EC551 Advanced Digital Design with Verilog and FPGA (Fall 2022)

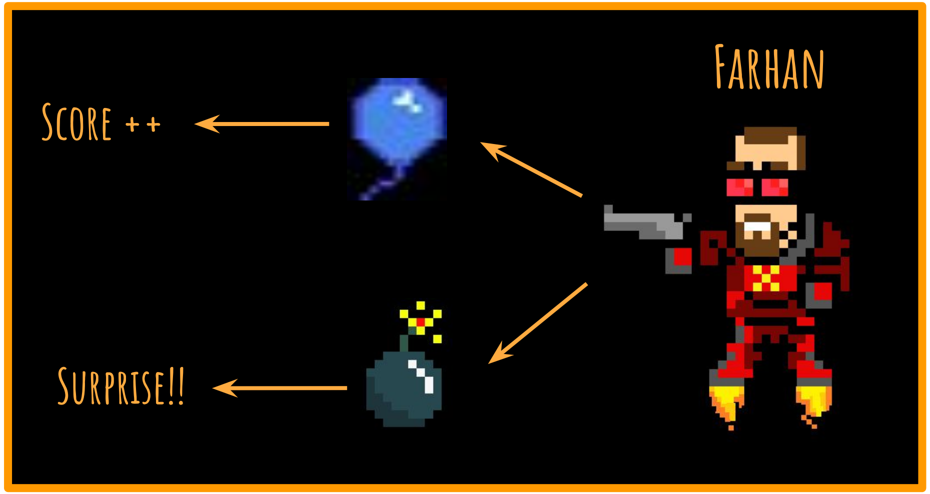
Bombs & Balloons

Team: Fried Chips

Chathura Rajapaksha
Farhan Tanvir Khan
Farbin Fayza

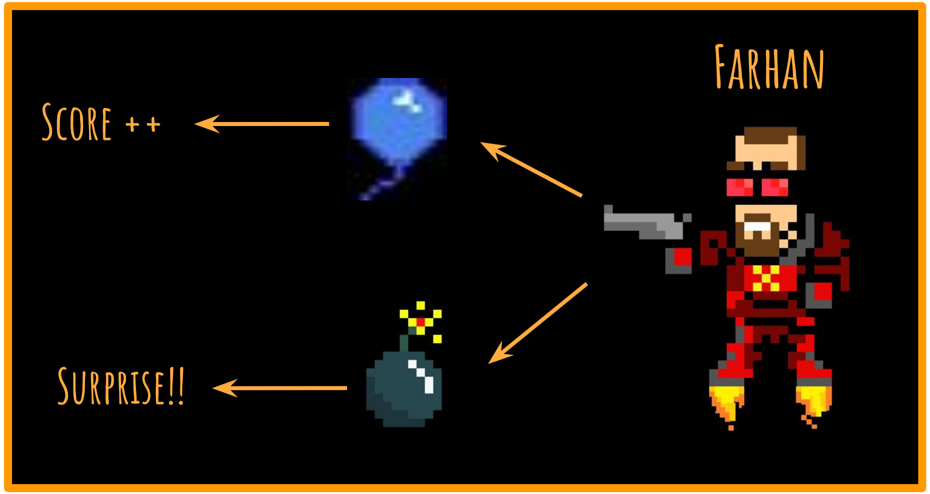
Goal, Short Specification & Functionality

- Goal:
  - Implement a fun game with FPGA
- Game Components:
  - Character
    - Moves up & down
    - Shoots
  - Balloons
    - Spawn randomly
    - Rise upwards
  - Bombs
    - Spawn randomly
    - Fall downwards



Goal, Short Specification & Functionality

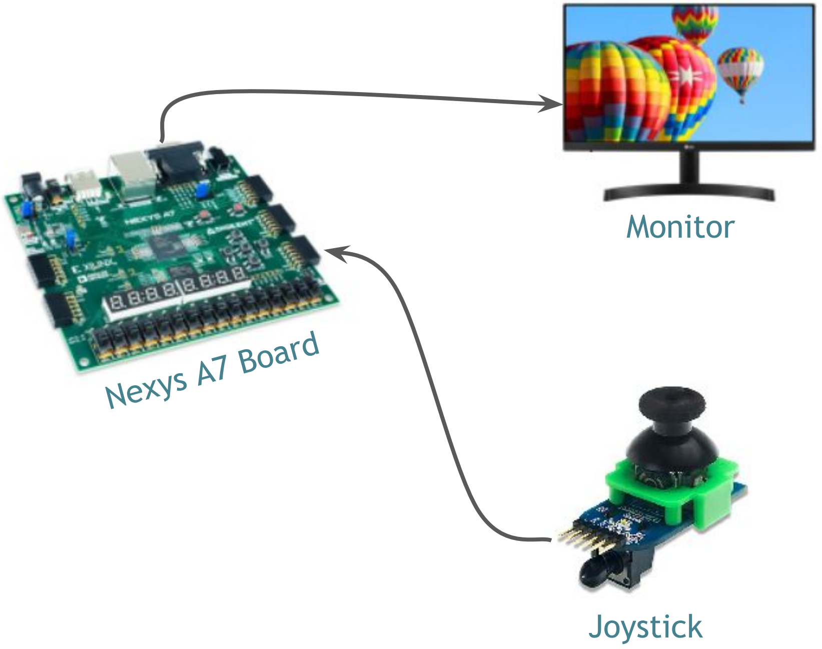
- Game Rules
  - 60 seconds time limit
  - Shooting balloons increments the score
  - Shooting bombs = !!!



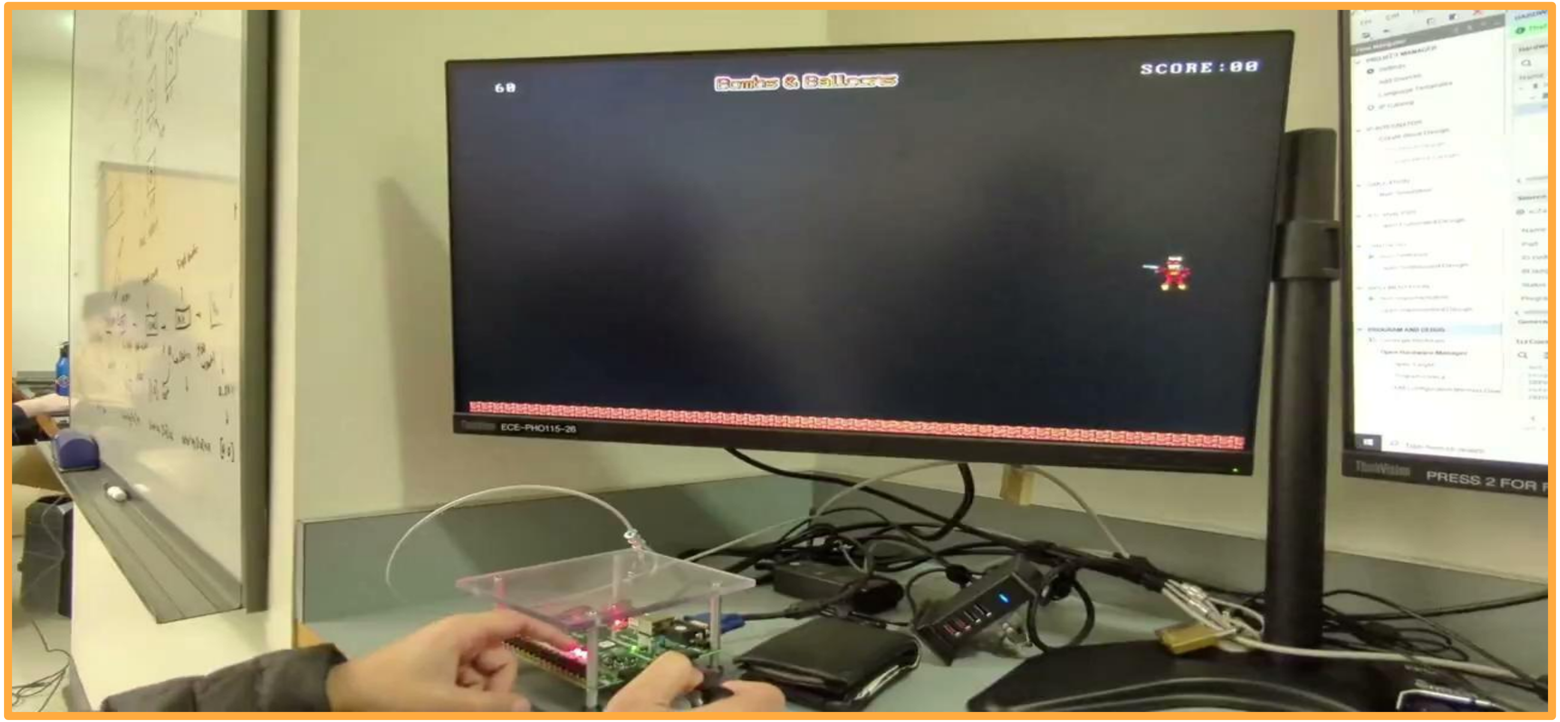
Goal, Short Specification & Functionality

- Game Control
  - Monitor for display
  - Joystick for game control
    - Y-axis up and down
    - Button to shoot

  - Optionally, Nexys A7 onboard buttons for game control
    - Helped us to work together at the same time



Demo for Detailed Functionality

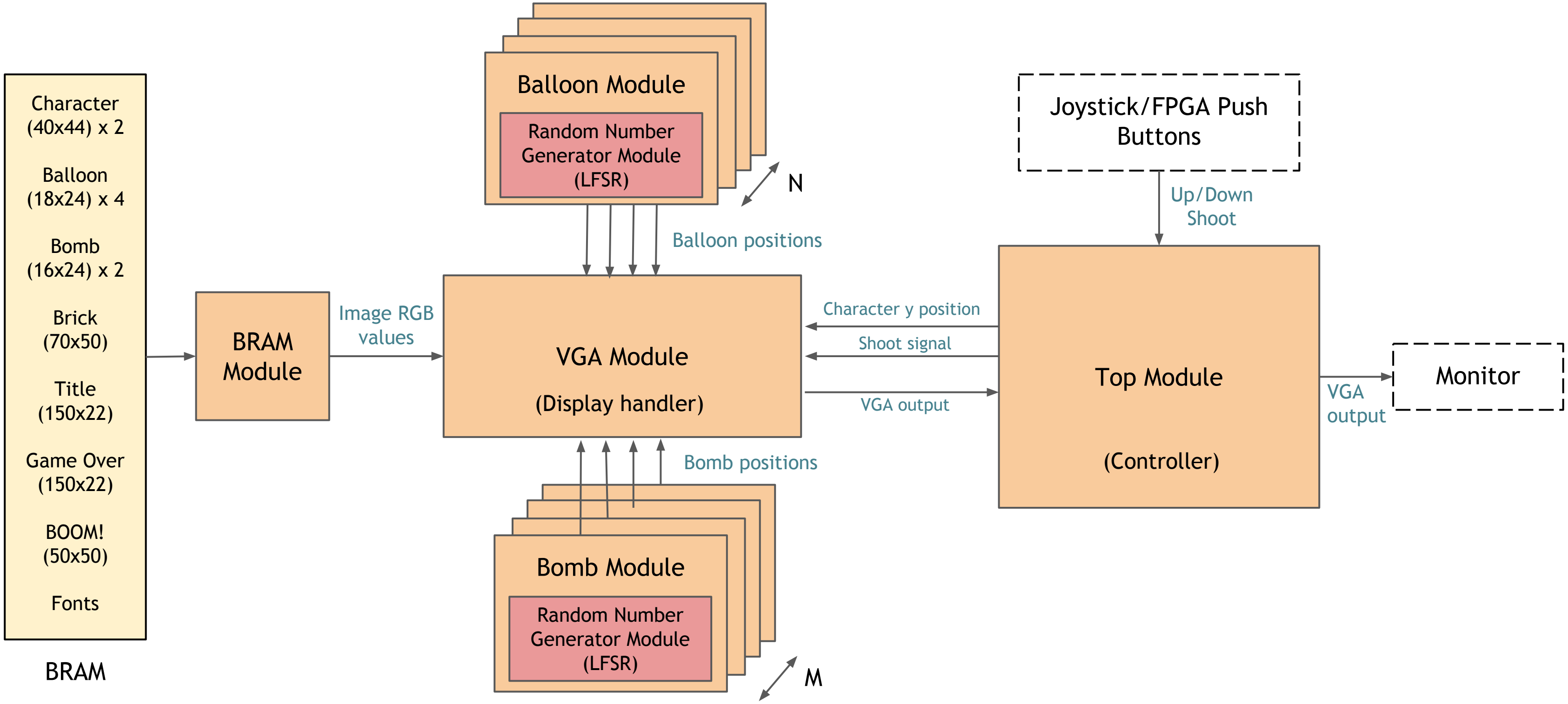


Video: 60 Seconds Plain Game

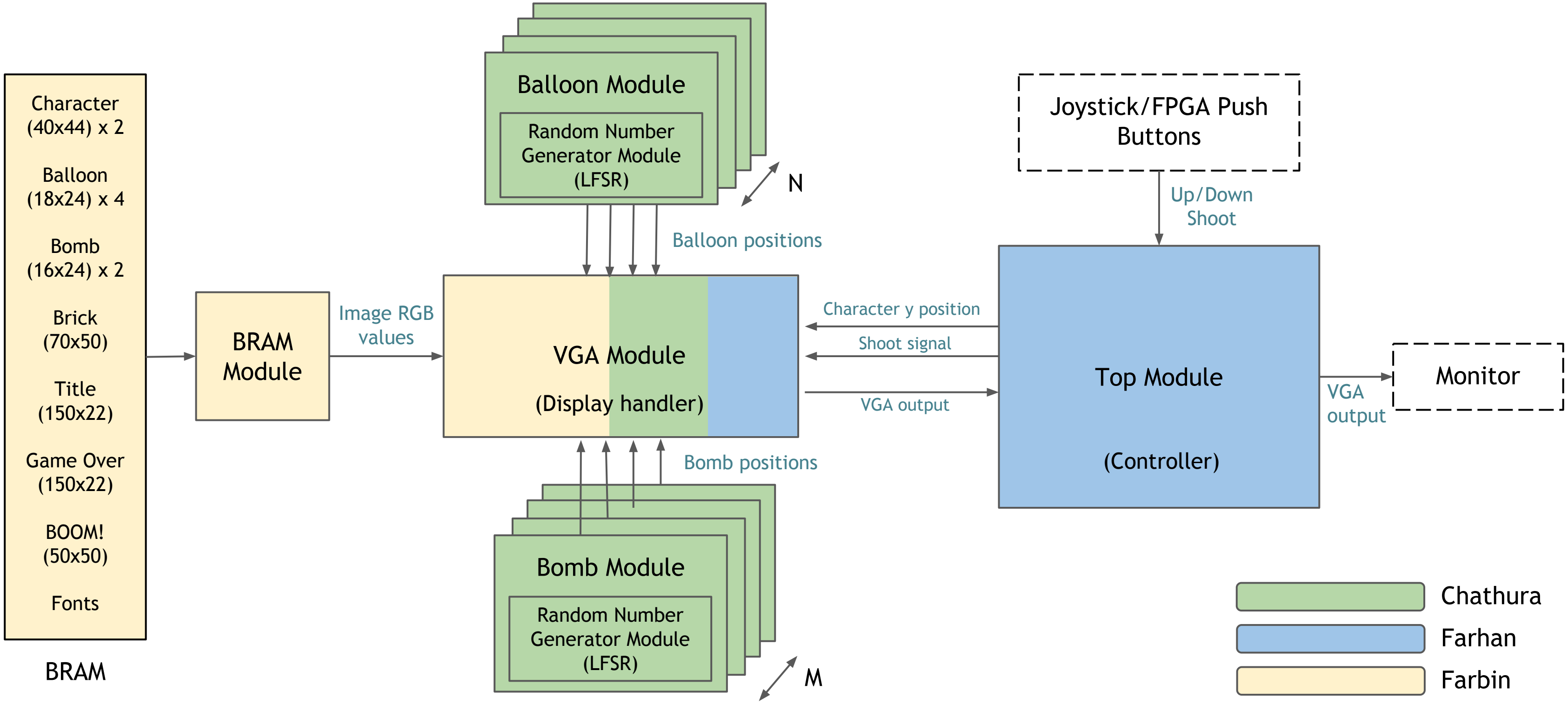
Demo for Detailed Functionality

Video: The Bombs!

Detailed Block Diagram

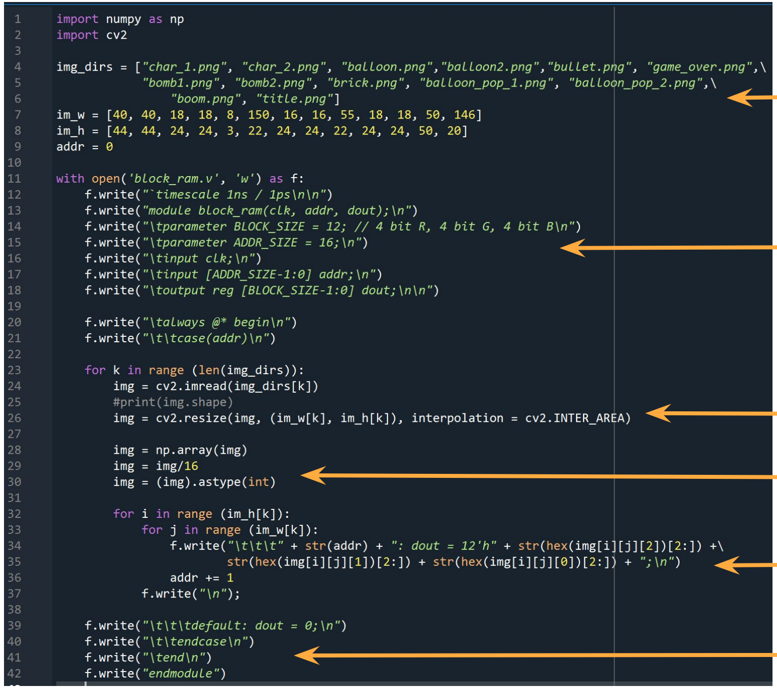


Detailed Block Diagram Showing Individual Contribution



BRAM Module

BRAM module generated with a python script



```python
import numpy as np
import cv2

img_dirs = ["char_1.png", "char_2.png", "balloon.png","balloon2.png","bullet.png", "game_over.png",\
            "bomb1.png", "bomb2.png", "brick.png", "balloon_pop_1.png", "balloon_pop_2.png",\
                "boom.png", "title.png"]
im_w = [40, 40, 18, 18, 8, 150, 16, 16, 55, 18, 18, 50, 146]
im_h = [44, 44, 24, 24, 3, 22, 24, 24, 22, 24, 24, 50, 20]
addr = 0

with open('block_ram.v', 'w') as f:
    f.write("`timescale 1ns / 1ps\n\n")
    f.write("module block_ram(clk, addr, dout);\n")
    f.write("\tparameter BLOCK_SIZE = 12; // 4 bit R, 4 bit G, 4 bit B\n")
    f.write("\tparameter ADDR_SIZE = 16;\n")
    f.write("\tinput clk;\n")
    f.write("\tinput [ADDR_SIZE-1:0] addr;\n")
    f.write("\toutput reg [BLOCK_SIZE-1:0] dout;\n\n")

    f.write("\talways @* begin\n")
    f.write("\t\tcase(addr)\n")

    for k in range (len(img_dirs)):
        img = cv2.imread(img_dirs[k])
        #print(img.shape)
        img = cv2.resize(img, (im_w[k], im_h[k]), interpolation = cv2.INTER_AREA)

        img = np.array(img)
        img = img/16
        img = (img).astype(int)

        for i in range (im_h[k]):
            for j in range (im_w[k]):
                f.write("\t\t\t" + str(addr) + ": dout = 12'h" + str(hex(img[i][j][2])[2:]) +\
                    str(hex(img[i][j][1])[2:]) + str(hex(img[i][j][0])[2:]) + ";\n")
                addr += 1
            f.write("\n");

    f.write("\t\t\tdefault: dout = 0;\n")
    f.write("\t\tendcase\n")
    f.write("\tend\n")
    f.write("endmodule")
```

pixel_converter.py

Specify the image locations, desired widths and heights

Print the initial codes in verilog module

Resize the image

Scale the RGB values to fit in 4 bits (Here, max pixel value = 256)

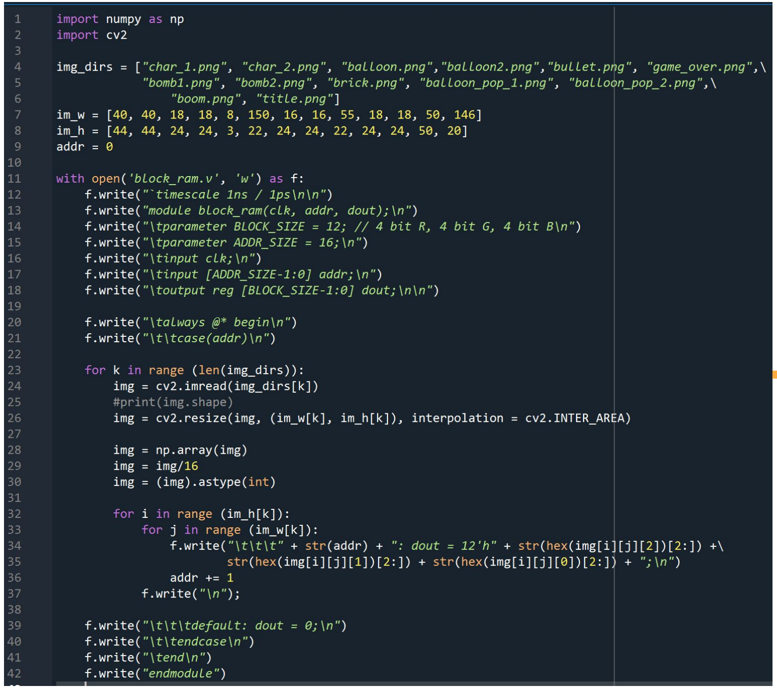
Print the pixel address and the RGB values in hex digits

Print the ending verilog codes

bombs-n-balloons/sprites_bram_generator/pixel_converter.py

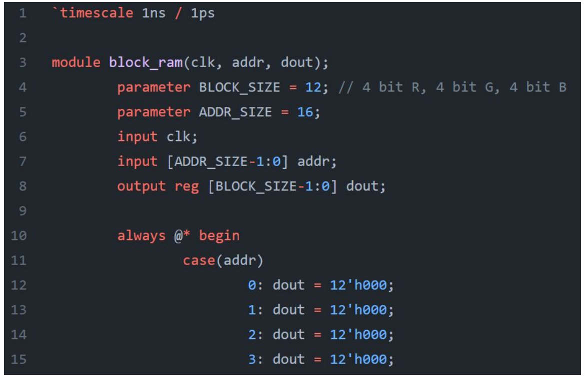
BRAM Module

BRAM module generated with a python script



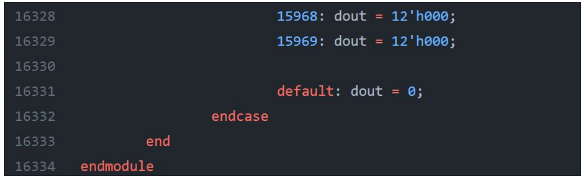
```python
import numpy as np
import cv2

img_dirs = ["char_1.png", "char_2.png", "balloon.png","balloon2.png","bullet.png", "game_over.png",\
               "bomb1.png", "bomb2.png", "brick.png", "balloon_pop_1.png", "balloon_pop_2.png",\
                    "boom.png", "title.png"]
im_w = [40, 40, 18, 18, 8, 150, 16, 16, 55, 18, 18, 50, 146]
im_h = [44, 44, 24, 24, 3, 22, 24, 24, 22, 24, 24, 50, 20]
addr = 0

with open('block_ram.v', 'w') as f:
    f.write("`timescale 1ns / 1ps\n\n")
    f.write("module block_ram(clk, addr, dout);\n")
    f.write("\tparameter BLOCK_SIZE = 12; // 4 bit R, 4 bit G, 4 bit B\n")
    f.write("\tparameter ADDR_SIZE = 16;\n")
    f.write("\tinput clk;\n")
    f.write("\tinput [ADDR_SIZE-1:0] addr;\n")
    f.write("\toutput reg [BLOCK_SIZE-1:0] dout;\n\n")

    f.write("\talways @* begin\n")
    f.write("\t\tcase(addr)\n")

    for k in range (len(img_dirs)):
        img = cv2.imread(img_dirs[k])
        #print(img.shape)
        img = cv2.resize(img, (im_w[k], im_h[k]), interpolation = cv2.INTER_AREA)

        img = np.array(img)
        img = img/16
        img = (img).astype(int)

        for i in range (im_h[k]):
            for j in range (im_w[k]):
                f.write("\t\t\t" + str(addr) + ": dout = 12'h" + str(hex(img[i][j][2])[2:]) +\
                        str(hex(img[i][j][1])[2:]) + str(hex(img[i][j][0])[2:]) + ";\n")
                addr += 1
            f.write("\n");

    f.write("\t\t\tdefault: dout = 0;\n")
    f.write("\t\tendcase\n")
    f.write("\tend\n")
    f.write("endmodule")
```

pixel_converter.py



```verilog
`timescale 1ns / 1ps

module block_ram(clk, addr, dout);
        parameter BLOCK_SIZE = 12; // 4 bit R, 4 bit G, 4 bit B
        parameter ADDR_SIZE = 16;
        input clk;
        input [ADDR_SIZE-1:0] addr;
        output reg [BLOCK_SIZE-1:0] dout;

        always @* begin
                case(addr)
                        0: dout = 12'h000;
                        1: dout = 12'h000;
                        2: dout = 12'h000;
                        3: dout = 12'h000;
```

⋮

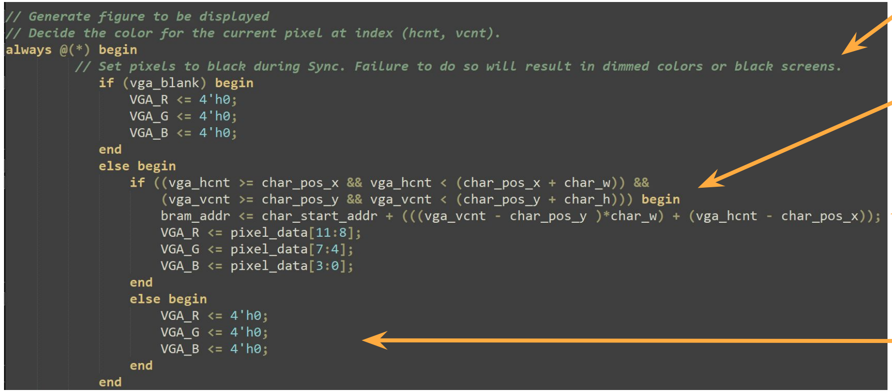


```verilog
                        15968: dout = 12'h000;
                        15969: dout = 12'h000;

                        default: dout = 0;
                endcase
        end
endmodule
```

block_ram.v

VGA Module

- Draws all the sprites and displays texts
- How to draw?
  - Iterate through all the pixels
  - See if the pixel position corresponds to a sprite position
  - Read the color of the pixel from the BRAM
  - Set VGA R, G, B outputs accordingly



```
// Generate figure to be displayed
// Decide the color for the current pixel at index (hcnt, vcnt).
always @(*) begin
         // Set pixels to black during Sync. Failure to do so will result in dimmed colors or black screens.
            if (vga_blank) begin
                VGA_R <= 4'h0;
                VGA_G <= 4'h0;
                VGA_B <= 4'h0;
            end
            else begin
                if ((vga_hcnt >= char_pos_x && vga_hcnt < (char_pos_x + char_w)) &&
                    (vga_vcnt >= char_pos_y && vga_vcnt < (char_pos_y + char_h))) begin
                    bram_addr <= char_start_addr + (((vga_vcnt - char_pos_y )*char_w) + (vga_hcnt - char_pos_x));
                    VGA_R <= pixel_data[11:8];
                    VGA_G <= pixel_data[7:4];
                    VGA_B <= pixel_data[3:0];
                end
                else begin
                    VGA_R <= 4'h0;
                    VGA_G <= 4'h0;
                    VGA_B <= 4'h0;
                end
            end
```

An example of drawing an image (character) in vga.v

Refresh the screen (always set to black)

Check if the sprite (character) needs to be drawn in the current pixel

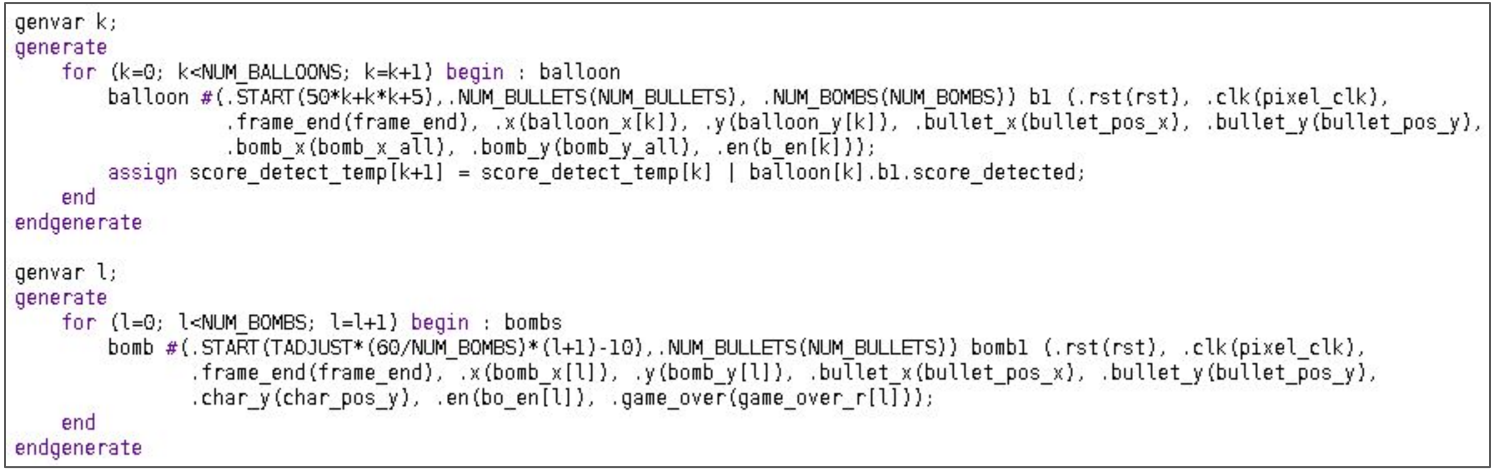
Yes? Then calculate the BRAM address for the pixel of that sprite, and read RGB values (the bram module sets the RGB to pixel_data given the bram_addr)

No? Just draw black pixel

Sample VGA code was given for our lab assignment 1.

Verilog Example

- Generate blocks to make the design scalable

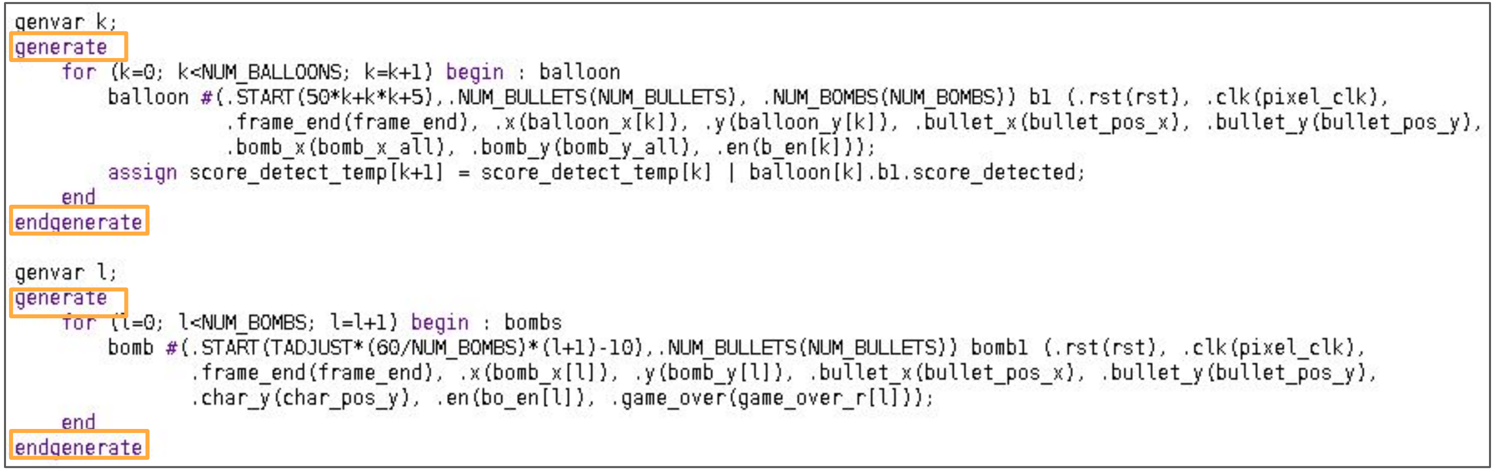


```
genvar k;
generate
    for (k=0; k<NUM_BALLOONS; k=k+1) begin : balloon
        balloon #(.START(50*k+k*k+5),.NUM_BULLETS(NUM_BULLETS), .NUM_BOMBS(NUM_BOMBS)) bl (.rst(rst), .clk(pixel_clk),
                  .frame_end(frame_end), .x(balloon_x[k]), .y(balloon_y[k]), .bullet_x(bullet_pos_x), .bullet_y(bullet_pos_y),
                  .bomb_x(bomb_x_all), .bomb_y(bomb_y_all), .en(b_en[k]));
        assign score_detect_temp[k+1] = score_detect_temp[k] | balloon[k].bl.score_detected;
    end
endgenerate

genvar l;
generate
    for (l=0; l<NUM_BOMBS; l=l+1) begin : bombs
        bomb #(.START(TADJUST*(60/NUM_BOMBS)*(l+1)-10),.NUM_BULLETS(NUM_BULLETS)) bombl (.rst(rst), .clk(pixel_clk),
               .frame_end(frame_end), .x(bomb_x[l]), .y(bomb_y[l]), .bullet_x(bullet_pos_x), .bullet_y(bullet_pos_y),
               .char_y(char_pos_y), .en(bo_en[l]), .game_over(game_over_r[l]));
    end
endgenerate
```

Number of balloons and bombs are parameterized using generate blocks

Verilog Example

- Generate blocks to make the design scalable

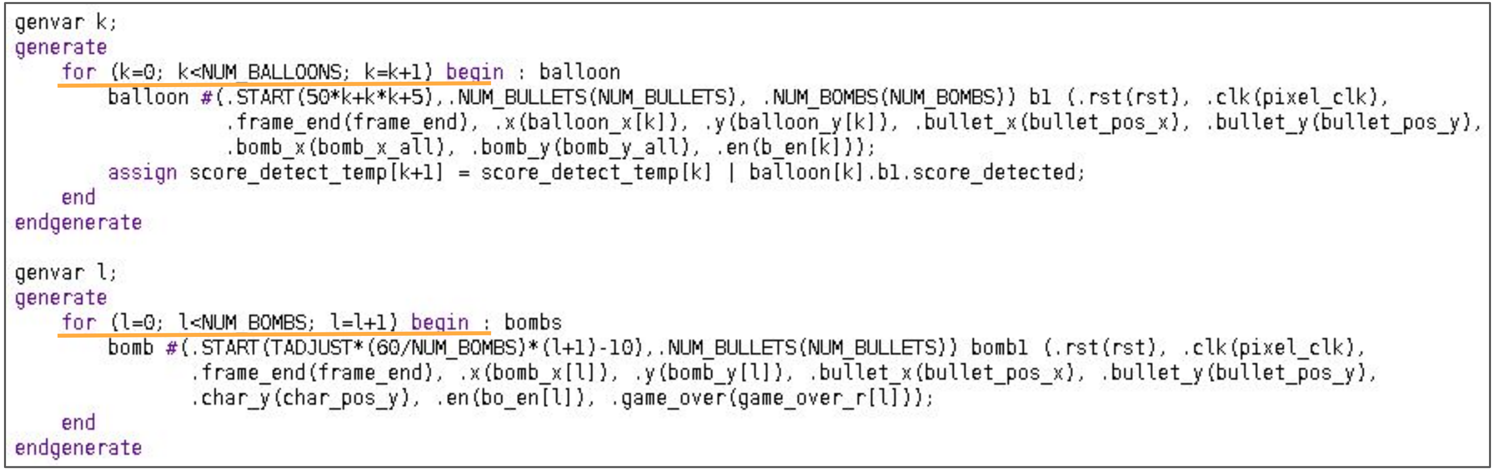


```
genvar k;
generate
    for (k=0; k<NUM_BALLOONS; k=k+1) begin : balloon
        balloon #(.START(50*k+k*k+5),.NUM_BULLETS(NUM_BULLETS), .NUM_BOMBS(NUM_BOMBS)) b1 (.rst(rst), .clk(pixel_clk),
                  .frame_end(frame_end), .x(balloon_x[k]), .y(balloon_y[k]), .bullet_x(bullet_pos_x), .bullet_y(bullet_pos_y),
                  .bomb_x(bomb_x_all), .bomb_y(bomb_y_all), .en(b_en[k]));
        assign score_detect_temp[k+1] = score_detect_temp[k] | balloon[k].b1.score_detected;
    end
endgenerate

genvar l;
generate
    for (l=0; l<NUM_BOMBS; l=l+1) begin : bombs
        bomb #(.START(TADJUST*(60/NUM_BOMBS)*(l+1)-10),.NUM_BULLETS(NUM_BULLETS)) bomb1 (.rst(rst), .clk(pixel_clk),
               .frame_end(frame_end), .x(bomb_x[l]), .y(bomb_y[l]), .bullet_x(bullet_pos_x), .bullet_y(bullet_pos_y),
               .char_y(char_pos_y), .en(bo_en[l]), .game_over(game_over_r[l]));
    end
endgenerate
```

Number of balloons and bombs are parameterized using generate blocks

Verilog Example

- Generate blocks to make the design scalable



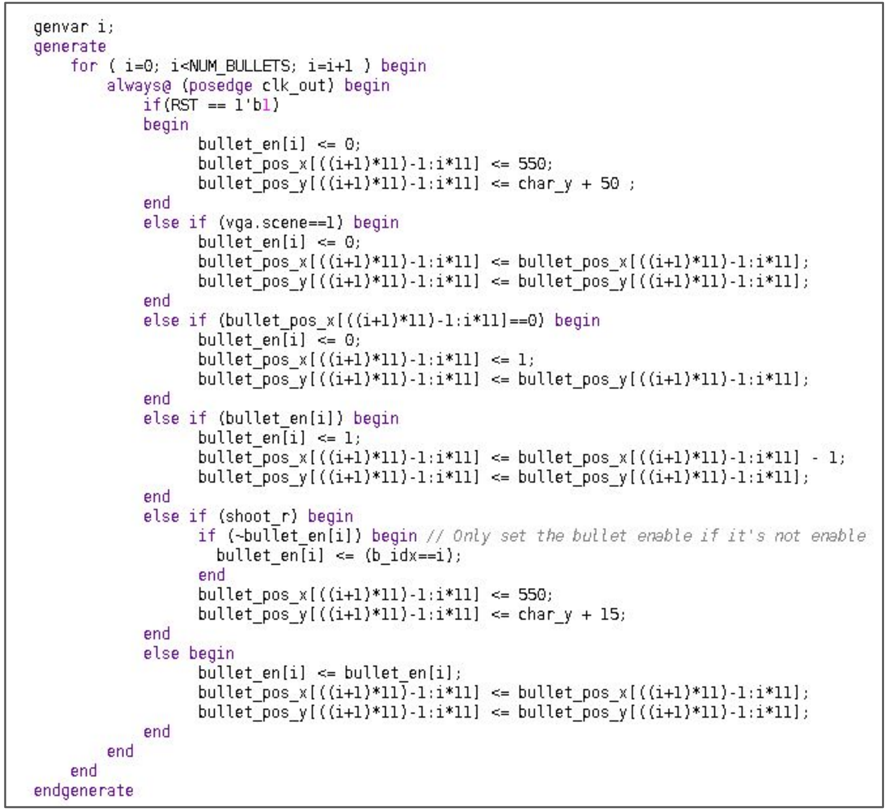
```
genvar k;
generate
    for (k=0; k<NUM_BALLOONS; k=k+1) begin : balloon
        balloon #(.START(50*k+k*k+5),.NUM_BULLETS(NUM_BULLETS), .NUM_BOMBS(NUM_BOMBS)) bl (.rst(rst), .clk(pixel_clk),
                  .frame_end(frame_end), .x(balloon_x[k]), .y(balloon_y[k]), .bullet_x(bullet_pos_x), .bullet_y(bullet_pos_y),
                  .bomb_x(bomb_x_all), .bomb_y(bomb_y_all), .en(b_en[k]));
        assign score_detect_temp[k+1] = score_detect_temp[k] | balloon[k].bl.score_detected;
    end
endgenerate

genvar l;
generate
    for (l=0; l<NUM_BOMBS; l=l+1) begin : bombs
        bomb #(.START(TADJUST*(60/NUM_BOMBS)*(l+1)-10),.NUM_BULLETS(NUM_BULLETS)) bombl (.rst(rst), .clk(pixel_clk),
               .frame_end(frame_end), .x(bomb_x[l]), .y(bomb_y[l]), .bullet_x(bullet_pos_x), .bullet_y(bullet_pos_y),
               .char_y(char_pos_y), .en(bo_en[l]), .game_over(game_over_r[l]));
    end
endgenerate
```

Number of balloons and bombs are parameterized using generate blocks

Verilog Example

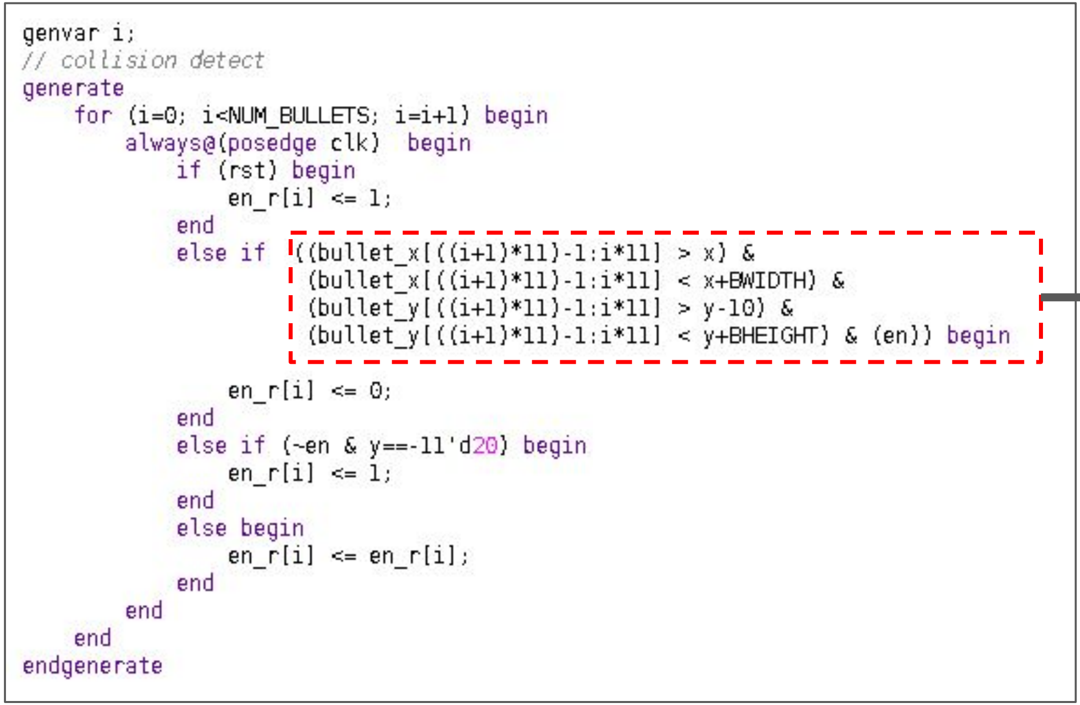
- Generate blocks to make the design scalable

  Enabling multiple bullets on the screen at the same time in a scalable manner



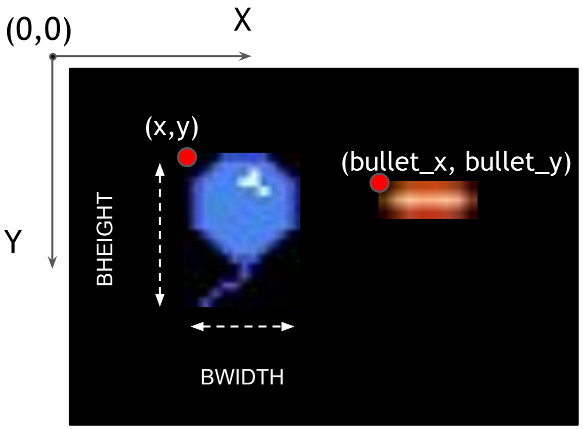
```verilog
genvar i;
generate
    for ( i=0; i<NUM_BULLETS; i=i+1 ) begin
        always@ (posedge clk_out) begin
            if(RST == 1'b1)
            begin
                  bullet_en[i] <= 0;
                  bullet_pos_x[((i+1)*11)-1:i*11] <= 550;
                  bullet_pos_y[((i+1)*11)-1:i*11] <= char_y + 50 ;
            end
            else if (vga.scene==1) begin
                  bullet_en[i] <= 0;
                  bullet_pos_x[((i+1)*11)-1:i*11] <= bullet_pos_x[((i+1)*11)-1:i*11];
                  bullet_pos_y[((i+1)*11)-1:i*11] <= bullet_pos_y[((i+1)*11)-1:i*11];
            end
            else if (bullet_pos_x[((i+1)*11)-1:i*11]==0) begin
                  bullet_en[i] <= 0;
                  bullet_pos_x[((i+1)*11)-1:i*11] <= 1;
                  bullet_pos_y[((i+1)*11)-1:i*11] <= bullet_pos_y[((i+1)*11)-1:i*11];
            end
            else if (bullet_en[i]) begin
                  bullet_en[i] <= 1;
                  bullet_pos_x[((i+1)*11)-1:i*11] <= bullet_pos_x[((i+1)*11)-1:i*11] - 1;
                  bullet_pos_y[((i+1)*11)-1:i*11] <= bullet_pos_y[((i+1)*11)-1:i*11];
            end
            else if (shoot_r) begin
                  if (~bullet_en[i]) begin // Only set the bullet enable if it's not enable
                    bullet_en[i] <= (b_idx==i);
                  end
                  bullet_pos_x[((i+1)*11)-1:i*11] <= 550;
                  bullet_pos_y[((i+1)*11)-1:i*11] <= char_y + 15;
            end
            else begin
                  bullet_en[i] <= bullet_en[i];
                  bullet_pos_x[((i+1)*11)-1:i*11] <= bullet_pos_x[((i+1)*11)-1:i*11];
                  bullet_pos_y[((i+1)*11)-1:i*11] <= bullet_pos_y[((i+1)*11)-1:i*11];
            end
        end
    end
endgenerate
```

Collision Detection Implementation



```
genvar i;
// collision detect
generate
    for (i=0; i<NUM_BULLETS; i=i+1) begin
        always@(posedge clk)  begin
            if (rst) begin
                en_r[i] <= 1;
            end
            else if ((bullet_x[((i+1)*11)-1:i*11] > x) &
                     (bullet_x[((i+1)*11)-1:i*11] < x+BWIDTH) &
                     (bullet_y[((i+1)*11)-1:i*11] > y-10) &
                     (bullet_y[((i+1)*11)-1:i*11] < y+BHEIGHT) & (en)) begin
                en_r[i] <= 0;
            end
            else if (~en & y==-11'd20) begin
                en_r[i] <= 1;
            end
            else begin
                en_r[i] <= en_r[i];
            end
        end
    end
endgenerate
```

Collision detection of a balloon with a bullet.
Defined in balloon module in balloon.v file.



480x640 resolution screen

Successes

- Finished making the game on time
  - All the functionalities proposed in the project proposal are implemented correctly
- Successfully used the sprites
  - Did not run out of memory (18% utilization)
  - Multiple use of low resolution sprites
- Animation of character, bombs, and balloons
- Balloon popping animation
  - We worked till 1 AM in the lab!
- Crazy bomb physics successfully implemented
- Hiding the non-transparent background of the sprites
  - Everything with black background or in rectangular shape
  - Bombs and balloons barely collide due to their physics

- Farhan finally found a title for the game!

Failures & Design Tradeoffs

- Could not add music (We are very sorry)
    - Needed a speaker or converter, ran out of time
    - Focused more on visual improvement
- Little glitches in the sprites at a certain part of the monitor
    - Solved some position-independent glitching problems by analyzing corner cases
    - Still some glitches at a certain position, could not figure out why
- We wanted more images
    - Very long synthesis time for more/larger images
    - Difficult to finish the project with a large BRAM
    - Added all the inactive sprites at the end

Github link: https://github.com/sammy17/bombs-n-balloons