ECE 385

Fall 2016

Final Project

Doesn't Sid get to be a character in the game? Or at least a sequel?

ECE 385: The Game

Awesome title

Neil Patel & Siddhant Jain

Lab AB1

Conor Gardner

Introduction

ECE 385: The Game, our final project, is a sideway-scroller which objective is for Neil, our main character, to collect as many as many difficulty points and coins for the 9 labs throughout the semester. One of those coins is a powerup coin that is meant for our transition from TTL Labs to SystemVerilog. The game ends when Neil fails to not reach his goal of A+ in ECE 385 and has to hope for the curve. We attempted the following points in our project: baseline, score counter, wall, multiple levels, bitmapped, palletized, ps/2 keyboard, basic graphics, font sprites, dynamic, humor, and custom scripts. Our design process started with us playing a few retro games; we explored the original Sonic Game and the Super Mario Bros 3. To get inspiration, we bought a usb NES controller to play these games to understand what we wanted to build. We decided to do our best emulation of Super Mario Bros 3. Our mindset to build our game was to start with lab 8 files. This meant we had a synthesized NIOS II processor and an integrated USB keyboard that outputted a moving ball onto the screen using the VGA interface. The first thing we did was generate the palletized bitmapped sprites which we would attempt to display through the color mapper and eventually through the ROM. We found images we liked and ran them through Rishi's scripts. We then made a python script that would sort them into nice arrays that we could input into our sprite.sv. Since we were using a palletized sprites, we also created a python script that would output code based on a particular palette that would go into our color mapper. For the text that we wanted to output, we came upon the issue of font's actually using various shades of grey; we wrote and edited version of our script to create Monochrome version of the text sprite. We also decided to implement the whole project in SystemVerilog, so we promptly instantiated the PS/2 Keyboard that was given to us and thus removed Eclipse from our

project. We then implemented coins and blocks so that we could build out the level; after that, we worked on object collisions and getting the coin to disappear when it is met with the character. We also made jump so that the hero character, Neil, would not be floating around through the game. We built a score counter to make the game interesting such that we count the number of difficulty points Neil obtained and at the same the number of labs we accomplished. Lastly, we tried to interface the NES USB controller however we were only able to get left and right working on the controller. Ultimately, we made a finished product with Neil traversing the level and collecting coins.

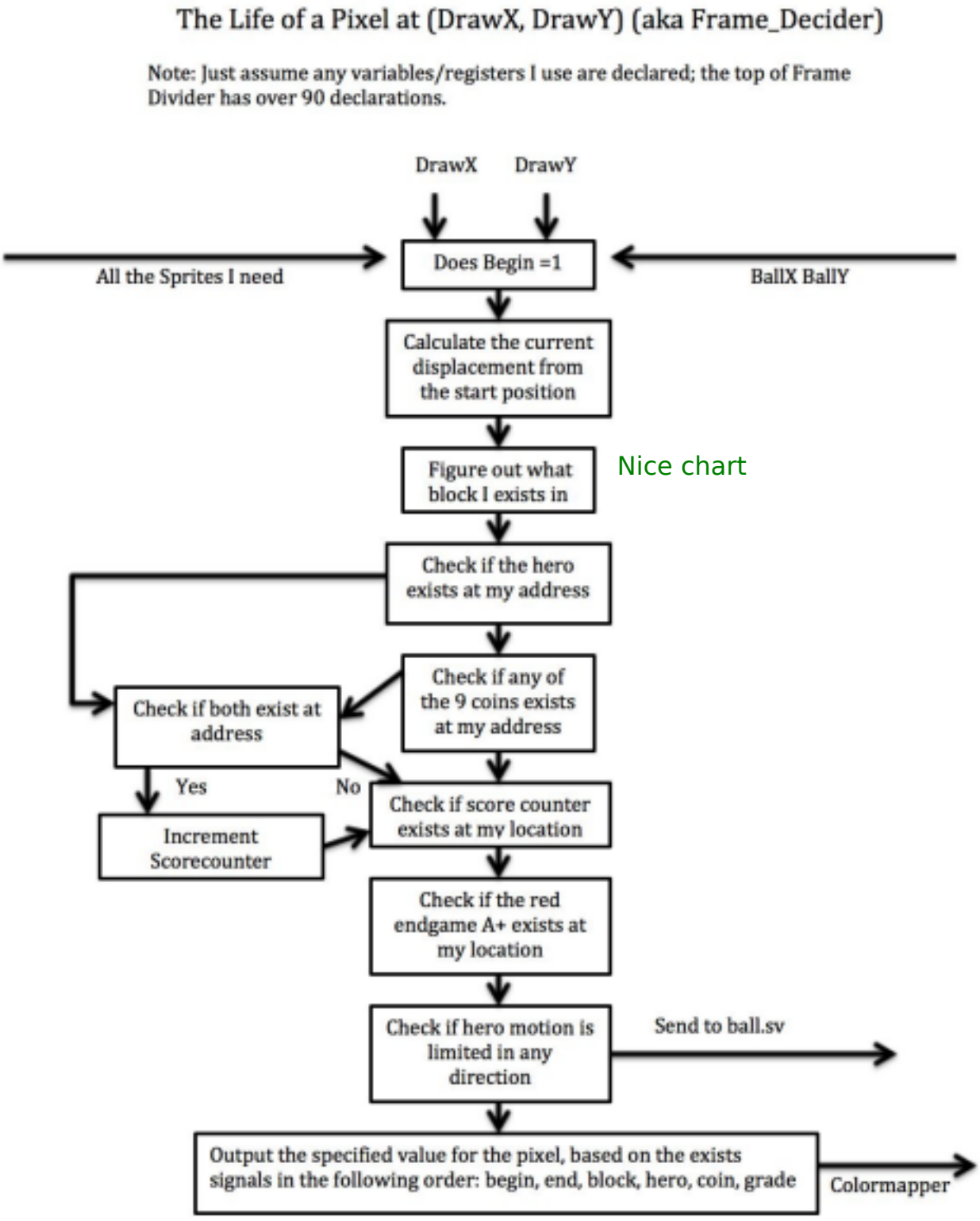
Summary of Circuit

Our final project uses an SV file called lab8.sv to kind of hold things together. All the major signals are routed through it. Many of the sprites used in game are stored in arrays in sprite.sv . They are output to lab8.sv, where they are then transferred into framedecider.sv. The keyboard module records both keypresses and the keycodes coming in from the PS2 keyboard, and transfers through lab8.sv to ball.sv. Based on the keypress, and signals coming in from framedecider (these are the startgame, endgame, and various stop signals that are triggered by walls), ball.sv determines the position that hero will be located in the next frame (moves him). These x and y coordinates go back through lab8.sv and into framedecider.sv. Frame_decider has over 90 internal signals declared at the beginning, all either locations or values to be set that are used to check for existence of sprites at various locations. It starts off by waiting for a keypress to move it from the begin stage. The begin stage puts up an orange screen and displays various game text ontop of it. Once a key is pressed, the begin screen disappears. Now the game is split

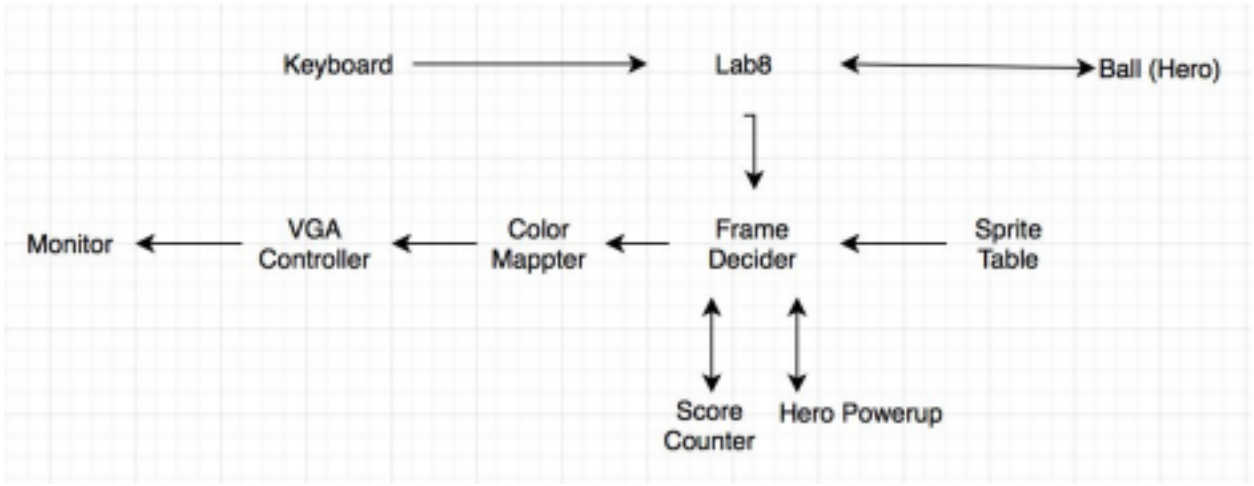
into a few major parts. The first determines where the walls go; we created a sort of “framemap” that is a representation of the level in 40*40 grid blocks. If a block exists as a 1 in the frameset, it is time to draw a wall block there; the first few sets of combinational logic determine if a wall is there and what specific wall_text pixel it should be. It also figures out how far hero has moved from his last position, allowing for the scrolling to function by using these numbers as an offset initialx added to DrawX. The next part of Frame_decider determines if certain sprites need to appear, and if so where they are. It does this by determining which pixel it is (using draw_X and draw_Y) and then determining where in the sprite this pixel exists, and what its value is, assigning value to its exists signal. Value is a palletized number from 1-50 that correlates to a color. Frame_decider instantiates a hero_sprite_decider module, which takes a powerup signal from the “SystemVerilog red coin” and outputs either the regular (if Neil hasn’t eaten the red coin yet) or golden version of the sprite. Other sprites in this category include the coins and the grade text. It also checks for collisions between hero and these parts and outputs signals if a collision has occurred. Once the coins have been drawn, Frame_decider actually instantiates a scorecounter.sv object; this object takes the scoreincrement signals that are outputted when a coin is eaten and adds 1 to the score counter. Based on the current score internal signal, score counter changes which numbers are outputted to frame_decider as the score1 and score2 numbers; these numbers are then printed in the top righthand corner. These are some of the only things not to have the offset initialx added to their x positions because they need to stay in the top right as hero moves, not appear to move along as part of the game's environment. The final part of Frame_Decider is an always combinational loop that takes the signals from each of the sprites and blocks and determines, based on various signals including DrawX and Draw Y, the

various exists signals from the sprites, and certain game state signals (begin and end) to determine the value that should be outputted. Value comes out of framedecider and goes through lab8.sv and into color_mapper.sv. Based on the value coming in, color mapper maps to one of 56 colors; this color is then passed to the VGA_controller.sv file in Red, Green, and Blue channels for them to be displayed on the screen.

Flow Chart


The Life of a Pixel at (DrawX, DrawY) (aka Frame_Decider)
Note: Just assume any variables/registers I use are declared; the top of Frame Divider has over 90 declarations.
DrawX
DrawY
All the Sprites I need
Does Begin =1
BallX BallY
Calculate the current displacement from the start position
Figure out what block I exists in
Nice chart
Check if the hero exists at my address
Check if any of the 9 coins exists at my address
Check if both exist at address
Yes
No
Increment Scorecounter
Check if score counter exists at my location
Check if the red endgame A+ exists at my location
Check if hero motion is limited in any direction
Send to ball.sv
Output the specified value for the pixel, based on the exists signals in the following order: begin, end, block, hero, coin, grade
Colormapper

Top Level Block Diagram



Modules

Module: keyboard.sv

Inputs: Clk, psClk, psData, reset

Outputs: [7:0] keyCode, press

Description: PS/2 Keyboard interface that allows to interact with our game without the NIOS II Eclipse Software.

Module: Dreg.sv

Inputs: Clk, Load, Reset, D

Outputs: Q

Description: It is a 1-bit register that is a part of the functionality of keyboard.sv and the PS/2 keyboard.

Module: 11_reg.sv

Inputs: Clk, Reset, Shift_In, Load, Shift_En, [10:0] D

Outputs: Shift_Out, [10:0] Data_Out

Description: It is a 1-bit register that is a part of the functionality of keyboard.sv and the PS/2 keyboard.

Module: VGA_controller.sv

Inputs: Clk, reset

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

Description: The purpose of the VGA controller is to control video output from the FPGA and through to the monitor.

Module: lab8.sv

Inputs: CLOCK_50, [3:0] KEY, PS2_CLK, PS2_DAT

Outputs: [6:0] HEX0, HEX1, [7:0] VGA_R, VGA_G, VGA_B, VGA_CLK, VGA_SYNC_N, VGA_BLANK_N, VGA_VS, VGA_HS

Description: Lab8.sv is the top level file that connects all the other components to each other, and ensures that the right sized buses are present.

Module: HexDriver.sv

Inputs: [3:0] In0

Outputs: [6:0] Out0

Description: The purpose for the Hex Driver is to convert a 4-bit input into a 7-bit hex display code. It also defines how the FPGA hex displays particular inputs.

Module: Color_Mapper.sv

Inputs: [0:5] value

Outputs: [7:0] Red, Green, Blue

Description: Based on the inputted value, color mapper finds the appropriate color in its palette and outputs it as R, G, and B to VGA_controller, allowing for that specific pixel to be printed in that specific color.

Module: ball.sv

Inputs: Reset, frame_clk, [15:0] keycode, press, downstop, rightstop, upstop, leftstop, startscreen, ported

Outputs: [9:0] BallX, BallY

Description: Ball.sv is primarily used to calculate the position of our hero neil and the movement of it on the screen. It also specifies how to control our hero for specific keypresses.

Module: sprite.sv

Inputs: clk

Outputs: [0:39][0:39][0:5] coin_text, [0:39][0:39][0:5] enemy_text, [0:39][0:39][0:5] wall_text, [0:39][0:39][0:5] pow_text, [0:39][0:39][0:5] pow_coin_text, [0:39][0:39][0:5] grade_text, [0:11][0:34][0:1] frameset, [0:37][0:277][0:1] game_text, [0:11][0:143][0:1] diff1_text,

[0:11][0:143][0:1] diff2_text, [0:11][0:143][0:1] diff3_text, [0:11][0:143][0:1] diff4_text, [0:11][0:143][0:1] ending1_text, [0:11][0:143][0:1] ending2_text, [0:11][0:143][0:1] ending3_text, [0:11][0:143][0:1] ending4_text, [0:46][0:253][0:1] gameover_text

Description: Sprite.sv is the module that holds the arrays for all of the different sprites that we have.

Module: Frame_decider.sv

Inputs: [9:0] BallX, BallY, DrawX, DrawY, Ball_size, clk, press, [0:39][0:39][0:5] coin_text, [0:39][0:39][0:5] enemy_text, [0:39][0:39][0:5] wall_text, [0:39][0:39][0:5] pow_text, [0:39][0:39][0:5] pow_coin_text, [0:39][0:39][0:5] grade_text, [0:11][0:34][0:1] frameset, [0:37][0:277][0:1] game_text, [0:11][0:143][0:1] diff1_text, [0:11][0:143][0:1] diff2_text, [0:11][0:143][0:1] diff3_text, [0:11][0:143][0:1] diff4_text, [0:11][0:143][0:1] ending1_text, [0:11][0:143][0:1] ending2_text, [0:11][0:143][0:1] ending3_text, [0:11][0:143][0:1] ending4_text, [0:46][0:253][0:1] gameover_text

Outputs: [0:5] value, downstop, rightstop, leftstop, upstop, startscreen

Description: The main purpose of Frame_decider is to control the operation of what goes onto the screen since the inputs of the module consist of the Ball movement and the multitude of sprites. Based on various internal signals and registers, it determines whether a) something exists at a particular pixel, b) whether that thing is to be printed or not, and c) is there a collision happening that can changes things. For more indepth information look at the High level circuit description.

Module: scorecounter.sv

Inputs: clk, scoreincrement1, scoreincrement2, scoreincrement3, scoreincrement4, scoreincrement5, scoreincrement6, scoreincrement7, scoreincrement8, scoreincrement9, scoreincrement10

Outputs: [0:39][0:39][0:5] score1_text, [0:39][0:39][0:5] score2_text

Description: Scorecounter.sv is the module that uses the input scoreincrements to determine the current score and outputs those digits arrays to Framedecider.

Module: hero_sprite_decider.sv

Inputs: clk, power

Outputs: [0:79][0:39][0:5] hero_text

Description: This module determines what the hero looks like. It renders Neil before eating the powerup and after eating the powerup based off an input signal called power. The purpose of this module is to help create dynamic sprites.

Picture of Game Cool graphics

Design and Resource Table

LUT	5,507
DSP	N/A
Memory (BRAM)	N/A
Flip-Flop	135
Frequency (MHz)	95.2
Static Power (mW)	98.89

Dynamic Power (mW)	0.00
Total Power (mW)	150.32

Conclusion

Building ECE 385: The Game was the biggest accomplishment we have made in our academic career. It made us respect the engineers who made games in the 20th century with subpar equipment and who had to code these complex games using TTL Chips. It made us appreciate the advancement in technology in the past 40 years. We also really enjoyed this project because we had to start from the barebones. We were able to get Neil to traverse through the level and collect coins to get to the final A+ grade. This capstone project made the class worth it and we thoroughly enjoyed what we learned this past semester.

Citations

Coin: http://makepixelart.com/peoplepods/files/images/1844351.original.png

Enemy:https://orig08.deviantart.net/90dc/f/2011/209/3/0/dig_dug_enemy_by_zenak781-d41yzx4.png

Hero:http://orig05.deviantart.net/ccf0/f/2012/253/9/8/pokemon_black_white_hero_ultra_pixel_by_thekeko-d5e7l8r.png

Wall: http://img05.deviantart.net/19ca/i/2011/359/3/5/brick_block_by_kkiittuuss-d4k5lrp.png

A+: https://www.gtc.edu/sites/default/files/files/connect/grade.jpg

Pow block: http://piq.codeus.net/static/media/userpics/piq_10231_400x400.png

Power up Coin: http://piq.codeus.net/static/media/userpics/piq_46746_400x400.png

ECE 385 Unified Final Project Rubric
Fall 2016
Conor Gardner's Grading Pool

High Category	Mid Category	Low Category	Earned (Percent)	Points Possible
Final project proposal		Was attempted	100%	2.5
		Contained a high-level description of the project	100%	2.5
		Contained a rough high-level block diagram	100%	2.5
		Contained a feature list or difficulty point estimation	100%	2.5
Final project mid-checkpoint		Attendance	100%	2.5
		Students started the project	100%	2.5
Final project demo		Project barely works and has serious bugs	100%	5
		Project works as expected (minor bugs or missing features OK)	100%	10
Final project report		Introduction	100%	2
	Written Descriptions	High level written description	100%	3
		Described each module's function	100%	2
		Described each module's connectivity to other modules	100%	2
	High level block diagram	Was included	100%	3
		Was neat (no messy quartus outputs)	100%	2
	State machine OR flowchart OR state table	Was included (if required)	100%	2
		Was neat (1 page, no messy quartus outputs)	100%	2
		Conclusion	100%	2
Final project difficulty	Basic	Baseline. Everyone who attempted the final project gets this	100%	1
		Humor. Silly project titles / funny sprites / etc. This skill helps you survive ECE :)	100%	1
		Multiple levels / Start Screen / End Screen / Main Menu	100%	0.5
		Controllable parameters. Can change things like audio volume / game speed while device is running.	0%	0.5
	AI	Computer controlled players for games.	0%	1
	Physics	Basic walls. Detecting and handling any kind of hard object (bricks in brick breaker / screen edge)	100%	0.5
		Basic overlap detection (such as hitting incoming notes in DDR / Guitar Hero)	100%	0.5
		Advanced walls (mazes/packman)or more interactive objects (springboards/portals)	0%	1
		Advanced deflection (motion of paddle in brick breaker affects how a ball bounces)	0%	0.5
		Advanced projectile motion. Parabolic trajectories. Irregular gravity.	0%	1
		Highly irregular shapes other than walls such as tetris blocks	0%	1
		Object rotations	0%	0.5
	Graphics	Basic graphics	100%	0.5
		Dynamic sprites (explosions or color changes)	100%	0.5
		Bitmapped Sprites	100%	0.5
		Palletized sprites	100%	0.5
		Dithering	0%	0.5
		Wrote custom code in any language to convert images (png/bmp/jpg) to mif/hex/systemverilog files	100%	0.5
		Text/font sprites	100%	0.5
		Transparency or alpha channel	0%	0.5
		Additional image processing filters. Can go over 100% for multiple filters up to 300%	0%	0.5
	Audio	Basic audio output	0%	0.5
		Basic audio input	0%	0.5
		Generated audio (building specialized hardware to generate complex outputs)	0%	1
		Sampled audio (stored audio samples in RAM and streaming to DAC)	0%	1
		Dynamic audio effects such as changing amplitude/pitch or audio mixing	0%	1
	Interface	USB / PS2 keyboard	100%	0.5
		Xbox or other game controller	50%	1.5
		MIDI controller	0%	1
		Camera	0%	1
		Integration of non-trivial IP not used in class (RAM/ROM wizard doesn't count)	0%	1
	Architecture	Pipelining	0%	1
		Look-up tables (other than sprites)	0%	0.5
		Fixed-point numbers	0%	0.5
		Approximation (increasing speed or reducing area at the expense of accuracy)	0%	0.5
		Replacing multiplies/divides with shifts/adds/subtracts	0%	0.5
		RAM/ROM banking	0%	1
		Random number generators in hardware	0%	0.5
		Double buffering. Usually used for graphics	0%	0.5
		Auto-increment for address generators	0%	0.25
		Adding/modifying LC3 or processor instructions to control hardware such as DMA or accelerators	0%	0.5

You scored 7.75 out of 10 difficulty points and 57.75 out of 60 total points for the final project

Erratum (changes from the rubric included with the proposal)

1. This is the only report which I did not grade anonymously, due to the unique nature of each project
2. Removed the "Score Counter" item from the "Basic" category since it is essentially the same thing as "Font Sprites" in the "Graphics" category.
3. Added "Image processing filters" to the "Graphics" category
4. Added "IP integration" to "Interface" category
5. Added "Auto-increment" item to the "Architecture" category
6. Added "Adding/modifying LC3 instructions" item to the "Architecture" category
7. Reduced some point weights to balance out additional items.