ECE385

SPRING 2020

Final Project – RUNNNN!

Final Project – RUNNNN!

Xinhao Tong & Jieting Chen
LA-3
Ziyang Xu

1. Written Description:

(Description of the overview of the circuit)

RUNNNN, our final project , is a Dungeoned-liked game while the definition of victory is different from Dungeoned. There are two players, a predator and a prey, namely, player 1 and player2. Both players have two different unique skills to choose. For player1, the predator, the goal is to catch player2 while for player 2, the objective is keeping running as long as possible to avoid being caught by the predator. For convenience, we define more than 300 seconds as a victory for prey. We start from lab 8 files. This means we have a synthesized NIOS II processor and an integrated USB keyboard that control the moving of two players and output their position onto the screen using the VGA interface.

Generally speaking, there are three scenes in RUNNNN. In the first scene, two players can choose their skills such as putting traps. In the second scene, the main game screen, there are default blocks of road and wall, with the positions of the two players initialized in the upper left and lower right corner of the screen. Besides, there is a timer on the far left of the screen, recording the survival time of player 2. The last scene shows up when the prey is caught up, either touched by predator or fall into the trap. This is an image of van Gogh's work, with the survival time of the prey on it.

There are several special features in our game. Not only does

RUNNNN implement all basic functions, such as displaying the map with wall-blocks and road-blocks, displaying the players' current position on the map, controlling the movements of the players with keyboard-inputs, successfully judging the win and lose condition, but also achieves some difficult features. RUNNNN supports 2-players mode and adds a timing module to record the time to win in the second scene and displays a scoreboard on the third scene. What's more, players can choose their skills before the chasing game. That is, there are totally four ways to enjoy the game for two players.

(Description of the general flow of the circuit)

Inputs and outputs for the top level:

Inputs

CLOCK_50 – logic

KEY – logic [3:0]

OTG_INT – logic

Outputs

VGA_R, VGA_G, VGA_B – logic [7:0]

VGA_CLK, VGA_SYNC_N, VGA_BLANK_N, VGA_VS, VGA_HS – logic

OTG_DATA – logic [15:0]

OTG_ADDR – logic [1:0]

OTG_CS_N, OTG_RD_N, OTG_WR_N, OTG_RST_N – logic

DRAM_ADDR – logic [12:0]

DRAM_BA – logic [1:0]

DRAM_DQM – logic [3:0]

DRAM_RAS_N, DRAM_CAS_N, DRAM_CKE, DRAM_WE_N, DRAM_CS_N , DRAM_CLK – logic

Inout

DRAM_DQ – wire [31:0]

How are the inputs/outputs processed?

The description for all input/outputs are shown in the following picture:

```
module lab8( input               CLOCK_50,
             input        [3:0]  KEY,          //bit 0 is set up as Reset
             output logic [6:0]  HEX0, HEX1,
             // VGA Interface
             output logic [7:0]  VGA_R,        //VGA Red
                                 VGA_G,        //VGA Green
                                 VGA_B,        //VGA Blue
             output logic        VGA_CLK,      //VGA Clock
                                 VGA_SYNC_N,   //VGA Sync signal
                                 VGA_BLANK_N,  //VGA Blank signal
                                 VGA_VS,       //VGA virtical sync signal
                                 VGA_HS,       //VGA horizontal sync signal
             // CY7C67200 Interface
             inout  wire  [15:0] OTG_DATA,     //CY7C67200 Data bus 16 Bits
             output logic [1:0]  OTG_ADDR,     //CY7C67200 Address 2 Bits
             output logic        OTG_CS_N,     //CY7C67200 Chip Select
                                 OTG_RD_N,     //CY7C67200 Write
                                 OTG_WR_N,     //CY7C67200 Read
                                 OTG_RST_N,    //CY7C67200 Reset
             input               OTG_INT,      //CY7C67200 Interrupt
             // SDRAM Interface for Nios II Software

             output logic [12:0] DRAM_ADDR,    //SDRAM Address 13 Bits
             inout  wire  [31:0] DRAM_DQ,      //SDRAM Data 32 Bits
             output logic [1:0]  DRAM_BA,      //SDRAM Bank Address 2 Bits
             output logic [3:0]  DRAM_DQM,     //SDRAM Data Mast 4 Bits
             output logic        DRAM_RAS_N,   //SDRAM Row Address Strobe
                                 DRAM_CAS_N,   //SDRAM Column Address Strobe
                                 DRAM_CKE,     //SDRAM Clock Enable
                                 DRAM_WE_N,    //SDRAM Write Enable
                                 DRAM_CS_N,    //SDRAM Chip Select
                                 DRAM_CLK      //SDRAM Clock
                   );
```

In more details, the VGA signals will control the in/out to the screen using the VGA interface. The value of VGA_R, VGA_G, VGA_B is determined by whether the pixel being drawing the background or players. The OTG signals server as the interface to USB-port device, that

is, the keyboard. The keycode information is generated from keyboard, then sent to EZ-OTG chip, finally given to NIOS II with the help of hpi_to_inst module. The DRAM signals are for reading or writing data from DRAM. DRAM_DQ is the data bus and DRAM_ADDR is the address.

If the outputs are correct, then in the main (second) game screen, the players should move fluently when the keys are pressed and everything works well as the previous description.

2. Module Description:

```
module  color_mapper ( input         [31:0] testpX,
                       input         [31:0] testpY,
                       input         [31:0] testpX2,
                       input         [31:0] testpY2,
                       input         [7:0] p1ab,
                       input         [7:0] p2ab,
                       input         [7:0] scene,
                       input         [31:0] f1,
                       input         [31:0] f2,
                       input         [31:0] tick,

                       input         [9:0] DrawX, DrawY,
                       output logic  [7:0] VGA_R, VGA_G, VGA_B
                     );
```

Description: This is the core and most complex module in our project. It provides the RGB value for each pixel in VGA. testpX, testpY, testpX2, testpY2 are the x and y values to describe the two player's positions, which are provided from the PIO ports between hardware and software. P1ab and p2ab are the values to describe which abilities the two players choose, which will lead to several different maps in the game. Scene is the variable which controls whether the "Start scene", "End scene" or the "main scene" of the game should be

displayed on the screen. The f1 and f2 decide which frame of the players should be displayed. The tick is a kind of time in the game system, an increment by 1 in the tick means the game passes five frames' time. The DrawX and DrawY shows which pixel on the screen is prepared to be draw, and the VGA_R, VGA_G, VGA_B is the RGB values of the current pixel need to be displayed.

```
module wall(input [9:0] wallX,
            input [9:0] wallY,
            input [7:0] p1ab,
            input logic isDoor,
            output logic [2:0] cidx
            );
```

Description: wallX, and wallY describes the pixel's position in the picture, p1ab tells whether to draw the doors on the certain position on the wall and isDoor specify the position where the doors are, cidx give the color index for the palette.

```
module road(input [9:0] roadX,
            input [9:0] roadY,
            input [7:0] p1ab,
            input [7:0] p2ab,
            input [1:0] SD,
            output logic [2:0] cidx
            );
```

Description: roadX, and roadY describes the pixel's position in the picture, p1ab tells whether to draw the planes on the certain position on the road and p2ab whether to draw the shit on the certain position on the road, and SD specify the start or the destination of the plane, cidx give the color index for the palette.

```
module big(input [9:0] nX,
           input [9:0] nY,
           output logic [10:0] cidx
           );
```

Description: it's a big background picture, nX and nY tell the pixel's

position in the picture, and cidx give the color index for the palette.

```
module pallete (input [2:0] cidx,
                output logic [7:0] Red, Green, Blue);

module pallete2 (input [2:0] cidx,
                output logic [7:0] Red, Green, Blue);

module pallete3 (input [2:0] cidx,
                output logic [7:0] Red, Green, Blue);

module palletebig (input [10:0] cidx,
                output logic [7:0] Red, Green, Blue);
```

Description: these four palettes are almost the same, given a specific index, and fetch the corresponding color from switch cases with the RGB values. Palette big have much more colors than other three, and it is for the large background picture.

```
module player1(input [9:0] pX,
               input [9:0] pY,
               input [31:0] f1,
               output logic [2:0] cidx
               );

module player2(input [9:0] pX,
               input [9:0] pY,
               input [31:0] f2,
               output logic [2:0] cidx
               );
```

Description: These two modules are almost the same, pX and oY tell the pixel's position in the picture, and cidx give the color index for the palette, and f1 and f2 tell which frame of the two characters need to be displayed currently.

```
module isplayer(input [9:0] pX,
                input [9:0] pY,
                input [9:0] pX2,
                input [9:0] pY2,
                input [9:0] DrawX,
                input [9:0] DrawY,
                output logic isp1,
                output logic isp2
    );
```

Description: This module is to decide the layer of the picture, given the positions of the two players and the current pixel, return if the

pixel is belongs to the two players by isp1 and isp2.

```
module ability(input [9:0] aX,
               input [9:0] aY,
               input [1:0] aidx,
               input [7:0] p1ab,
               input [7:0] p2ab,
               output logic [1:0] BWY
);
```

Description: this module provides the four abilities' pictures, the aX and aY describe the position where the pixel in the picture, the aidx provides the information that which picture to display, the p1ab and the p2ab describes which abilities are chosen, the BWY is the output tells that the pixel is black or white or yellow.

```
module number (input [9:0] nX,
               input [9:0] nY,
               input [7:0] num,
               input [7:0] scale,
               output logic [1:0] BW);
```

Description: this module provides the ten numbers' pictures, the nX and nY describe the position where the pixel in the picture, the num provides the information that which number to display, the scale is the actual size's portion with 50*50, the BW is the output tells that the pixel is black or white.

```
module lab8( input               CLOCK_50,
             input        [3:0]  KEY,          //bit 0 is set up as Reset
             output logic [6:0]  HEX0, HEX1,
             // VGA Interface
             output logic [7:0]  VGA_R,        //VGA Red
                                 VGA_G,        //VGA Green
                                 VGA_B,        //VGA Blue
             output logic        VGA_CLK,      //VGA Clock
                                 VGA_SYNC_N,   //VGA Sync signal
                                 VGA_BLANK_N,  //VGA Blank signal
                                 VGA_VS,       //VGA virtical sync signal
                                 VGA_HS,       //VGA horizontal sync signal
             // CY7C67200 Interface
             inout  wire  [15:0] OTG_DATA,     //CY7C67200 Data bus 16 Bits
             output logic [1:0]  OTG_ADDR,     //CY7C67200 Address 2 Bits
             output logic        OTG_CS_N,     //CY7C67200 Chip Select
                                 OTG_RD_N,     //CY7C67200 Write
                                 OTG_WR_N,     //CY7C67200 Read
                                 OTG_RST_N,    //CY7C67200 Reset
             input               OTG_INT,      //CY7C67200 Interrupt
             // SDRAM Interface for Nios II Software


             output logic [12:0] DRAM_ADDR,    //SDRAM Address 13 Bits
             inout  wire  [31:0] DRAM_DQ,      //SDRAM Data 32 Bits
             output logic [1:0]  DRAM_BA,      //SDRAM Bank Address 2 Bits
             output logic [3:0]  DRAM_DQM,     //SDRAM Data Mast 4 Bits
             output logic        DRAM_RAS_N,   //SDRAM Row Address Strobe
                                 DRAM_CAS_N,   //SDRAM Column Address Strobe
                                 DRAM_CKE,     //SDRAM Clock Enable
                                 DRAM_WE_N,    //SDRAM Write Enable
                                 DRAM_CS_N,    //SDRAM Chip Select
                                 DRAM_CLK      //SDRAM Clock
                   );
```

Description: this is the top-level of our project, it connects the VGA part and DRAM part together. What's more, it provides KEY and HEX for debugging easily.

```
// You need to make sure that the port names here match the port
lab8_soc nios_system(
                        .clk_clk(clk),
                        .reset_reset_n(1'b1),    // Never reset
                        .sdram_wire_addr(DRAM_ADDR),
                        .sdram_wire_ba(DRAM_BA),
                        .sdram_wire_cas_n(DRAM_CAS_N),
                        .sdram_wire_cke(DRAM_CKE),
                        .sdram_wire_cs_n(DRAM_CS_N),
                        .sdram_wire_dq(DRAM_DQ),
                        .sdram_wire_dqm(DRAM_DQM),
                        .sdram_wire_ras_n(DRAM_RAS_N),
                        .sdram_wire_we_n(DRAM_WE_N),
                        .sdram_clk_clk(DRAM_CLK),
                        .position1x_export(p1x),
                        .position1y_export(p1y),
                        .position2x_export(p2x),
                        .position2y_export(p2y),
                        .keycode_export(keycode),
                        .f1_export(f1),
                        .f2_export(f2),
                        .tick_export(tick),
                        .p1ab_export(p1ab),
                        .p2ab_export(p2ab),
                        .scene_export(scene),
                        .otg_hpi_address_export(hpi_addr),
                        .otg_hpi_data_in_port(hpi_data_in),
                        .otg_hpi_data_out_port(hpi_data_out),
                        .otg_hpi_cs_export(hpi_cs),
                        .otg_hpi_r_export(hpi_r),
                        .otg_hpi_w_export(hpi_w),
                        .otg_hpi_reset_export(hpi_reset)
);
```

Description: This module provides the values from software to hardware. And the values have been already introduced in the Color mapper module.

3. Design Procedure / State Diagram

1. Overview of the design procedure

   a) What project/codes is used as the foundation of the project?

   We use the lab8 codes as the foundation of this project, because the keycode part and VGA part in lab8 can be reused in this project.

   b) What are the different objectives of the project (choices of inputs, state machine, sprites, algorithm IPs, storage units, choices of outputs)?

For most part, we use on-chip-memory for storage, because the size of each picture is really small for most scenes, such as 30*30 or 50*50 (pixels), and most of them can be reused by instantiating only once. What's more, we use PIO for the interaction between software and hardware, because the information we need to interact is little. We create two state machines, one big and one small, the big one controls the switching of scenes in the whole game, and the small one only presented in one player's one special ability.

c) What research/background study has been done to achieve the objectives?

Learn how to use SDRAM, learn how to use PIO to pass values from C codes to hardware, learn how to use VGA to display the colored pixels, learn the basic idea of SystemVerilog coding, learn how to present multi-layer pictures on the screen in a right order.

d) How are the different objectives linked together to form a complete project?

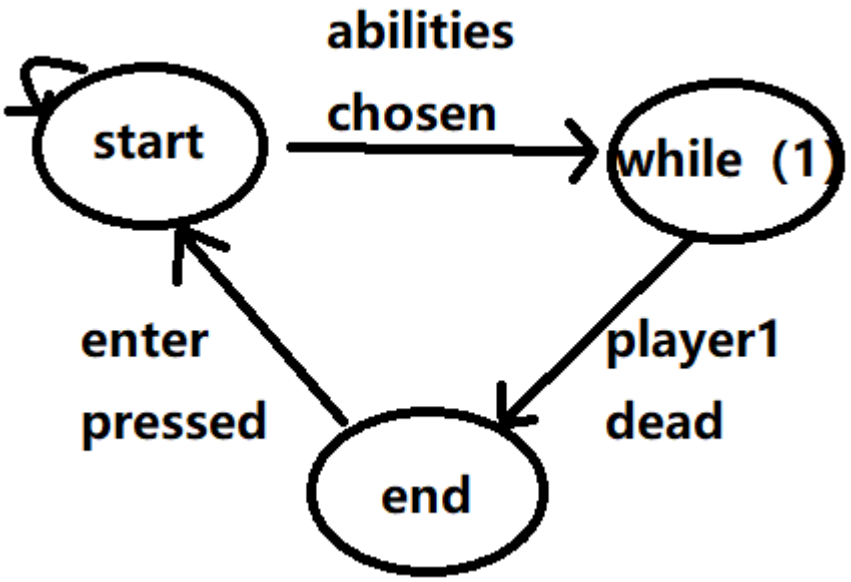
In the software part, the state machine will output the values of the core information of the scene, players and so on. Through PIO, these information will be passed to the hardware, and the modules in the hardware will uses the pictures which are

stored in the on chip memory to get each pixel's color and present them on the screen by VGA.

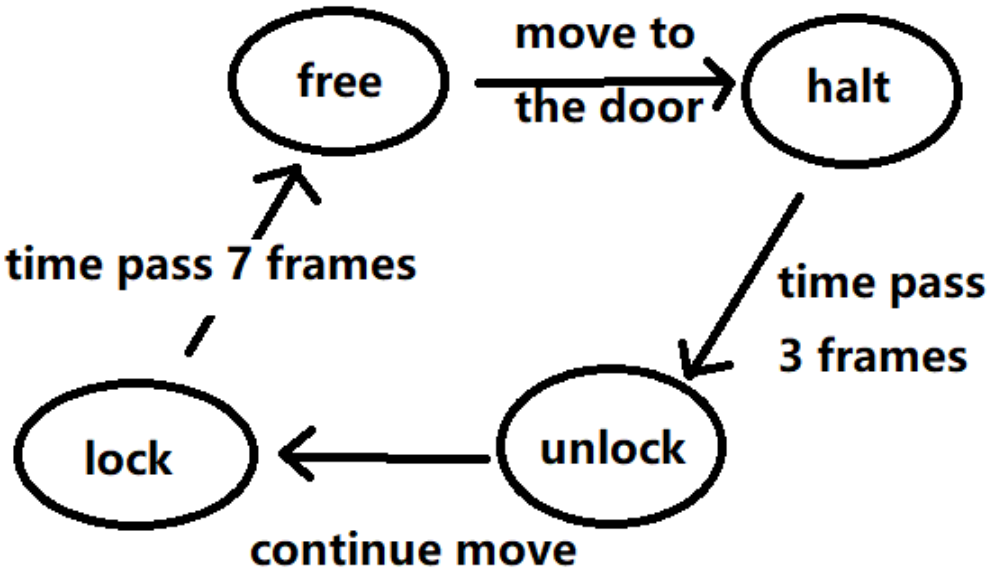
2. If some sort of serial processing is used, especially for the projects that deals with algorithms, a state machine and a simulation waveform should be included in the report

i. State Diagram

The first State diagram is about an overview of the scenes in the game:

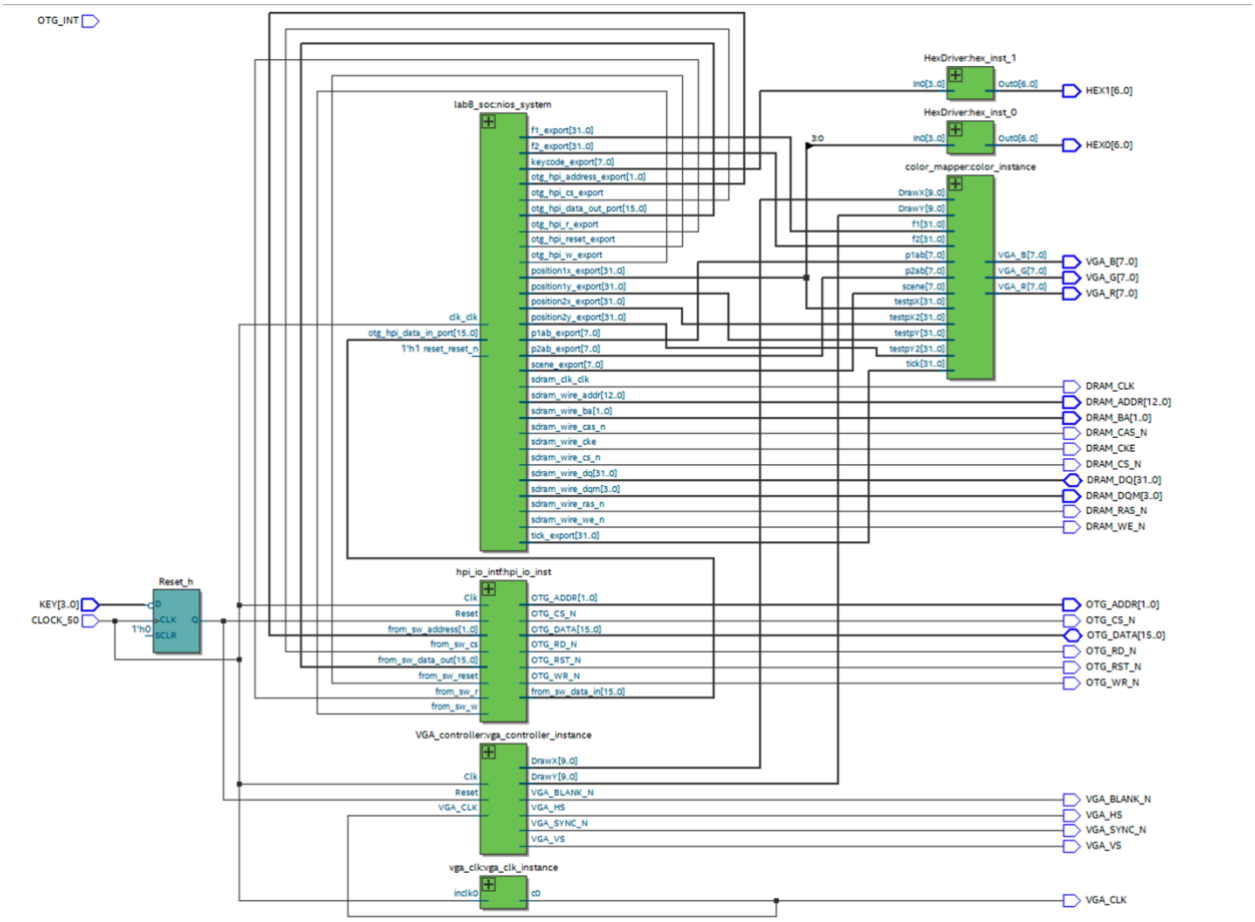


The second state diagram is about one of the abilities of player1:



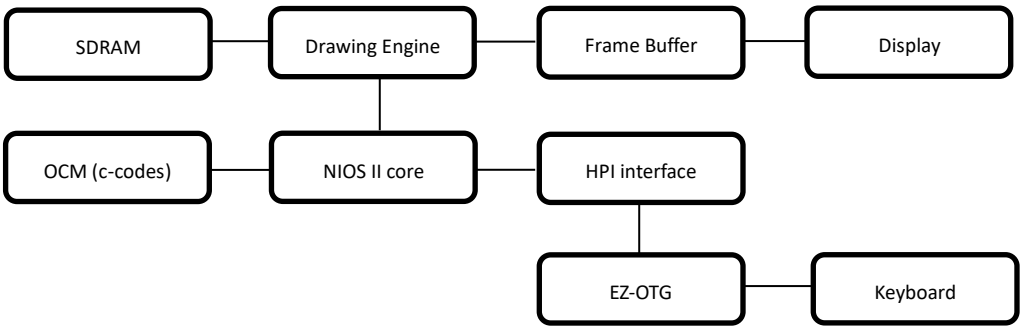
4. Block Diagram:

The top-level view of block diagram is shown below, with all modules, ports and interconnections labeled (which is legible enough). For more detailed, can be seen in Module Description section.



From another view:

lab8.sv



Above block diagram is our top-level diagram, with C code written in OCM and executed on NIOS II. We store the background images(blocks) also in the OCM since they are not memory-occupied in our implementation. Keyboard module will take the inputs as parameters

and control the movement of the players. Then the NIOS II will select the corresponding sprite and get position information with help of the drawing engine and store it into frame buffer. Finally, by looking up the palette, we draw all pixels onto the monitor with help of the display engine.

5. Post-Project

LUT	28819
DSP	/
Memory (BRAM)	11392
Flip-Flop	2771
Frequency	132.07 MHz
Static Power	106.90mW
Dynamic Power	0.88
Total Power	174.54mW

6. Conclusion

It’s a cool game and we think we make it well. Some schoolmates have enjoyed it and appreciated it.