ECE 385

Fall 2019

FINAL PROJECT

Pokemon SystemVerilog Edition

Eric Dong, Yifu Guo

ericd3, yifuguo3

Section AB3, Thursday 2pm

Gene Shiue

Introduction:

In this lab, we created a Pokemon game that is based on lab8. We implement a USB and VGA system that is able to control the movement of our character on a screen with USB keyboard inputs. Depending on which one of WASD keywe input, the character will “go” into the corresponding direction. The goal of the project is to recreate the Psychic gym of the classic Pokemon Fire Red game. The gym consists of 9 rooms, and several more teleportation pads that allows the player to explore the other rooms. In the game the player is supposed to try to get to the center room, room 5, in order to fight the gym leader and get the badge. Due to time limitations, we have only been able to implement the teleportation logic in SystemVerilog.

Game explanation:

The in game character is trying to reach the gym leader who is supposed to be located in the center room, so that he can battle her. In our replication of the game, the character is only able to traverse the rooms and try to get the center room. There are special teleportation tiles on the floor that is able to transport the character to different rooms until he reaches the center room. The character is able to walk slowly and run depending on whether a key is pressed.

Difficulty:

- Sophisticated graphics drawing: the framebuffer logic to draw the next pixel is really sophisticated, there was complicated indexing to get the character to move as well as the map to scroll by tiles. To teleport the character, we also had to make the character spin, teleport then spin again.

Overall design procedure:

Hardware design procedure in System Verilog:

The hardware design procedure is based on a framebuffer approach in drawing the game. The framebuffer is located in the on chip memory, and already contains a start screen when the file is copied to the OCM. The resolution of the game is really 240x160 in order to have the same user experience as on the Gameboy Advance, where the game was first launched. Since the resolution is pretty small, we were able to scale up the game by 2 in both the X direction as well as the Y direction.

Software design procedure in C:

The software we used is basically the software we used in the lab8. Since almost all the functions are implemented in SystemVerilog, the software is pretty simple in our project.

Software description:

The main goal of the software is to achieve the interaction between DE2-115 board and keyboard.

There are 4 main functions:

1. The IO_Write function simply writes to the address of the OTG chip with the address we want to write to with the data that we would like to write.
2. The IO_Read function simply reads the data available from the address that we provide it with.
3. The USBWrite function writes data from the NIOS processor to the OTG chip. It first writes an address to the address register, then it writes the desired data to the data register to send to the host.
4. The USBRead function is used for reading data available from the device. We first write to the address register the address we want to read from, then we can read from the data register with the data from device already available.

Hardware description:

The entire game is controlled by a finite state machine which determines what should happen, and what is next. The game first starts with the start screen with our department head, Bruce Hajek since he is our "boss". Then when the enter key is pressed, there is a fade state where the pixels dim out, but in actuality, it happened way to fast for human eyes to see the fading. Next, and final state is actual drawing of the main map. There, a character is drawn in the center of the map while the background is being scrolled.

The FSM determines several outputs that drives the game. These include whether the character is moving which determines whether the background should scroll. There is also character move frame which determines which of the frames of the walking character should be drawn. In addition, there is also the direction variable which determines which direction the character should be facing based on the keyboard input. Finally there is also the tile counter which makes sure that the character is walking in 16 pixels which is the width of one game tile.

In the core of the project, the frame drawer module, the system updates the framebuffer during the blanking period when the monitor is not displaying anything. In the framebuffer logic module, we utilize the drawX and drawY coordinates of the VGA and determines what pixel should be displayed next. For example, in the very center of the screen, we draw the main character, and if the coordinates are not within that small window, we draw the background sprite instead. Depending on the keycode input from the keyboard, we are able to scroll the background and change different frames of the walking character to make the character appear to be walking in the gym.

In order for the map to scroll, we determined that if one of the WASD keys is pressed, then there should be scrolling. In the original game, the character is able to either walk or run, so we implemented both. For walking, when a key is pressed, we scroll the background every other frame, using the VGA_VS as a helping tool. And as for running, we would scroll the background every frame so it would be twice as fast.

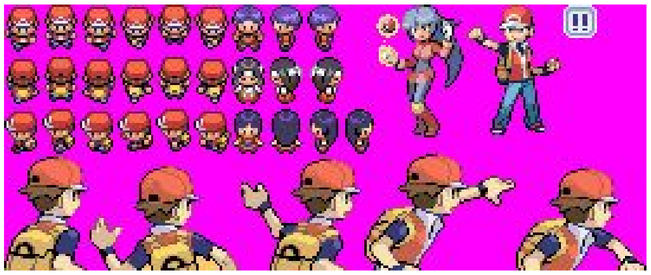
In order for the character not to walk through the walls of the gym map, we created a gymBoundsChecker module which is fed with the current coordinates of the character on the map. Then it compares it to several different hard coded values in order to determine whether the map should keep on scrolling or not.

Another obstacle detection feature is determining which tile the character is stepping on. In the gymTileLogic module, there is a list of positions where there is a tile. If the character's position on the map is at that tile, then the character will be teleported to another tile. Also within the gymTileLogic module, there is a FSM. This FSM is able to override the position in the main frame drawer logic as well as the direction of the character. In the original game, the character first spins then he teleports, we are doing something similar here. The FSM has a counter which is split into 4 parts, each encoding a different direction of spinning. In action, the FSM first determines that the character is on a tile, then it first spins, then teleports, the spins again at the new tile, the finally it holds. We were not able to completely finish debugging the tile logic: the character is not able to move off the new tile when we draw the background, but without the background, the character is able to get off the tile.

One last final component of the project is the sprite drawing. The character is drawn according to a sprite that is stored in the on chip memory. The framedrawer module is given several parameters: the direction that the character faces, the walking/running frame that should be drawn as well as whether the character is running. Then indexing is done so that the correct image is drawn. Notice below that there is no character sprite that faces right, that means we were able to use the left facing sprite to generate the right facing one using more indexing. The rest of the sprite is not used due to time constraints on the project.

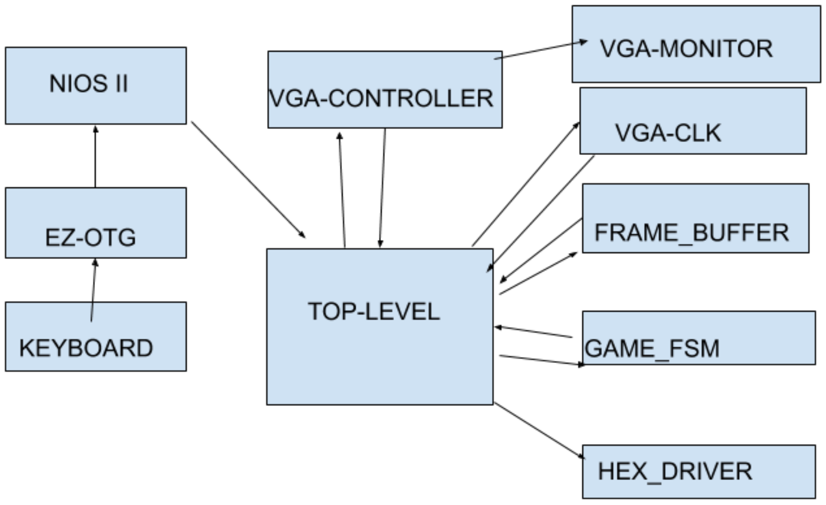
Description of sprite sheet:

We used a 271*112 sized sprite sheet, which contains the character we will use in our game. First, we use the python script on the website to generate the txt file. Then we use a module called ram.sv to read and store the txt file in the on chip memory for further use. In order to efficiently use the on chip memory, we had to save the sprite in a way such that instead of having each pixel take up 24 bits, we used 8 bits to represent each color, and in the actual ram module in SystemVerilog, we created a look up table to decode each color into the original 24 bits. To do this, we edited the python script written by Rishi by using piskelapp.com to generate a histogram of all the colors present in the image.

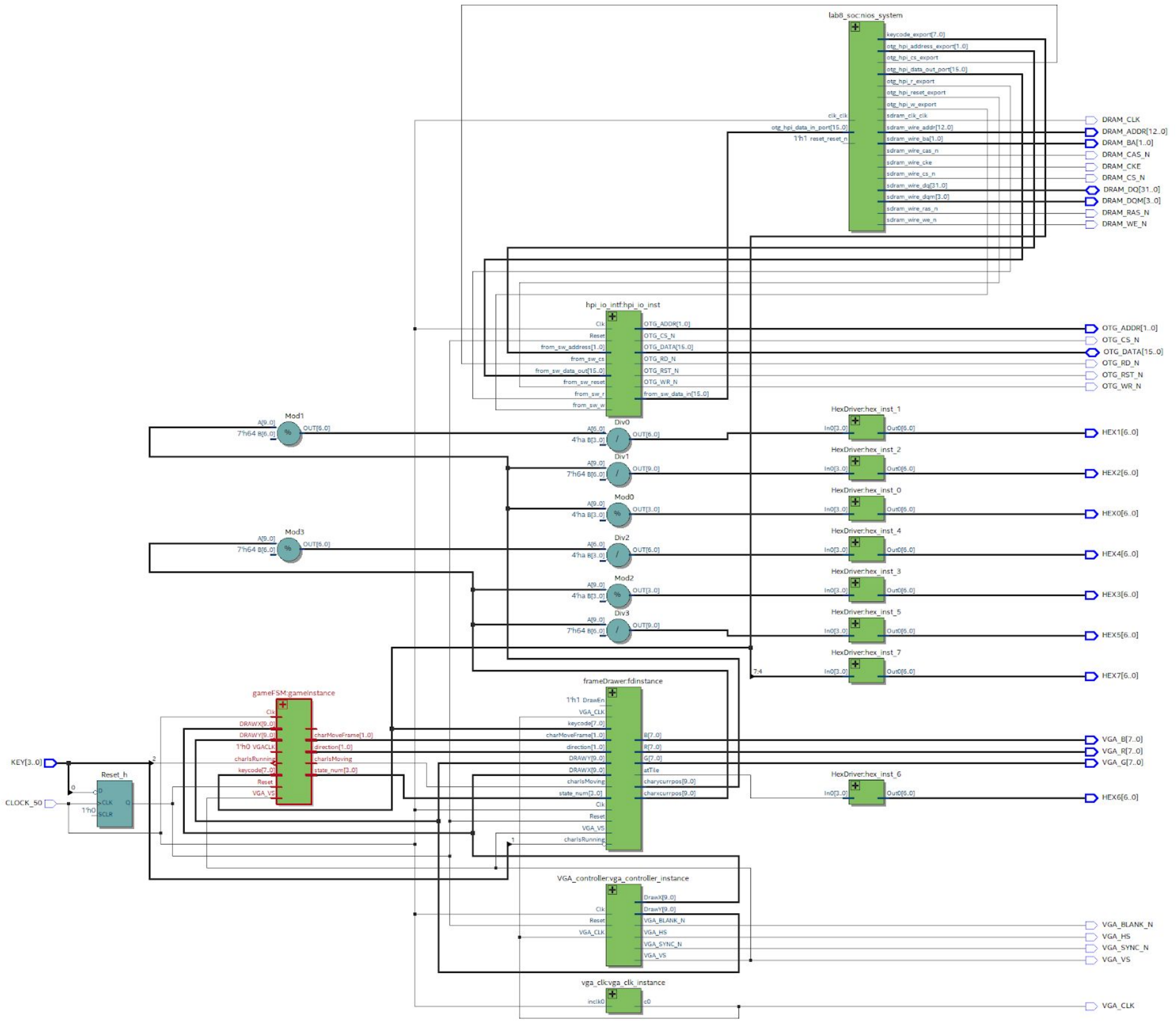


We also use the same method to read and store another picture as our game's background.

Block diagram:

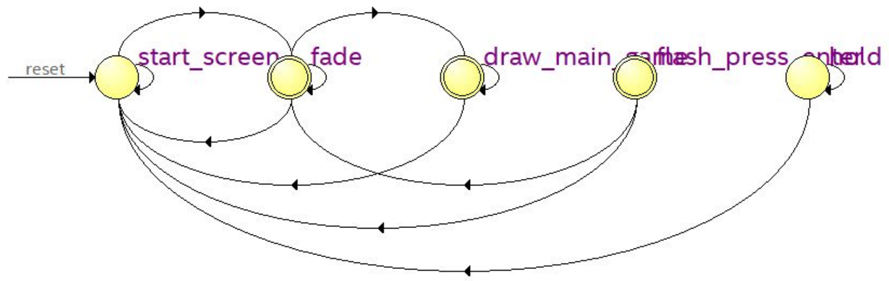


Top level diagram:

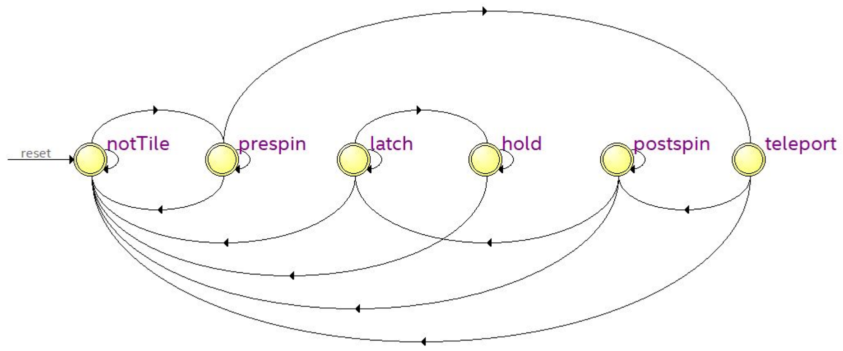


State diagrams:

Game finite state machine:



Framebuffer state machine:



Module descriptions:

- HexDriver
  - Input:
    - In0
  - Output:
    - Out0
  - Description:
    - This module helps to display the hex_value that we input on the DE2-115 board.
  - Purpose:
    - This module is used to display the keyboard input ASCII character onto the hex display.

- Hpi_io_intf
  - Input:
    - Clk, Reset, from_sw_address, from_sw_data_out, from_sw_r, from_sw_w, from_sw_cs, from_sw_reset,
  - Output:
    - From_sw_data_in, OTG_ADDR, OTG_RD_N, OTG_WR_N, OTG_CS_N, OTG_RST_N, OTG_DATA
  - Description:
    - This module interfaces with the OTG chip to help us use the USB keyboard.
  - Purpose:
    - This module sends the read, write, address, databuffer signals to the OTG chip. It also determines what signals should be high when the USB chip is being reset. Finally the module also allows us to control the input output bus.
- Lab8
  - Input:
    - CLOCK_50, KEY, OTG_INT,
  - Output:
    - HEX0, HEX1, VGA_R, VGA_G, VGA_B, VGA_CLK, VGA_SYNC_N, VGA_BLANK_N, VGA_VS, VGA_HS, OTG_DATA, OTG_ADDR, OTG_CS_N, OTG_RD_N, OTG_WR_N, OTG_RST_N, OTG_INTDRAM_ADDR, DRAM_DQ, DRAM_BA, DRAM_DQM, DRAM_RAS_N, DRAM_CAS_N, DRAM_CKE, DRAM_WE_N, DRAM_CS_N, DRAM_CLK
  - Description:
    - This is the top level module of our game.
  - Purpose:
    - This is to connect all the individual modules together. It makes the connections between the NIOS 2, easy OTG chip connection, our frame_buffer, Ram, gameFSM, gymBoundsChecker, gymTileLogic, and the VGA controller.

VGA_clk

Input:

inclk0

Output:

c0

Description:

This module generates the VGA_clock.

Purpose :

This module helps us to generate the 25MHZ clock for our VGA_controller.

VGA_controller

Input:

Clk, Reset, VGA_CLK

Output:

VGA_HS, VGA_VS, VGA_BLANK_N, VGA_SYNC_N, DrawX, DrawY

Description:

This module controls vertical and horizontal sync signals in VGA.

Purpose:

This module helps us to track which pixel we are working on.

framebuffer:

Input:

Clk, VGA_CLK, VGA_VS, DrawEn, Reset,
charIsMoving, charIsRunning,
[1:0] direction, charMoveFrame,
[3:0] state_num,
[9:0] DRAWX, DRAWY,
[7:0] keycode,

Output:

[7:0] R, G, B,
[9:0] charxcurrpos, charycurrpos,
atTile

Description:

This contains all the ram modules which holds all the sprites. It also tells the VGA controller what color to draw.

Purpose:

We use a framebuffer in order to make the drawing process faster. When the monitor is displaying the current frame, the next frame is being drawn at the same time: after th VGA draws a line of the frame onto the monitor, while the monitor is blanking, the next frame is being determined and stored into the framebuffer memory.

gameFSM:

Input:

Clk, VGACLK, VGA_VS, Reset,
[7:0] keycode,
[9:0] DRAWX, DRAWY,
charIsRunning,

Output:

charIsMoving, PLAY,
[1:0] direction, charMoveFrame,
[3:0] state_num

Description:

This module provides a state machine to start the process of the game.

Purpose:

This module is used to control the process of the game, and sends to the framebuffer several signals that controls what to draw in the next frame.

gymBoundsChecker:

Input:

[1:0] direction,
[9:0] charxcurrpos, charycurrpos,

Output:

atBounds

Description:

This module provides a lot of conditions to detect the boundaries of the gym map.

Purpose:

This module is used to check whether or not the character encountered a boundary: the module returns a 1 if the character is at the bounds, and tells the framedrawer not to keep scroll the map anymore.

gymTileLogic:

Input:

Clk, Reset, VGA_VS,

[9:0]   xright, ybottom,

Output:

atTile,

[1:0]   spin_direction,

[9:0]   xleft_next_out, ytop_next_out

Description:

This module gets the input of the current location and output the next jump location if the character is standing on a special teleporting tile.

Purpose:

This module makes the character able to jump to the designated tile. This the main idea of the game so that the character can traverse through the map and get to the gym leader.

Ram:

Input:

[23:0] data_In,

[18:0] write_address, read_address,

we, Clk

Output:

[23:0] data_Out

Description:

This module contains multiple modules to read the on-chip memory.

Purpose:

This module is used to read and write to the on-chip memory. Of these ram modules, two are used to store game images such as the character sprite as well as the gym map, and the other one is used as the framebuffer.

Design resources and statistics:

LUT	34,469
DSP	0
Memory	935,776
Flip-Flop	2312
Frequency	45.14 MHz
Static Power	109.72 mW
Dynamic Power	339.04 mW
Total Power	534.59 mW

Conclusion:

Overall the project was a really fun process. We were mostly successful, and everything works pretty well except little glitches. I think if we were given more time and more man power, then we would've been able to perfect this project. We learned a lot from this project, starting with sprite drawing, using memory, as well as using a framebuffer, there was so much to learn from this experience.

Our game works almost correctly, but has a tiny problem. When we load the background image, the function of teleporting to the designated tile doesn't work correctly. However, when we didn't load the background image, the function of jumping through designated tile works perfectly. So, we still need to do some extra research to work out this problem.

Thank you so much for this great semester, Gene! Hope you have a blessed holiday!

Works Cited Page:

Sprites:
https://www.spriters-resource.com/game_boy_advance/pokemonfireredleafgreen/
https://github.com/Atrifex/ECE385-HelperTools
https://pokemondb.net/sprites

Other Pokemon game References:
https://github.com/Tarokan/ECE-385-Final-Project
https://github.com/jyerra2/PokemonGameFPGA

Audio Attempt:
https://kttechnology.wordpress.com/2018/11/10/ece-385-final-project-notes-simple-sound-effect/
https://wiki.illinois.edu/wiki/download/attachments/700854875/Audio_documentation.txt?version=1&modificationDate=1565813028000&