ECE 385

Fall 2016

4 November 2016

You should decide from the begining wheather you're going to use bitmapped graphics or not. Once you start a project, it's really hard to add features later and you'll need to discard your previous graphics. Bitmaps aren't even that difficult and have a great (points)/(effort) ratio.

Approved!

Final Project Proposal

Neil Patel & Siddhant Jain

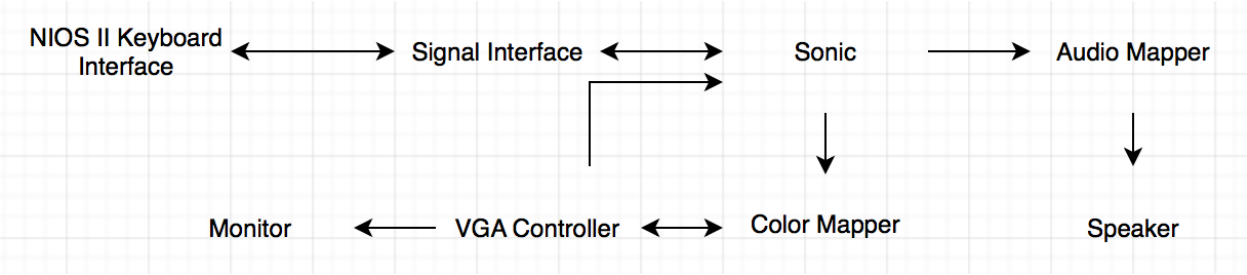
Lab AB1

Conor Gardner

Idea & Overview

We propose to design and implement our take on Sonic the Hedgehog on the Altera DE2 board. Using System Verilog, we plan on implementing a sonic controller, color mapper, and VGA controller, as well as controllers for various types of enemies and objects. Using The NIOS II CPU, we plan on implementing a USB interface (like lab 8) to allow for the keyboard (and eventually hopefully an NES controller) to control our character. The display will be output over VGA, allowing for a fully functional Sonic game to be played.

Block Diagram



List of Features

Baseline Features

For our Sonic game to be considered working, we need several things. A start screen, the first level, and an end screen for level progression. The level itself will have basic walls. Visuals include basic graphics (shapes and the like). For the interface, we will hook up the NIOS II Keyboard, allowing for the sprite to be navigated across the level using the arrow keys.

Additional Features

The additional features we really hope to include span all the categories.

For the physics, implementing some kind of overlap detection allows for us to add enemies and/or coins to the screen. Developing more advanced walls/obstacles allow for us to add springboards. We hope to implement the ability for sonic to bounce off the walls at different angles and speeds resulting in different bounces, which would implement both advanced deflection and projectile motion. If we feel really ambitious, we could implement the famous Sonic loop-de-loops, which would require irregular gravity. For the actual interfaces, we plan to have a score counter, which displays the points earned from killing enemies, and possibly the number of coins collected, if we implement coins. The stretch goal is also to implement a USB NES controller for our game, as a more authentic controller for the game. In reality, we hope to implement bitmapped sprites so that our Sonic actually looks like a sonic, rather than some sort of shape. We are also hoping for dynamic sprites, so that enemies "explode" on contact. Other graphics improvements we plan to implement are palletized sprites and sprites for text. We are also planning to implement some sort of audio; basic audio at the least, but hopefully some sort of sampled audio so that the sounds are authentic.

Expected Difficulty

Our baseline project is around a 5 in difficulty; although at that point it won't much resemble sonic in terms of graphics or game play, it will more be some sort of basic shape making its way through a level. These points come from between walls, basic graphics, and keyboard inputs, and a finished project.

We think the expected difficulty for our project will be much closer to a 10 if we get enough of our additional functionality listed above working; springboards, coins, enemies, more realistic

physics, real graphics (read bitmapped and dynamic), and audio output are all significant amounts of work and difficulty.

Proposed Timeline

In the 5 weeks we have to work on the Final Project, we think if we follow our defined plan below then we successfully create Sonic the Hedgehog on the Altera DE-155 FPGA board.

Week 1	Understand/ plan out exactly how/ what we need to build for each part of the game in order for it to work; defining PIO blocks for Qsys, understanding/programming the basic wall items.
Week 2 (Mid-Project Checkpoint)	Hopefully implementation of baseline game; at this point Sprites probably don't exist (sonic can be a ball) but keyboard input, the level, and basic walls exist. If they don't work together yet (ie game is unplayable), progress is demonstrable through showing code and sprite progress.
Week 3 (Thanksgiving Break)	Implement more complex physics and graphics; Lab Partner lives in same town back home.
Week 4	Implement Coins; final touches as the game should be done this week.
Week 5 (Demo Week)	Implement NES controller. Demo Project. Write Project Report.

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		2.5
		Contained a high-level description of the project		2.5
		Contained a rough high-level block diagram		2.5
		Contained a feature list or difficulty point estimation		2.5
Final project mid-checkpoint		Attendance		2.5
		Students started the project		2.5
Final project demo		Project barely works and has serious bugs		5
		Project works as expected (minor bugs or missing features OK)		10
Final project report		Introduction		2
	Written Descriptions	High level written description		3
		Described each module's function		2
		Described each module's connectivity to other modules		2
	High level block diagram	Was included		3
		Was neat (no messy quartus outputs)		2
	State machine OR flowchart OR state table	Was included (if required)		2
		Was neat (1 page, no messy quartus outputs)		2
		Conclusion		2
Final project difficulty (50% credit if a hardware feature was implemented in software)		Baseline. Everyone who attempted the final project gets this	100%	1
		Humor. Silly project titles / funny sprites / etc. This skill helps you survive ECE :)	0%	1
	Basic	Multiple levels / Start Screen / End Screen / Main Menu	0%	0.5
		Score counter (must either be on-screen or in decimal). No credit for hex on 7-segment displays	0%	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%	2
		Pattern recognition for images or audio. Doesn't need to be a neural net	0%	1
		Used a neural network in a place where simpler approaces would be insufficient.	0%	2
		If neural nets were used, demonstrated an intuitive knowledge of how they were trained.	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)	0%	1
		Bitmapped Sprites	0%	1
		Palletized sprites	0%	1
		Dithering	0%	1
		Wrote custom code in any language to convert images (png/bmp/jpg) to mif/hex/systemverilog files	0%	1
		Text/font sprites	0%	1
		Transparency or alpha channel	0%	1
		Hardware image processing filters such as gaussian blur or anti-aliasing.	0%	2
	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	0%	1.5
		MIDI controller	0%	2
		SD Card	0%	1.5
		Camera	0%	2
	Architecture	Pipelining	0%	2
		Look-up tables (other than sprites)	0%	1
		Fixed-point numbers	0%	1
		Approximation (increasing speed or reducing area at the expense of accuracy)	0%	1
		Replacing multiplies/divides with shifts/adds/subtracts	0%	1
		RAM/ROM banking	0%	1
		Random number generators in hardware	0%	0.5
		Double buffering. Usually used for graphics	0%	1
		Created hardware SIMD units	0%	1
		Direct memory access (DMA) or bulk memory mapped IO	0%	0.5
		Wrote low-level C++ or assembly code to exploit special features such as SIMD.	0%	0.5

Estimated minimum difficulty score 3 out of 10 for the final project

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		2.5
		Contained a high-level description of the project		2.5
		Contained a rough high-level block diagram		2.5
		Contained a feature list or difficulty point estimation		2.5
Final project mid-checkpoint		Attendance		2.5
		Students started the project		2.5
Final project demo		Project barely works and has serious bugs		5
		Project works as expected (minor bugs or missing features OK)		10
Final project report		Introduction		2
	Written Descriptions	High level written description		3
		Described each module's function		2
		Described each module's connectivity to other modules		2
	High level block diagram	Was included		3
		Was neat (no messy quartus outputs)		2
	State machine OR flowchart OR state table	Was included (if required)		2
		Was neat (1 page, no messy quartus outputs)		2
		Conclusion		2
Final project difficulty (50% credit if a hardware feature was implemented in software)		Baseline. Everyone who attempted the final project gets this	100%	1
		Humor. Silly project titles / funny sprites / etc. This skill helps you survive ECE :)	0%	1
	Basic	Multiple levels / Start Screen / End Screen / Main Menu	100%	0.5
		Score counter (must either be on-screen or in decimal). No credit for hex on 7-segment displays	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.	100%	2
		Pattern recognition for images or audio. Doesn't need to be a neural net	0%	1
		Used a neural network in a place where simpler approaces would be insufficient.	0%	2
		If neural nets were used, demonstrated an intuitive knowledge of how they were trained.	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)	100%	1
		Advanced deflection (motion of paddle in brick breaker affects how a ball bounces)	0%	0.5
		Advanced projectile motion. Parabolic trajectories. Irregular gravity.	100%	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%	1
		Bitmapped Sprites	100%	1
		Palletized sprites	100%	1
		Dithering	100%	1
		Wrote custom code in any language to convert images (png/bmp/jpg) to mif/hex/systemverilog files	100%	1
		Text/font sprites	100%	1
		Transparency or alpha channel	100%	1
		Hardware image processing filters such as gaussian blur or anti-aliasing.	0%	2
	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	0%	1.5
		MIDI controller	0%	2
		SD Card	0%	1.5
		Camera	0%	2
	Architecture	Pipelining	100%	2
		Look-up tables (other than sprites)	0%	1
		Fixed-point numbers	0%	1
		Approximation (increasing speed or reducing area at the expense of accuracy)	0%	1
		Replacing multiplies/divides with shifts/adds/subtracts	0%	1
		RAM/ROM banking	100%	1
		Random number generators in hardware	0%	0.5
		Double buffering. Usually used for graphics	100%	1
		Created hardware SIMD units	0%	1
		Direct memory access (DMA) or bulk memory mapped IO	0%	0.5
		Wrote low-level C++ or assembly code to exploit special features such as SIMD.	0%	0.5

Estimated maximum difficulty score 10 out of 10 for the final project