32'hffeffffff,
32'hbffbffff,
32'h1b,
32'h0,
32'h80000000,
32'hffbffebe,
32'haffffffff,
32'hafbf7ffa,
32'ha0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'haee90000,
32'hefbefef8,
32'hef7bffff,
32'hefabbffa,
32'ha,
32'h0,
32'hf8000000,
32'hffffffbff,
32'hbffffffbf,
32'hfffffffff,
32'hbfa,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h2aa00000,
32'hbaabbba0,
32'hbbeaffff,
32'ha68aeeea,
32'h0,
32'h0,
32'hfe000000,
32'hffffffff,
32'heffffffff,
32'hfffffffff,
32'h2fff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hea0aaa00,
32'haa52aeed,
32'h29aa0,
32'h0,
32'h0,
32'hfe000000,
32'hffffbbff,
32'hfffffffff,
32'hfffbffff,
```

```
32'h2fff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'ha8000000,
32'h2a80abbb,
32'h2a80,
32'h0,
32'h0,
32'hf8000000,
32'hfffeab7f,
32'hbbfffbee,
32'hffaaffff,
32'hbf6,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h80000000,
32'haaa,
32'h0,
32'h0,
32'h0,
32'ha0000000,
32'hbbaa0aaa,
32'hafbfeeab,
32'ha680aefa,
32'h2a2,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'haa800000,
32'h2aef6802,
32'h28000aa0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'haaa8000,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h2ba,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h2ba,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'ha0000000,
32'h2bff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'ha0000000,
32'h2bff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'he8000000,
32'hbfff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'he8000000,
32'hbfff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hf8000000,
32'h2a8affff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hf8000000,
32'h2a8affff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hea80027e,
```

```
32'hffaffffff,
32'h2,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hea80027e,
32'hffaffffff,
32'h2,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'ha0000000,
32'hbfe80bff,
32'hfffffffff,
32'h2f,
32'h0,
32'h0,
32'h0,
32'h0,
32'ha0000000,
32'hbfe80bff,
32'hfffffffff,
32'h2f,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hf8000000,
32'hfffeaaff,
32'hfffbffee,
32'hbf,
32'h0,
32'h0,
32'h0,
32'h0,
32'hf8000000,
32'hfffeaaff,
32'hfffbffee,
32'hbf,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfe000000,
32'hffffbffbf,
32'hfaeefeff,
32'habf,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfe000000,
32'hffffbffbf,
32'hfaeefeff,
32'habf,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffafa000,
32'hffffffef,
32'hdffeabff,
32'h282bef,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffafa000,
32'hffffffef,
32'hdffeabff,
32'h282bef,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hfefffe00,
32'hffefffff,
32'hffffefff,
32'h2febfff,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'hfefffe00,
32'hffefffff,
32'hffffefff,
32'h2febfff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffffff80,
32'hffffffff,
32'hffffffbff,
32'hbfffffff,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffffff80,
32'hffffffff,
32'hffffffbff,
32'hbfffffff,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'heeffff80,
32'hffffffff,
32'hffffffff,
32'hbffffffe,
32'h0,
32'h2ba0000,
32'h0,
32'h0,
32'heeffff80,
32'hffffffff,
32'hffffffff,
32'hbffffffe,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'haadffe00,
```

```
32'hfefbbfff,
32'hbfffeeff,
32'h2fdbfea,
32'h0,
32'h2bffa000,
32'h0,
32'h0,
32'haadffe00,
32'hfefbbfff,
32'hbfffeeff,
32'h2fdbfea,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h2ba0,
32'h0,
32'h0,
32'h82aaa800,
32'hfbaaeeea,
32'h2bbeabef,
32'ha8a9a0,
32'h0,
32'haefff829,
32'h0,
32'h0,
32'h82aaa800,
32'hfbaaeeea,
32'h2bbeabef,
32'ha8a9a0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h2bffa,
32'h0,
32'h0,
32'h0,
32'hda00aaa0,
32'h2a80abb,
32'ha00,
32'ha0000000,
32'habfffebf,
32'h2aa82,
32'h0,
32'h0,
32'hda00aaa0,
32'h2a80abb,
32'ha00,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h82900000,
32'haefff,
32'h0,
32'h0,
32'h0,
32'ha0000000,
32'h2aa,
32'h0,
32'hea0bbc00,
32'hdffffbff,
32'h2aeeea,
32'h0,
32'h0,
32'ha0000000,
32'h2aa,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hebfa0000,
32'ha82abfff,
32'h2a,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h9fa7ff00,
32'h77ffe6eb,
32'habbbffb,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hc0000000,
32'hbffea0bb,
32'heeadffff,
32'h2ae,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hffefff80,
32'hefeeffbf,
32'haffeeffe,
```

```
32'h2,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hf0000000,
32'h6eb9fa7f,
32'hffb77ffe,
32'habbb,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hbfbb7b80,
32'hffffbffb,
32'hfffa7fff,
32'ha,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hf8000000,
32'hfbfffeff,
32'hffeefeef,
32'h2affee,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'haeeaee00,
32'hbbffbbfa,
32'hbbeaefbb,
32'h2,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'hb8000000,
32'hffbbfbb7,
32'hffffffb,
32'hafffa7,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'haaa2a800,
32'heeefaee0,
32'h2a82bae9,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'he0000000,
32'hbfaaeeae,
32'hfbbbbffb,
32'h2bbeae,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'ha000000,
32'habba2980,
32'haa82,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h80000000,
32'hee0aaa2a,
32'hae9eeefa,
32'h2a82b,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'haa80000,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h9800a000,
32'ha82abba2,
32'ha,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'haa80,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
32'h0,
```

```
                                                    32'h0,
                                                    32'h0,
                                                    32'h0,
                                                    32'h0,
                                                    32'h0,
                                                    32'h0,
                                                    32'h0,
                                                    32'h0,
                                                    32'h0,
                                                    32'h0,
                                                    32'h0,
                                                    32'h0,
                                                    32'h0,
                                                    32'h0,
                                                    32'h0,
                                                    32'h0,
                                                    32'h0,
                                                    32'h0,
                                                    32'h0,
                                                    32'h0,
                                                    32'h0,
                                                    32'h0,
                                                    32'h0,
                                                    32'h0,
                                                    32'h0,
                                                    32'h0,
                                                    32'h0,
                                                    32'h0,
                                                    32'h0,
                                                    32'h0};
endmodule // mem_back
```

3 Software/game.c

```
/*
 * Userspace program that communicates with the vga_led device driver
 * primarily through ioctls
 *
 * Stephen A. Edwards
 * Columbia University
 *
 * Name: Bernardo de Almeida Abreu
 * UNI:  bd2440
 *
 * Name: Henrique Pizzol Grando
 * UNI:  hp2409
 *
 * Name: Lucas Ikenaga Barros
 * UNI:  li2176
 *
 * Name: Tomas Mantelato
 * UNI:  tm2779
 *
 */

#include "util.h"
#include "input_user.h"

int vga_led_fd;
screen back;
sprite_info ground[3][3];
//sprite_info power_sprite;
int line_length[3] = { -1, -1, -1 };

int main()
{
    static const char filename[] = "/dev/vgaled";

    printf("GAME TEST Userspace program started\n");

    if ( (vga_led_fd = open(filename, O_RDWR)) == -1)
    {
        fprintf(stderr, "could not open %s\n", filename);
        return -1;
    }

    // Seeding srand
    srand(time(NULL));

    // Cleaning sprites
    clean();

    // Generating background
    back.life_1 = 4;
```

```
back.life_2 = 4;
// back.choice = COLOR;
back.background_color = 0xff0000;

// Printing the background
write_screen(back);

// Generating start ground
int row = 0;                        // The next row of ground to be generated
generate_ground(320, row++);
generate_ground(160, row++);
generate_ground(0, row++);
row = 0;

// Writing ground driver
int i, j;
for (i = 0; i < 3; i++)
{
    for (j = 0; j < line_length[i]; j++)
    {
        write_sprite(ground[i][j]);
    }
}

// Setting grandpa and grandma starting positions
sprite_info grandpa_sprite;
grandpa_sprite.pos.y = 200;
grandpa_sprite.pos.x = 200;
grandpa_sprite.shape = GP_STAND;
grandpa_sprite.id = GP_ID;
grandpa_sprite.count = 1;
grandpa_sprite.layer = OBJECTS;
grandpa_sprite.orientation = RIGHT;
write_sprite(grandpa_sprite);

sprite_info grandma_sprite;
grandma_sprite.pos.y = 200;
grandma_sprite.pos.x = 260;
grandma_sprite.shape = GM_STAND;
grandma_sprite.id = GM_ID;
grandma_sprite.count = 1;
grandma_sprite.layer = OBJECTS;
grandma_sprite.orientation = LEFT;
write_sprite(grandma_sprite);

// Creating characters structures
character grandpa;
grandpa.pos = &(grandpa_sprite.pos);
grandpa.id = GP_ID;
grandpa.vx = 0;
grandpa.vy = 1;
grandpa.jumping = 0;
```

```
character grandma;
grandma.pos = &(grandma_sprite.pos);
grandma.id = GM_ID;
grandma.vx = 0;
grandma.vy = 1;
grandma.jumping = 0;

// Starting peripheric
start_user_input();

int count_ground = 0;
while (1)
{
    // Line of platforms have a 175 pixels gap between one another
    if (count_ground == 35)
    {
        generate_ground(-8, row);
        row = (row + 1) % 3;
        count_ground = 0;
    }
    count_ground++;

    // Moves all platforms 5 pixels down
    int i, j;
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < line_length[i]; j++)
        {
            ground[i][j].pos.y += 5;
        }
    }

    // Writing to drivers
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < line_length[i]; j++)
        {
            write_sprite(ground[i][j]);
        }
    }

    int input = input_from_user();

    // grandpa under user motion capture
    if (input !=  -100)
    {
        grandpa.vx = -2 * input;
    }


    grandma.vx = 1;

    // Try to move grandpa
```

```
x_translation (&grandpa, grandma);
y_translation (&grandpa, grandma);


if (grandpa.jumping)        // Fall
{
    grandpa_sprite.shape = GP_JUMP;
    grandpa.vy += 1;
}
else                        // Jump
{
    grandpa_sprite.shape = GP_STAND;
    grandpa.jumping = 1;
    grandpa.vy = -13;
}

write_sprite(grandpa_sprite);

// Try to move grandma
x_translation (&grandma, grandpa);
y_translation (&grandma, grandpa);

if (grandma.jumping)        // Fall
{
    grandma_sprite.shape = GM_JUMP;
    grandma.vy += 1;
}
else                        // Jump
{
    grandma_sprite.shape = GM_STAND;
    grandma.jumping = 1;
    grandma.vy = -13;
}

write_sprite(grandma_sprite);

// Check for end of game
if (grandpa.pos->y >= 480 )
{
    if (!(--back.life_2))
    {
        back.choice = COLOR;
        write_screen(back);
        exit(1);
    }
    write_screen(back);
    grandpa.pos->y = 20;
    write_sprite(grandpa_sprite);
}
if (grandma.pos->y >= 480)
{
    grandma.pos->y = 20;
    write_sprite(grandma_sprite);
```

```
        }

        usleep(30000);

    }

    printf("GAME␣TEST␣Userspace␣program␣terminating\n");

    return 0;
}
```

4 Software/input_user.c

```
#include "input_user.h"
#include <stdlib.h>
#include <stdio.h>


void start_user_input(){
  if ( (keyboard = openkeyboard(&endpoint_address)) == NULL ) {
    fprintf(stderr, "Did␣not␣find␣the␣accelerometer␣(keyboard)\n");
  }
}

int input_from_user(){
        libusb_interrupt_transfer(keyboard, endpoint_address,
                  (unsigned char *) &packet, sizeof(packet),
                 &transferred, 0);
    int ret = 0;
    if (transferred == sizeof(packet)) {
        ret = get_number(packet.keycode[0]);
      }
    return ret;

}
int get_number(uint8_t keycode){
     char c = 'a' + keycode - 0x04;
     if (keycode == 0x00 ) return -100;

   //pegar do bernardo.
  //retornar int entre -10 e +10;
  // se for precisar diminuir a escala, faca isso aqui.
  int ret = (int) (c - 107);
  return ret;
}
```

5 Software/input_user.h

```
#ifndef _INPUT_USER_H_
#define _INPUT_USER_H_

#include "input_user.h"
#include "usbkeyboard.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>
#include <unistd.h>

#define SERVER_HOST "192.168.1.1"
#define SERVER_PORT 42000

#define BUFFER_SIZE 128

/*
 * References:
 *
 * http://beej.us/guide/bgnet/output/html/singlepage/bgnet.html
 * http://www.thegeekstuff.com/2011/12/c-socket-programming/
 *
 */


struct libusb_device_handle *keyboard;
uint8_t endpoint_address;

  int err, col;

  struct sockaddr_in serv_addr;

  struct usb_keyboard_packet packet;
  int transferred;
  char keystate[12];



void start_user_input();


int input_from_user();


int get_number(uint8_t keycode);


#endif
```

6 Software/usbkeyboard.c

```c
#include "usbkeyboard.h"

#include <stdio.h>
#include <stdlib.h>

/* References on libusb 1.0 and the USB HID/keyboard protocol
 *
 * http://libusb.org
 * http://www.dreamincode.net/forums/topic/148707-introduction-to-using-
     libusb-10/
 * http://www.usb.org/developers/devclass_docs/HID1_11.pdf
 * http://www.usb.org/developers/devclass_docs/Hut1_11.pdf
 */

/*
 * Find and return a USB keyboard device or NULL if not found
 * The argument con
 *
 */
struct libusb_device_handle *openkeyboard(uint8_t *endpoint_address) {
  libusb_device **devs;
  struct libusb_device_handle *keyboard = NULL;
  struct libusb_device_descriptor desc;
  ssize_t num_devs, d;
  uint8_t i, k;

  /* Start the library */
  if ( libusb_init(NULL) < 0 ) {
    fprintf(stderr, "Error: libusb_init failed\n");
    exit(1);
  }

  /* Enumerate all the attached USB devices */
  if ( (num_devs = libusb_get_device_list(NULL, &devs)) < 0 ) {
    fprintf(stderr, "Error: libusb_get_device_list failed\n");
    exit(1);
  }

  /* Look at each device, remembering the first HID device that speaks
     the keyboard protocol */

  for (d = 0 ; d < num_devs ; d++) {
    libusb_device *dev = devs[d];
    if ( libusb_get_device_descriptor(dev, &desc) < 0 ) {
      fprintf(stderr, "Error: libusb_get_device_descriptor failed\n");
      exit(1);
    }

    if (desc.bDeviceClass == LIBUSB_CLASS_PER_INTERFACE) {
      struct libusb_config_descriptor *config;
```

```
      libusb_get_config_descriptor(dev, 0, &config);
      for (i = 0 ; i < config->bNumInterfaces ; i++)
        for ( k = 0 ; k < config->interface[i].num_altsetting ; k++ ) {
          const struct libusb_interface_descriptor *inter =
            config->interface[i].altsetting + k ;
          if ( inter->bInterfaceClass == LIBUSB_CLASS_HID &&
               inter->bInterfaceProtocol == USB_HID_KEYBOARD_PROTOCOL) {
            int r;
            if ((r = libusb_open(dev, &keyboard)) != 0) {
              fprintf(stderr, "Error: libusb_open failed: %d\n", r);
              exit(1);
            }
            if (libusb_kernel_driver_active(keyboard,i))
              libusb_detach_kernel_driver(keyboard, i);
            //libusb_set_auto_detach_kernel_driver(keyboard, i);
            if ((r = libusb_claim_interface(keyboard, i)) != 0) {
              fprintf(stderr, "Error: libusb_claim_interface failed: %d\n", r
                  );
              exit(1);
            }
            *endpoint_address = inter->endpoint[0].bEndpointAddress;
            goto found;
          }
        }
    }
  }

 found:
  libusb_free_device_list(devs, 1);

  return keyboard;
}
```

7 Software/usbkeyboard.h

```
#ifndef _USBKEYBOARD_H
#define _USBKEYBOARD_H

#include <libusb-1.0/libusb.h>

#define USB_HID_KEYBOARD_PROTOCOL 1

/* Modifier bits */
#define USB_LCTRL  (1 << 0)
#define USB_LSHIFT (1 << 1)
#define USB_LALT   (1 << 2)
#define USB_LGUI   (1 << 3)
#define USB_RCTRL  (1 << 4)
#define USB_RSHIFT (1 << 5)
#define USB_RALT   (1 << 6)
#define USB_RGUI   (1 << 7)

struct usb_keyboard_packet {
  uint8_t modifiers;
  uint8_t reserved;
  uint8_t keycode[6];
};

/* Find and open a USB keyboard device.  Argument should point to
   space to store an endpoint address.  Returns NULL if no keyboard
   device was found. */
extern struct libusb_device_handle *openkeyboard(uint8_t *);

#endif
```

8 Software/util.c

```
#include "util.h"

extern int vga_led_fd;
extern screen back;
extern sprite_info ground[3][3];
extern int line_length[3];

void clean ()
{
    back.life_1 = 0;
    back.life_2 = 0;
    back.background_color = 0x0;
    back.choice = CLOUDS;

    sprite_info cleaner;
    cleaner.count = 0;

    int i, j;
    for (i = 0; i < 2; i++)
    {
        cleaner.layer = i;

        for (j = 0; j < 8; j++)
        {
            cleaner.id = j;
            write_info(cleaner, back);
        }
    }

}

void write_sprite(sprite_info sprite)
{
    vga_screen_arg_t screen_game;
    screen_game.sprite = sprite;
    screen_game.option = SPRITE;
    if (ioctl(vga_led_fd, VGA_LED_WRITE_DIGIT, &screen_game))
    {
        perror("ioctl(VGA_LED_WRITE_DIGIT)␣failed");
        return;
    }
}

void write_screen(screen background)
{
    vga_screen_arg_t screen_game;
    screen_game.background = background;
    screen_game.option = BACK;
    if (ioctl(vga_led_fd, VGA_LED_WRITE_DIGIT, &screen_game))
    {
```

```
        perror("ioctl(VGA_LED_WRITE_DIGIT)␣failed");
        return;
    }
}


void write_info(sprite_info sprite, screen background)
{
    vga_screen_arg_t screen_game;
    screen_game.sprite = sprite;
    screen_game.background = background;
    screen_game.option = BOTH;
    if (ioctl(vga_led_fd, VGA_LED_WRITE_DIGIT, &screen_game))
    {
        perror("ioctl(VGA_LED_WRITE_DIGIT)␣failed");
        return;
    }
}

void generate_ground (int line, int row)
{

    // Cleaning row to be used
    if (line_length[row] != -1)
    {
        int j;
        for (j = 0; j < line_length[row]; j++)
        {
            ground[row][j].count = 0;
            write_sprite(ground[row][j]);
        }
    }

    int blocks_chance = rand() % 20;
    // 3 blocks = 50% chance
    // 2 blocks = 40% chance
    // 1 block  = 10% chance

    if (blocks_chance < 10)
    {
        // Checking if it won't surpass the maximum number of sprites allowed
            (8)
        int i;
        int sum = 0;
        for (i = 0; i < 3; i++)
            sum += line_length[i];
        sum -= line_length[row];

        if (sum == 6)
        {
            line_length[row] = 2;
        }
        else
```

```
    {
        line_length[row] = 3;
    }
}
else if (blocks_chance < 18)
{
    line_length[row] = 2;
}
else
{
    line_length[row] = 1;
}

// Defining the ground size, starting point, id and layer
// The starting point is defined depending on how many blocks we
// have per line (line_length[row]) and an offset of 8 pixels on
// each side of the screen is added.  (8 ~ 632)

int j;
int dx = (632 - 8) / line_length[row];         // size of each column
for (j = 0; j < line_length[row]; j++)
{
    ground[row][j].shape = GROUND;
    ground[row][j].pos.y = line;
    ground[row][j].layer = SCENARIO;
    ground[row][j].orientation = RIGHT;

    // Defining the id of each sprite
    if (row != 0)
    {
        ground[row][j].id =
            ground[row - 1][line_length[row - 1] - 1].id + j + 1;
    }
    else
    {
        ground[row][j].id = j;
    }

    int begin = dx * j + 8;                    // begin of curr. column

    // 5 blocks long = 50% chance
    // 4 blocks long = 40% chance
    // 3 blocks long = 10% chance

    int length_chance = rand() % 20;

    if (length_chance < 10)
    {
        ground[row][j].count = 5;
        ground[row][j].pos.x =
            (rand() % (dx - ground[row][j].count)) + begin;

    }
```

```
        else if (length_chance < 18)
        {
            ground[row][j].count = 4;
            ground[row][j].pos.x =
                (rand() % (dx - ground[row][j].count)) + begin;
        }
        else
        {
            ground[row][j].count = 3;
            ground[row][j].pos.x =
                (rand() % (dx - ground[row][j].count)) + begin;
        }
    }
}

void x_translation (character *c, character other)
{
    int collision = 0;

    // Checking collision

    // If sprites have pixels on the same line
    if (((other.pos->y + OFFSET <= c->pos->y + OFFSET) &&
        (other.pos->y + OFFSET >= c->pos->y - OFFSET)) ||
        ((other.pos->y - OFFSET <= c->pos->y + OFFSET) &&
        (other.pos->y - OFFSET >= c->pos->y - OFFSET)))
    {
        // Check collision in case of negative speed
        if (c->vx < 0)
        {
            if (other.pos->x + OFFSET < c->pos->x - OFFSET)
            {
                collision = (other.pos->x + OFFSET >= c->pos->x + c->vx -
                    OFFSET);

                if (collision)
                {
                    c->pos->x = other.pos->x + 2 * OFFSET + 1;
                    return;
                }

            }
        }

        // Check collision in case of positive speed
        if (c->vx > 0)
        {
            if (other.pos->x - OFFSET > c->pos->x + OFFSET)
            {
                collision = (other.pos->x - OFFSET <= c->pos->x + c->vx +
                    OFFSET);

                if (collision)
```

```
                {
                    c->pos->x = other.pos->x - 2 * OFFSET - 1;
                    return;
                }
            }
        }
    }

    // No collision
    c->pos->x += c->vx;

   // Checking borders condition
   if (c->pos->x <= 0)
    c->pos->x = 640 + c->pos->x;
   if (c->pos->x >= 640)
    c->pos->x = c->pos->x - 640;

}

void y_translation (character *c, character other)
{
    int collision = 0;
    int first_collision;

    // Checking collision

    // If the other character has pixels on the same column
    if (((other.pos->x + OFFSET <= c->pos->x + OFFSET) &&
        (other.pos->x + OFFSET >= c->pos->x - OFFSET)) ||
        ((other.pos->x - OFFSET <= c->pos->x + OFFSET) &&
        (other.pos->x - OFFSET >= c->pos->x - OFFSET)))
    {
        // Check collision in case of negative speed (jump)
        if (c->vy < 0)
        {
            if (other.pos->y + OFFSET < c->pos->y - OFFSET)
            {
                collision = (other.pos->y + OFFSET >= c->pos->y + c->vy -
                    OFFSET);

                if (collision)
                {
                    c->pos->y = other.pos->y + 2 * OFFSET + 1;
                    return;
                }

            }
        }

        // Check collision in case of positive speed (fall)
        if (c->vy > 0)
        {
            if (other.pos->y - OFFSET > c->pos->y + OFFSET)
```

```
        {
            collision = (other.pos->y - OFFSET <= c->pos->y + c->vy +
                OFFSET);

            if (collision)
            {
                first_collision = other.pos->y - 2 * OFFSET - 1;
                c->jumping = 0;
            }
        }
    }
}

// Check for platform collision (only fall apply)
if (c->vy > 0)
{
    int i, j;
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < line_length[i]; j++)
        {
            // Checking if it's above one of the platforms
            if ((c->pos->x + OFFSET > ground[i][j].pos.x - OFFSET) &&
                (c->pos->x - OFFSET < ground[i][j].pos.x +
                    2 * ground[i][j].count * OFFSET - OFFSET))
            {
                if (c->pos->y  + OFFSET < ground[i][j].pos.y - OFFSET)
                {
                    int collision2 = (c->pos->y + c->vy + OFFSET >=
                        ground[i][j].pos.y - OFFSET);

                    if (collision2)
                    {
                                c->jumping = 0;

                        int second_collision = ground[i][j].pos.y - 2 *
                            OFFSET - 1;

                        if (collision)
                        {
                            c->pos->y = (first_collision <
                                second_collision) ?
                                first_collision : second_collision;
                            return;
                        }

                        c->pos->y = second_collision;
                                return;
                    }
                }
            }
        }
    }
```

```
    }

    if (collision)
    {
        c->pos->y = first_collision;
        return;
    }

    // No collision
    c->pos->y += c->vy;
    c->jumping = 1;
}
```

9 Software/util.h

```
#ifndef _UTIL_H_
#define _UTIL_H_

#include <stdio.h>
#include <stdlib.h>
#include <sys/ioctl.h>
#include "vga_led.h"
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <time.h>

// Shapes
#define GROUND 0
#define GP_STAND 3
#define GP_JUMP 4
#define GM_STAND 1
#define GM_JUMP 2

// Back Choice
#define COLOR 0
#define CLOUDS 1

// ID for OBJECTS' layer
#define GP_ID 0
#define GM_ID 1
#define PW_ID 2

// Layers
#define SCENARIO 0
#define OBJECTS  1

// Orientation
#define RIGHT 0
#define LEFT  1

// Offset from center
#define OFFSET 16

// Options for write
#define SPRITE 1
#define BACK 2
#define BOTH 3

/* Struct to facilitate control of sprites on software */
typedef struct
{
    coordinate *pos;
```

```
    int id;
    int vy;
    int vx;
    int jumping;
    int speed;
}character;

/* Cleans the screen by erasing every sprite */
void clean();

/* Sends information to the driver through the ioctl call */
void write_info(sprite_info, screen);

/* Sends a sprite to the driver */
void write_sprite(sprite_info);

/* Sends background information to the driver */
void write_screen(screen);

/* Generates platforms in the given line, allocating the memory according to
   the given row */
void generate_ground (int, int);

/* Try to move the character received by parameter in the x axis handling the
   collision if necessary */
void x_translation (character *, character);

/* Try to move the character received by parameter in the y axis handling the
   collision if necessary */
void y_translation (character *, character);

#endif
```

10 Software/vga_led.c

```
/*
 * Device driver for the VGA LED Emulator
 *
 * References:
 * http://www.linuxforu.com/tag/linux-device-drivers/
 *
 */

#include <linux/module.h>
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/types.h>
#include <linux/kdev_t.h>
#include <linux/device.h>
#include <linux/fs.h>
#include <linux/cdev.h>
#include <linux/uaccess.h>
#include <asm/io.h>

#include "vga_led.h"
#include "socal/hps.h"
#include "hps_0.h"

/* Memory region visible through the lightweight HPS-FPGA bridge */
#define HW_REGS_BASE ALT_LWFPGASLVS_OFST
#define HW_REGS_SIZE 0x200000
#define HW_REGS_MASK (HW_REGS_SIZE - 1)

#define WRITE_BYTE(addr, val)  ( *(volatile unsigned int *)(addr) = (val))

static void __iomem *registers; /* Base of mapped memory */
static dev_t firstdev;
static struct class *cl;
static struct cdev c_dev;

static void *screen_registers; /* Start of registers for LEDs */

/* Current center state, since the hardware registers can't be read */
static sprite_info sp_info;
static screen background_info;



static void write_screen_sprite(sprite_info sprite) {
    unsigned int spr;
    if (sprite.layer ==  0){
      spr = (((unsigned int)sprite.pos.x) & 0x3ff) |
         ((((unsigned int)sprite.pos.y) & 0x3ff) << 10) |
         ((((unsigned int)sprite.orientation) & 0x1) << 20) |
         ((((unsigned int)sprite.count) & 0x7) << 21) |
```

```
          ((((unsigned int)sprite.id) & 0xf) << 24) |
          ((((unsigned int)sprite.shape) & 0x7) << 28) |
          ((((unsigned int)sprite.layer) & 0x1) << 31);
    }
    else{
      spr = (((unsigned int)sprite.pos.x) & 0x3ff) |
          ((((unsigned int)sprite.pos.y) & 0x3ff) << 10) |
          ((((unsigned int)sprite.orientation) & 0x1) << 20) |
          ((((unsigned int)sprite.count) & 0x7) << 21) |
          ((((unsigned int)sprite.id) & 0x7) << 24) |
          ((((unsigned int)sprite.shape) & 0xf) << 27) |
          ((((unsigned int)sprite.layer) & 0x1) << 31);
    }


        WRITE_BYTE(screen_registers + 0, spr);
        sp_info = sprite;
}

static void write_screen_back(screen background) {
    unsigned int bgnd;


    bgnd =  (((unsigned int)background.life_1) & 0x7) |
           ((((unsigned int)background.life_2) & 0x7) << 3) |
           ((((unsigned int)background.choice) & 0x1) << 6) |
           (((background.background_color) & 0xffffff) << 8);

    WRITE_BYTE(screen_registers + 4, bgnd);
    background_info = background;
}



static int my_open(struct inode *i, struct file *f) {
  return 0;
}

static int my_close(struct inode *i, struct file *f) {
  return 0;
}

/* Handle ioctls(): write to the display registers or read our state */
static long my_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
{
  vga_screen_arg_t vla;

  switch (cmd) {
  case VGA_LED_WRITE_DIGIT:
    if (copy_from_user(&vla, (vga_screen_arg_t *) arg, sizeof(
        vga_screen_arg_t)))
      return -EACCES;
```

```
    if(vla.option & 0x1)
      write_screen_sprite(vla.sprite);
    if(vla.option & 0x2)
      write_screen_back(vla.background);

    break;

  case VGA_LED_READ_DIGIT:
    if (copy_from_user(&vla, (vga_screen_arg_t *) arg, sizeof(
        vga_screen_arg_t)))
      return -EACCES;
    vla.sprite = sp_info;
    vla.background = background_info;
    if (copy_to_user((vga_screen_arg_t *) arg, &vla, sizeof(vga_screen_arg_t)
        ))
      return -EACCES;
    break;

  default:
    return -EINVAL;
  }

  return 0;
}

static struct file_operations my_fops = {
  .owner = THIS_MODULE,
  .open = my_open,
  .release = my_close,
  .unlocked_ioctl = my_ioctl
};

/* Initialize the driver: map the hardware registers, register the
 * device and our operations, and display a welcome message */
static int __init vga_led_init(void) {

  printk(KERN_INFO "vga_led: init\n");

  if ( (registers = ioremap(HW_REGS_BASE, HW_REGS_SIZE)) == NULL ) {
    printk(KERN_ERR "vga_led: Mapping hardware registers failed\n");
    return -1;
  }

  screen_registers = registers +
    ((unsigned long) VGA_LED_0_BASE & (unsigned long) HW_REGS_MASK);

  if (alloc_chrdev_region(&firstdev, 0, 1, "vgaled") < 0) goto unmap;
  if ((cl = class_create(THIS_MODULE, "chardrv")) == NULL) goto unregister;
  if (device_create(cl, NULL, firstdev, NULL, "vgaled") == NULL) goto
      del_class;
  cdev_init(&c_dev, &my_fops);
  if (cdev_add(&c_dev, firstdev, 1) == -1) goto del_device;
```

```
  /* Display a welcome message */
 //write_center(320,240);
  return 0;

  /* Clean up if something went wrong */
 unmap:       iounmap(registers);
 del_device:  device_destroy(cl, firstdev);
 del_class:   class_destroy(cl);
 unregister:  unregister_chrdev_region(firstdev, 1);
  return -1;
}

/* Disable the driver; undo the effects of the initialization routine */
static void __exit vga_led_exit(void) {
  printk(KERN_INFO "vga_led: exit\n");

  cdev_del(&c_dev);
  device_destroy(cl, firstdev);
  class_destroy(cl);
  unregister_chrdev_region(firstdev, 1);
  iounmap(registers);
}

module_init(vga_led_init);
module_exit(vga_led_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Stephen A. Edwards, Columbia University");
MODULE_DESCRIPTION("VGA 7-segment LED Emulator");
```

11 Software/vga_led.h

```
/*
 *
 * Lab 3
 *
 * Name: Bernardo de Almeida Abreu
 * UNI:  bd2440
 *
 * Name: Henrique Pizzol Grando
 * UNI:  hp2409
 *
 * Name: Lucas Ikenaga Barros
 * UNI:  li2176
 *
 * Name: Tomas Mantelato
 * UNI:  tm2779
 *
 */

#ifndef _VGA_LED_H
#define _VGA_LED_H

#include <linux/ioctl.h>

typedef struct
{
  short x, y;
}coordinate;

typedef struct
{
    coordinate pos;
    unsigned char shape;
    unsigned char orientation;
    unsigned char count;
    unsigned char id;
    unsigned char layer;
} sprite_info;

typedef struct
{
        unsigned char life_1, life_2;
    unsigned char choice;
        unsigned int background_color;
} screen;

typedef struct
{
        sprite_info sprite;
        screen background;
        char option;
```

```
} vga_screen_arg_t;


/* ioctls and their arguments */
#define VGA_LED_WRITE_DIGIT _IOW('q', 1, vga_screen_arg_t *)
#define VGA_LED_READ_DIGIT  _IOR('q', 2, vga_screen_arg_t *)

#endif
```