ECE 385

SP 2022

Final project

Final project:
Tank-Stars

Tianyu (Joe) Yu, Yulun (Ben) Wu

5/11/2022

Curtis Yu

1. Introduction

In this final project, we made a game that resembles the style of "Tank Stars": a game that allows two players in the game to control their tank in each round to move and adjust to shoot cannonballs, flying in parabola curves, to the other player's tank. We built our game based on lab 6.2 and added many additional features such as spite drawing, palette, image rotation, and the use of OCM.

2. Game Features and Descriptions

Game Procedure and Logistics:

There are two stages in the game: Tank selection stage and fight stage. In the Tank selection stage of the game, each player can select his type of tank: the game has 3 types of tank, varying in their appearance, power and speed of its cannonball, and the initial HP value. In the fight stage, each player will take turns to move and free fire using a keyboard. The HP value and the time left of each player will be displayed, and the game will be over when one's HP value reaches zero. Players may press the reset button to restart the game.

Graphic Features:

- **Tanks**

Figure 1. Tanks

Above are the three tanks we used (each 100 x 80 in size). The pink (#FF00FF) color is used as transparent color. The rotation of tanks is done by a rotation matrix, not additional tank Sprites.

- **Bullets**

Figure 2. Bullets

Above are the Bullets we used (each 20 x 20 in size). Each bullet corresponds to a type of tank. The pink (#FF00FF) color is used as transparent color.

- **Explosion**

Figure 3. Explosion

This is the explosion when the bullet hits an enemy tank. (20 x 20 in size)

- **Background**

Figure 4. Background

Background (Stored in OCM). Windows xp background with additional explosion.

- **HP bar**

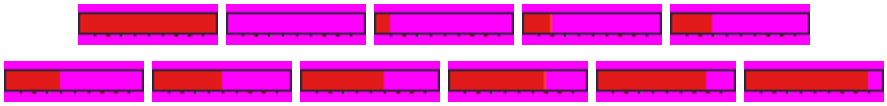

Figure 5. HP Bar

Above are the 11 HP bar sprites we used (From 0 HP to 10 HP). Each 70 x 20 in size The pink (#FF00FF) color is used as transparent color.

- **Count-down numbers**

Figure 6. Count-down numbers

Above are the 10 numbers (each 20 x 30 in size) we used (From 0 to 9) for indicating the time left for the player 1 to take action. Another set of numbers in blue is used to indicate the time left for player 2, this is done by build-in logic.

Key Game Mechanisms:

- Color Extraction

In the project, we applied a 16-colors palette for color drawing by initializing a module for palette in which each color is represented by a 24 bit RGB (8-8-8) value indexed in an unconstrained array. By sending the 4-bits color index to the palette, the color mapper can extract the 24-bits RGB value for the current pixel.

```
module palette(input[3:0] colorIdx,
                                    output[23:0] rgbVal);

        logic[23:0]local_Palette[16];

    assign local_Palette[0] = 24'hFF00FF;
        assign local_Palette[1] = 24'hFFFFFF;
        assign local_Palette[2] = 24'h7A7A7A;
        assign local_Palette[3] = 24'h2D2D2D;

        assign local_Palette[4] = 24'hFF4234;
        assign local_Palette[5] = 24'h1C43DC;
        assign local_Palette[6] = 24'hA31313;
        assign local_Palette[7] = 24'hEFFF6E;

        assign local_Palette[8] = 24'h359FE9;
        assign local_Palette[9] = 24'h136096;
        assign local_Palette[10] = 24'h0C4065;
        assign local_Palette[11] = 24'h96C03d;

        assign local_Palette[12] = 24'h3A572B;
        assign local_Palette[13] = 24'h272233;
        assign local_Palette[14] = 24'hABCCF3;
        assign local_Palette[15] = 24'hE21A1A;


        always_comb begin
            rgbVal = local_Palette[colorIdx];

        end
```

Figure 7. Palette Module

The rom of figures are stored in their rom module as arrays. The size of array is $[0: width \times height - 1][3: 0]$. Each element contains a 4-bits number as the index for the color, and its row-major location corresponds to the actual position of this pixel in the figure.

```
1   module exp_rom ( input [13:0]   addr,
2                                                   output [4:0]   data
3                                               );
4
5           parameter ADDR_WIDTH = 9;
6       parameter DATA_WIDTH =  4;
7           logic [ADDR_WIDTH-1:0] addr_reg;
8
9           // ROM definition
10          //20 x 20 explosion
11
12                  Pixel 0
13
14          parameter [0:399][3:0] EXP = {
15      4'h0,
16      4'h0,                                   Pixel 1
17      4'h0,
18      4'h0,
```

Figure 8. Data Storage of Figures in Roms

To extract the figure's pixel, we need first calculate the row-major address of the pixel within the figure by

$$address = row\ index \times L + column\ index$$
$$= (DrawX - figureX) \times L + (DrawY - figureY)$$

"DrawX" and "DrawY" signals are produced by the vga controller's current drawing position of screen traversal, and "figureX" and "figureY" represent the representative location of the figure, i.e. the coordinate of the upper left pixel .

The storage method for the background picture is a little different due to the memory constraints of a single on chip memory. Instead of having 640×480 resolution rate, we reduce it to 340×240. Since we only have "alt_u8" datatype in NIOS-II C library, we combine each two horizontal pixels' color index information into one element in a C-array. Color mapper will calculate the address of current bit with

$$address = \left(\frac{DrawY}{2} \times 320 + \frac{DrawX}{2}\right) \div 2$$

By sending the address to on-chip memory (declared by IP "background_interface" and initialized value by NIOS-II), we will obtain a 8-bits color index information, and we can extract the first or second 4-bits by checking if the index of the pixel is odd or even.

```
17
18
19  void VGARender()
20  {
21          //This is the function you call for your week 2 demo
22          alt_u8 BG_DATA[38400] = {0x11, 0x11, 0x11, 0x11, 0x11, 0x1
23                                   0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0xee,
24                                   0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0xee,
25                                   0x1e, 0xe1, 0x11, 0x11, 0x11, 0xee, 0xee,
26                                   0xee, 0xee, 0xee, 0xee, 0xee, 0xee, 0xe8,
27                                   0xee, 0xee, 0xee, 0xee, 0x8e, 0xe8, 0x88,
```


Figure 9. Data Storage of Background Picture

- Tank Motion (translational)

The trace of tanks' translational motion is a quadratic function of the tanks' horizontal position, in order to simulate the motion of moving up and down along the hillside. The expression of the parabola is $y = 0.000388x^2 - 0.142x + 267$. When users press the left or right motion key (A/D for tank 1 and J/K for tank 2), the tank will move horizontally one pixel in one frame, and its corresponding vertical motion is determined by the formula above.

Figure 10. The Parabola Which Tanks Move Along

- Tank Motion (rotational)

The tank's rotational motion is designed by angle calculation and linear rotation. To get the rotation angle, we first need to measure the horizontal distance from the upper left and upper right corner of the tank to the ground parabola, and calculate their distance. We applied small angle theorem here, that the rotation angle θ can be approximated by

$$\theta = tan(\theta) = tan(\frac{\Delta h}{L}).$$

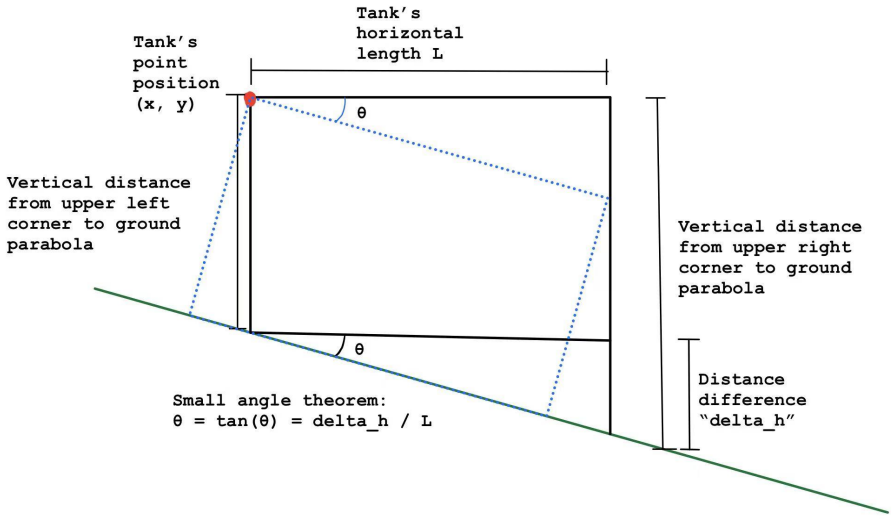

Figure 11. Design Diagram for Tank's Rotation

With θ value, we can apply rotational matrix to obtain the rotated pixel coordinate by

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} = \begin{pmatrix} \frac{1-\tan^2\frac{\theta}{2}}{1+\tan^2\frac{\theta}{2}} & \frac{2\tan\frac{\theta}{2}}{1+\tan^2\frac{\theta}{2}} \\ -\frac{2\tan\frac{\theta}{2}}{1+\tan^2\frac{\theta}{2}} & \frac{1-\tan^2\frac{\theta}{2}}{1+\tan^2\frac{\theta}{2}} \end{pmatrix} = \begin{pmatrix} \frac{1-\frac{\Delta h^2}{4L^2}}{1+\frac{\Delta h^2}{4L^2}} & \frac{2\frac{\Delta h}{2L}}{1+\frac{\Delta h^2}{4L^2}} \\ -\frac{2\frac{\Delta h}{2L}}{1+\frac{\Delta h^2}{4L^2}} & \frac{1-\frac{\Delta h^2}{4L^2}}{1+\frac{\Delta h^2}{4L^2}} \end{pmatrix} = \begin{pmatrix} \frac{4x^2-\Delta y^2}{4x^2+\Delta y^2} & \frac{4x\Delta y}{4x^2+\Delta y^2} \\ -\frac{4x\Delta y}{4x^2+\Delta y^2} & \frac{4x^2-\Delta y^2}{4x^2+\Delta y^2} \end{pmatrix}$$

With new tank coordinate, we can use the row-major method to extract pixel information

- Bullet Motion

Before the bullet is shooted, it is carried on the tank. There is a flag “isCarry” within the bullet module, such that its coordinate moves along with the tank's motion. After the user shoots the bullet, the bullet will move along a parabola, and when the bullet touches the edge of the screen or hits the ground, the “isCarry” flag will rise, letting the bullet stick on the tank again.

After the player presses the shoot key (“SPACE” for player 1 and “ENTER” for player 2), the Motion of the bullet is designed to follow the law of gravity motion. To simulate the effect of gravity acceleration, we increase the vertical step of the bullet by 1 during each frame. The initial vertical step is initialized and changed by the user's aiming control. This will modify the variable “y_component” for the bullet. Modeled as a vector, the horizontal component will reduce by 3 each time the “y_component” is increased by 1.

Figure 12. Trace of Bullet and Explosion

- Hit and HP

When the enemy's bullet position indicates the bullet is within the range of the tank's figure (the inner rectangle of tank tile), the flag "hit_A" or "hit_B" will rise, leading to a series of reactions.

The bullet's tile will first become the tile of explosion in the frame where the bullet's position overlaps with the enemy tank. The bullet's "isCarry" flag will first raise, letting the bullet stick on the tank again.

The HP of the enemy tank will be reduced based on the type of tank that shoots the bullet. The HP bar of each tank will be reflected on the top of the screen.

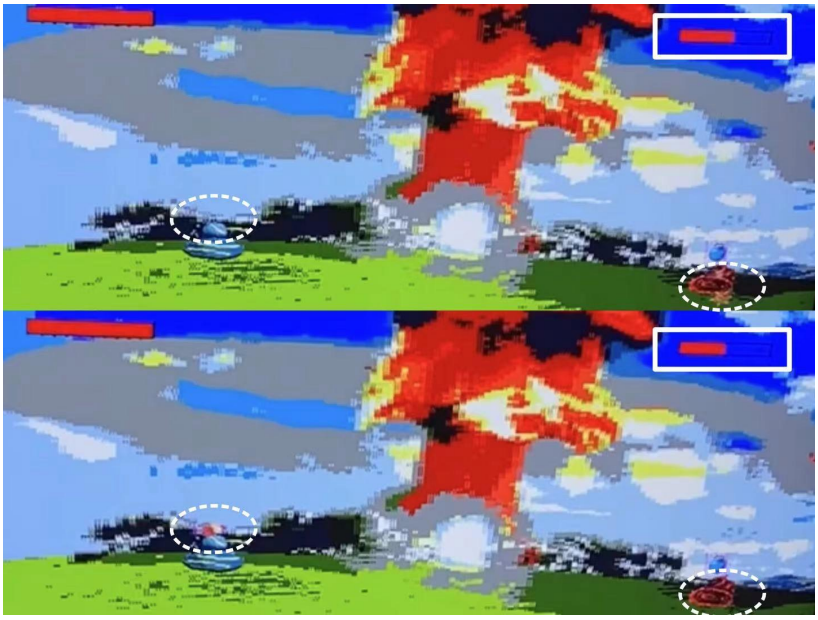

Figure 13. Explosion Effect and HP Value Reaction

- Count-down and "GAMEOVER"

The game is turn-based, each player has 10 seconds to control the tank to move or shoot. The count-down number will be shown in the top center of the screen, and the value changes every 60 frames (aka 1 second). The color counting number varies in 2 players' rounds.

There is a 1-bit "Control_EN" signal that is shared by two tanks. In each 10-seconds round, the signal will flip, allowing a single player to do effective control on their tank, and the other player's keycode will be disabled (set to null value).

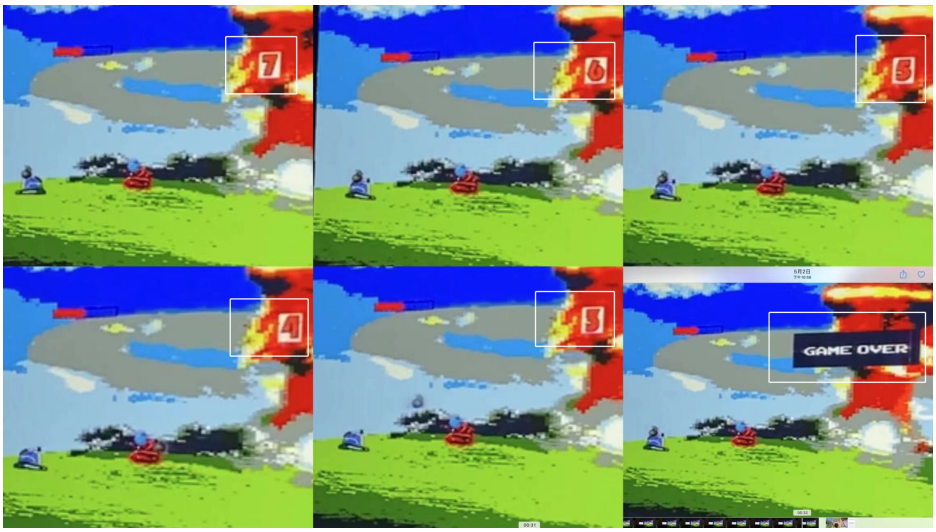

Figure 14. Demonstration of Count-down and “GAMEOVER”

- Keyboard Control

Two players will use different zones on the keyboard to control their tank. Player 1 will use “A/D” to move the tank around, and use “W/S " to adjust the bullet trajectory, and player 2 will use “J/L " and “I/K” to achieve a similar mission. Before shooting the bullet, each player will use “V” or “P” to reload, and use “SPACE” or “ENTER” to shoot

In the tank selection stage, player 1 shall use “Q/E” to change to the previous or next tank he wants to use, and player 2 will use “U/O” to select the tank.

3. Description of Software Part and IP

- IP: background_interface.sv (.tcl)

For the consideration of compilation time and the limited array size in SystemVerilog, we decided to store the background picture layer into on-chip memory: use the interface “.sv” file to declare the on-chip memory, and use NIOS-II to send the data to this memory block through Avalon bus.

```
module background_interface(
        input logic CLK,

        // Avalon Reset Input
        input logic RESET,

        // Avalon-MM Slave Signals
        input  logic [15:0] LOCAL_ADDR,
        input  logic AVL_READ,                                  // Avalon-MM Read
        input  logic AVL_WRITE,                                 // Avalon-MM Write
        input  logic AVL_CS,                                    // Avalon-MM Chip Select
        input  logic [15:0] AVL_ADDR,                   // Avalon-MM Address
        input  logic [7:0] AVL_WRITEDATA,               // Avalon-MM Write Data
        output logic [7:0] AVL_READDATA,                // Avalon-MM Read Data

        // Exported Conduit (mapped to VGA port - make sure you export in Platform Designer)
        output logic [7:0]  backgroundDATA);                                            // VGA HS/VS)



OCM_background ocm_bg(
        .address_a(AVL_ADDR),
        .address_b(LOCAL_ADDR),
        .clock(CLK),
        .data_a(AVL_WRITEDATA),
        .data_b(8'bxxxxxxxx),
        .wren_a(AVL_WRITE),
        .wren_b(1'b0),
        .q_a(AVL_READDATA),
        .q_b(backgroundDATA));
```

Figure 15. SystemVerilog Code for the Interface IP

The code above shows the functionality of the interface IP: it receives value from NIOS-II system through Avalon bus, and it writes the data to the on-chip memory “ocm_bg”. Besides, it also receives data “LOCAL_ADDR” which is the address of the current pixel calculated by the color mapper, and it sends the corresponding color data out in the “backgroundDATA” variable.

- NIOS-II C-code

In the ".h” file, the program declares a struct, and sets its address to 0x20000, which is the place that the platform designer allocates to the interface IP the SDRAM.

```
19  struct BACK_STRUCT {
20          alt_u8 VRAM[38400];
21
22  };
23
24
25
26  //you may have to change this line depending on your platform designer
27  static volatile struct BACK_STRUCT* vga_ctrl = 0x20000;
28
29  //CGA colors with names
30
```

Figure 16. Code in “text_mode_vga_color.h”

There is an array “VRAM” with length 38400 and datatype “alt_u8” for each element. Its mechanism is described in section 2 of this report (Game Features and Descriptions - Key Game Mechanisms - Color Extraction).

In the ".c” file, we have the function “”VGARender()”, which will use a giant for-loop to initialize the on-chip memory with the background data.

```
260                         0x33, 0x33, 0x33, 0x33, 0x33, 0x33, 0x33,
261                         0xdd, 0xdd, 0xdd, 0xdd, 0xdd, 0xdd, 0xdd,
262                         };
263         for (int i = 0; i<38400; i++)
264                 {
265                         vga_ctrl->VRAM[i] = BG_DATA[i];
266                         printf("rendering %x \n", i);
267                 }
268         //memcpy(vga_ctrl->VRAM, BG_DATA, sizeof(BG_DATA));
269         printf("done copying");
270 }
```

Figure 17. Code in “text_mode_vga_color.c”

By copying the local data in array “BG_DATA” to array “VRAM”, the program can then successfully load all essential data to the on-chip memory. After this step, we borrowed the codes from lab 6.2 (Lab: Ball) to use SPI protocol to build the connection with the keyboard. Below is the “main” function in C code.

```
int main() {

    printf("start ''''main");

    printf("start VGA rendering");
    //VGAColorClr();
    VGARender();

    BYTE rcode;
        BOOT_MOUSE_REPORT buf;      //USB mouse report
        BOOT_KBD_REPORT kbdbuf;

        BYTE runningdebugflag = 0;//flag to dump out a bunch of information when we first get to USB_STATE_RUNNING
        BYTE errorflag = 0; //flag once we get an error device so we don't keep dumping out state info
        BYTE device;
        WORD keycode;

        printf("initializing MAX3421E...\n");
        MAX3421E_init();
        printf("initializing USB...\n");
        USB_init();

    printf("finish OCM initialization");
    while (1) {
        printf(".");
        MAX3421E_Task();
        USB_Task();
        //usleep (500000);
        if (GetUsbTaskState() == USB_STATE_RUNNING) {
            if (!runningdebugflag) {
                runningdebugflag = 1;
                setLED(9);
                device = GetDriverandReport();
            } else if (device == 1) {
                //run keyboard debug polling
                rcode = kbdPoll((kbdbuf);
                if (rcode == hrNAK) {
                    continue; //NAK means no new data
                } else if (rcode) {
                    printf("Rcode: ");
                    printf("%x \n", rcode);
                    continue;
                }
                printf("keycodes: ");
```

render the ocm with background picture data

Looping keyboard connection code

Figure 18. Code in “main.c”

4. Block Diagram and RTL Viewer

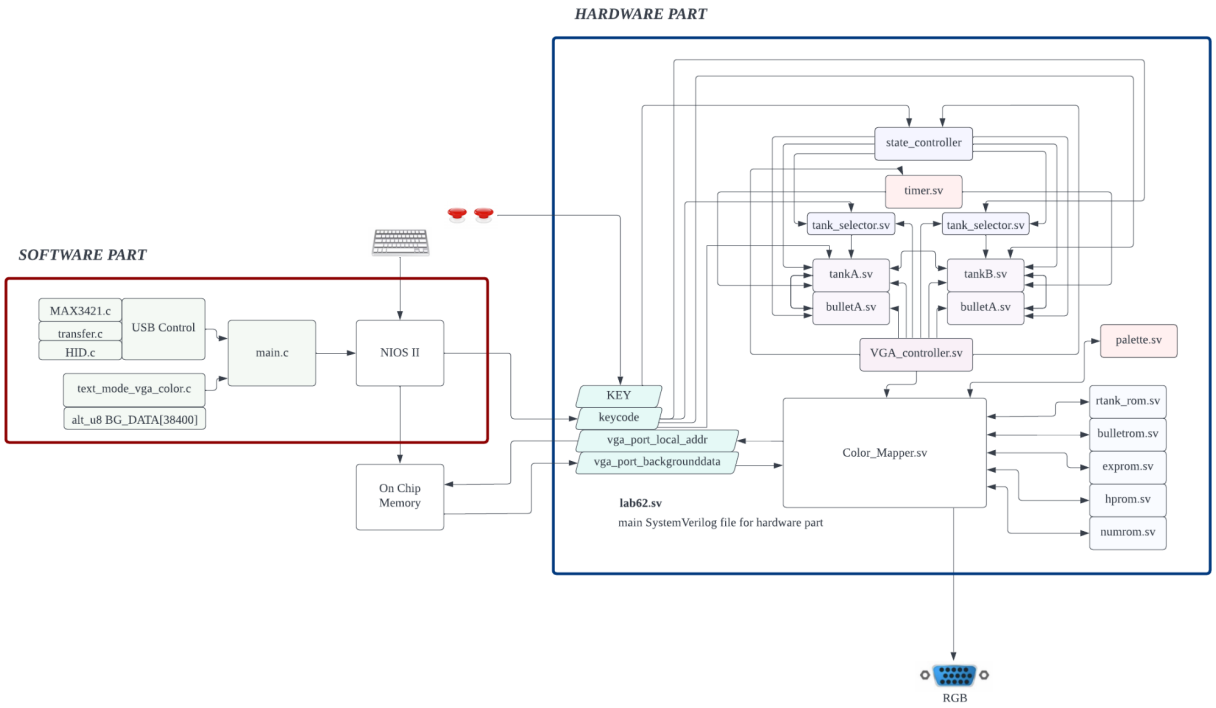

Figure 19. Block Diagram

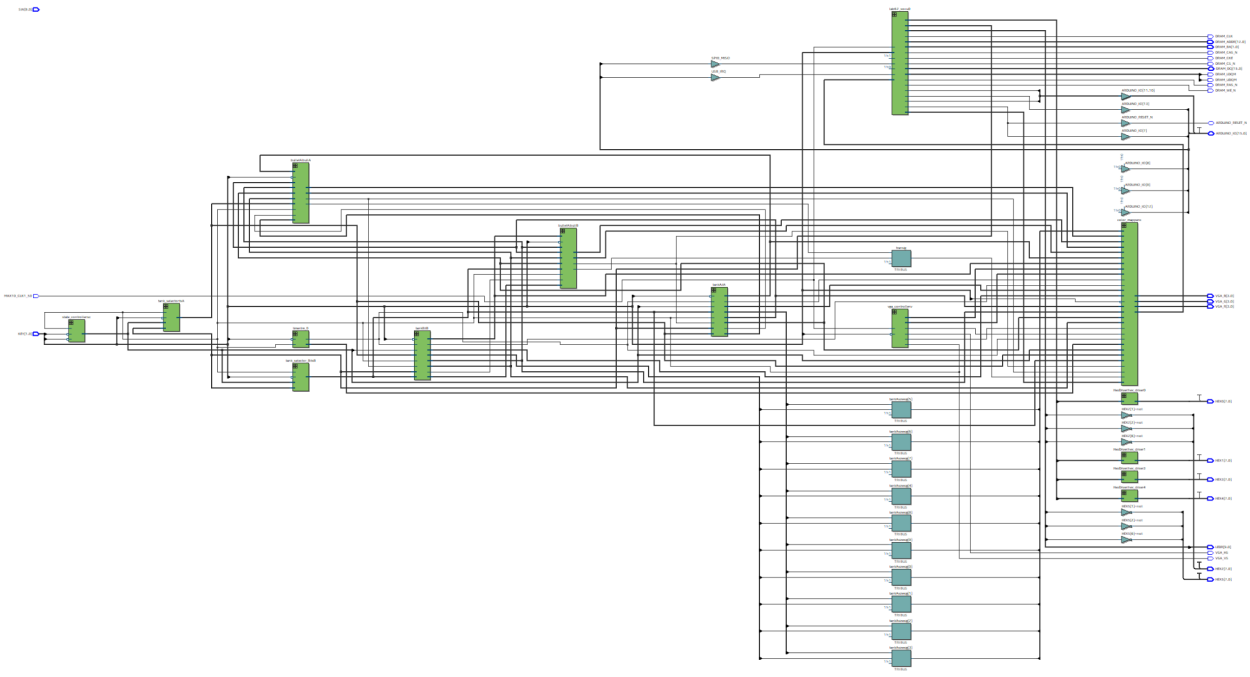

Figure 20. Top Level RTL Viewer

5. State Diagram

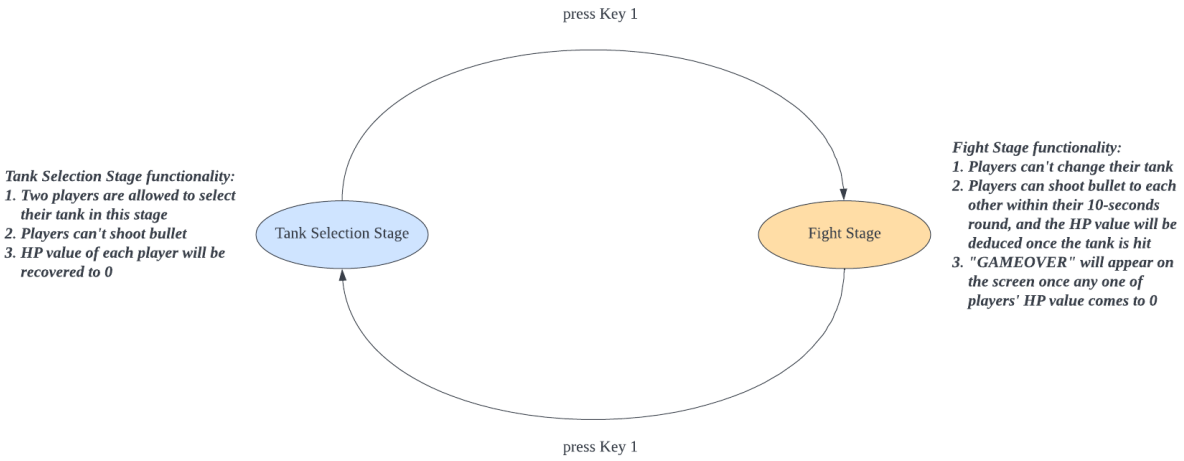

Figure 21. State Diagram

6. Module Descriptions

- **MODULE: Color_Mapper.sv**

Inputs: VGA_Clk, Reset,
[3:0] HP_A, HP_B, currNum,
[9:0] TankX_A, TankY_A, BulletX_A, BulletY_A, DrawX, DrawY, Ball_size, TankX_B, TankY_B, BulletX_B, BulletY_B,
[1:0] Direction_A, Direction_B,
transparent, blank, hit_A, hit_B, GG_A, GG_B, control_EN,
[1:0] currentState, currentTank_A,currentTank_B,
[7:0] vga_port_backgrounddata

Outputs: [15:0] vga_port_local_addr,
logic [7:0] Red, Green, Blue

Description: The color mapper module is responsible for mapping the correct color for display.

Purpose: Based on game status(such as tanks, timer, and background) and user interaction(move or shoot), print the corresponding RGB values which are read from a 16-color palette.

- **MODULE: VGA_controller.sv**

Inputs: clk, Reset

Outputs: hs, vs, pixel_clk, blank, sync,
[9:0] DrawX, DrawY

Description: Scan through every pixel row by row to indicate which pixel is Drawx and Drawy currently on.

Purpose: Generate the Drawx and Drawy which are inputs to color mapper

- **MODULE: background_interface.sv**

Inputs: CLK, RESET, AVL_READ, AVL_WRITE, AVL_CS,
[7:0] AVL_WRITEDATA,
[15:0] AVL_ADDR

Outputs: [7:0] AVL_READDATA, backgroundDATA

Description: Read and write data to OCM which contains the RGB palette information for our backgrounds.

Purpose: Read RGB palette data from OCM.

- **MODULE: bulletA.sv**

Inputs: [9:0] y_component, TankX, TankY, TankX_enemy, TankY_enemy
Fram_clk, Reset, shoot

Outputs: [9:0] BulletX, BulletY,
Hit, transparent

Description: The module calculates the motion and type of both bullets. It is developed from the ball.sv and has many additional features such as gravity and hits.

Purpose: Based on the inputs, calculate the type and position of the bullet.

- **MODULE: bulletrom.sv**

Inputs: [13:0] addr,
[1:0] tankSelection,

Outputs: [3:0] data

Description: A rom that stores the color palette information at each address. Each bullet is 20 by 20 and takes 400 addresses.

Purpose: Read the chosen bullet's color palette information from the rom.

- **MODULE: exprom.sv**

Inputs: [13:0] addr

Outputs: [3:0] data

Description: This rom stores a 20 by 20 explosion color palette information.

Purpose: Read the explosion's color palette information from the rom.

- **MODULE: gameoverrom.sv**

Inputs: [12:0] addr

Outputs: [3:0] data

Description: This rom stores a 150 by 50 "gameover" color palette information.

Purpose: Read the gameover's color palette information from the rom.

- **MODULE: ground_function.sv**

Inputs: [9:0]Tank_X_Pos

Outputs: [9:0]Tank_Y_Pos

Description: Based on the X position of tank, calculate the Y position (X and Y has a parabola relationship)

Purpose: Minic the ground so that the tank can stay over a surface that we designed.

- **MODULE: hprom.sv**

Inputs: [13:0] addr, [3:0] HPSelection

Outputs: [7:0] data

Description: This rom stores eleven 70 by 20 "health bar" color palette information.

Purpose: Read the chosen health bar's color palette information from the rom.

- **MODULE: lab62.sv**

Inputs: MAX10_CLK1_50
[1: 0] KEY
[9: 0] SW
[15:0] ARDUINO_IO
ARDUINO_RESET_N

Outputs: [9: 0] LEDR
[7: 0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5,
DRAM_CLK, DRAM_CKE, DRAM_LDQM, DRAM_UDQM, DRAM_CS_N, DRAM_WE_N, DRAM_CAS_N, DRAM_RAS_N,
[12: 0] DRAM_ADDR,
[1: 0] DRAM_BA ,
[15: 0] DRAM_DQ,
VGA_HS, VGA_VS,
[3: 0] VGA_R, VGA_G, VGA_B,

Description: Top level for Final lab.

Purpose: integrates the Nios II system with the rest of the hardware, and instantiate modules used for this Lab

- **MODULE: numrom.sv**

Inputs: [12:0] addr, [3:0] currNum

Outputs: [3:0] data

Description: This rom stores eleven 20 by 30 “number” color palette information.

Purpose: Read the chosen number’s color palette information from the rom.

- **MODULE: palette.sv**

Inputs: [3:0] colorIdx

Outputs: [23:0] rgbVal

Description: Stores the 24-bits RGB value corresponding to all 16 color palette numbers.

Purpose: Output the 24 bits RGB value (8 for each color) based on the color palette number.

- **MODULE: state_controller.sv**

Inputs: Reset, Clk, nextStateSig

Outputs: [1:0] currentState

Description: Have ff logic that makes sure that the state will switch once when nextStateSig goes high.

Purpose: Switch states when nextStateSig is “pressed”

- **MODULE: tankA.sv**

Inputs: Reset, frame_clk, hit, currentState, control_EN
[1:0] selfTank, enemyTank,
[7:0] keycode,

Outputs: [9:0] TankX, TankY, TankS, y_component
[1:0] Direction,
Shoot, GG_A
[3:0] HP

Description: Based on the inputs, change the position, direction, and condition of the tank.

Purpose: Use inputs to calculate the position and state of the tank

- **MODULE: hprom.sv**

Inputs: Reset, frame_clk, hit, currentState, control_EN
[1:0] selfTank, enemyTank,
[7:0] keycode,

Outputs: [9:0] TankX, TankY, TankS, y_component
[1:0] Direction,
Shoot, GG_B
[3:0] HP

Description: Same as TankA.sv we need separate sv files because GG_A and GG_B refer to different end-results.

Purpose: Same as TankA.sv

- **MODULE: tank_selector.sv**

Inputs: Clk, Reset, [7:0] keycode, [1:0] currentState

Outputs: [1:0] currentTank

Description: If in the selection state, the user will be able to select three different tanks.

Purpose: Allow user to select tank during the selection state

- **MODULE: tank_selector_B.sv**

Inputs: Clk, Reset, [7:0] keycode, [1:0] currentState

Outputs: [1:0] currentTank

Description: Same as tank_selector.sv

Purpose: Same as tank_selector.sv

- **MODULE: timer.sv**

Inputs: frameclk, Reset

Outputs: control_EN, [3:0] currNum

Description: based on the 60 Hz frame clock, calculate the real world time, and which users are allowed to take action.

Purpose: Keep track of real world time which allows the users to take turns to control their tank.

7. Platform Designer

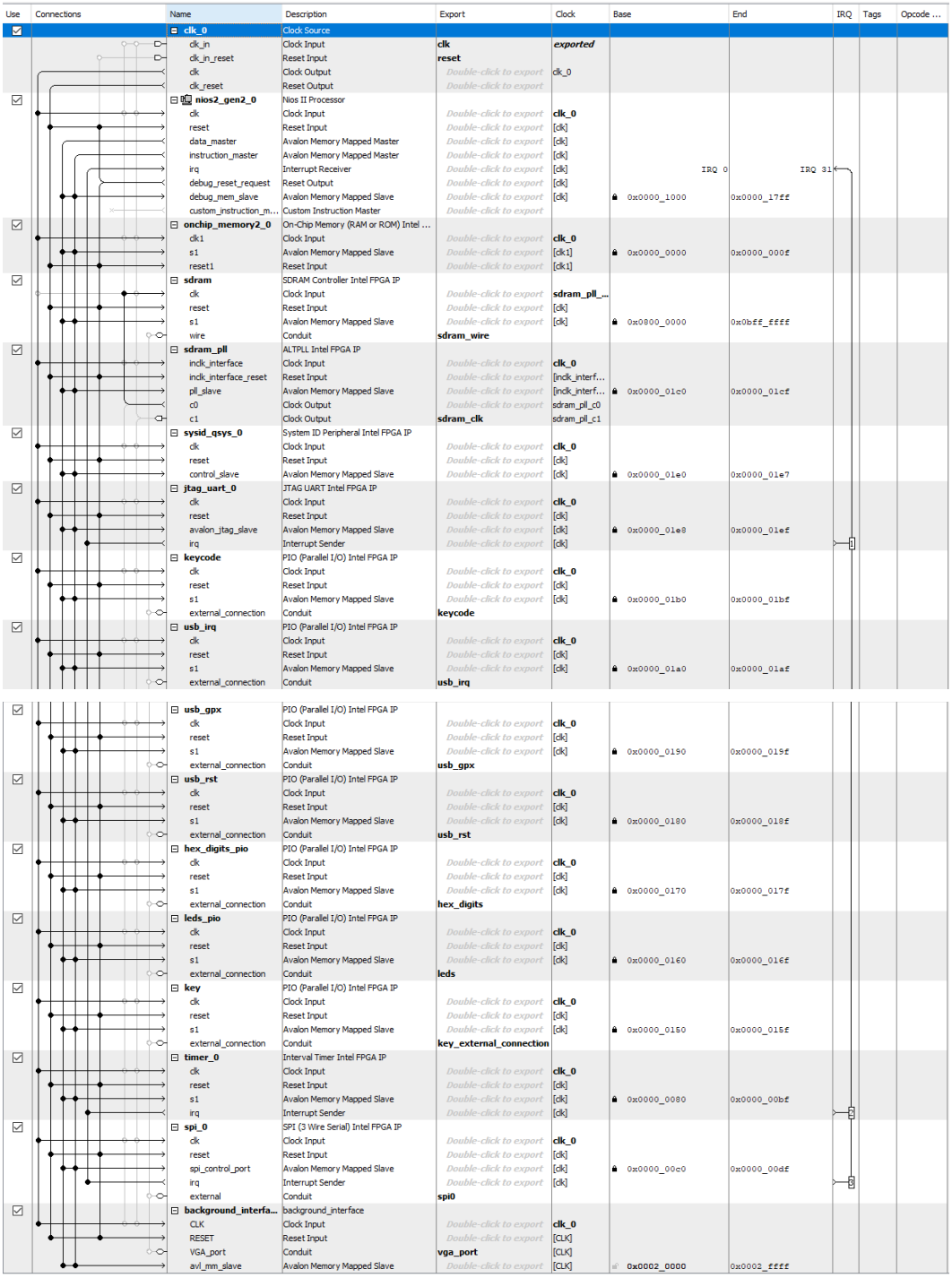

Figure 21. Platform Designer

8. Design Resources and Statistics

LUT	27504
DSP	0
Memory (BRAM)	535,680 / 1,677,312 (32 %)
Flip-Flop	2911
Frequency	71.2 MHz
Static Power	98.34 mW
Dynamic Power	437.85 mW
Total Power	556.89 mW

Table 1. Statistics of Project

9. Conclusion

Overall the final project went really well for us. We followed our planned time line well and made some brilliant discoveries such as the use of the rotation matrix. Many things we learned in this class were very useful for achieving the goals we want for the game, especially the NIOS II interface.

It's a pity that we failed to build the audio portion in the project. There are some optimizations such as the glitch around the tank, given the short period of time, we think we did a really good job at achieving the goals that we designed.