

ECE 385

Spring 2020

Final Project

**Legend of Zelda Dungeon Demo using SOC
with USB and VGA Interface in SystemVerilog**

Michael Faitz and Zohair Ahmed

Section ABJ: Friday 2:00-4:50

Yuming Wu and Lian Yu

Introduction

Our final project was to create a short dungeon in the style of the original The Legend of Zelda. We did this by modifying the SystemVerilog code from Lab 8. We created a simple dungeon without puzzles. We simply required that the enemy in the room was defeated before being able to advance. Each room also features an immovable object. Link has access to a sword, and is expected to defeat the enemies in the dungeon without taking a hit, and the game ends with Link holding up the Triforce. We chose this game because we are both fans of games in the Zelda series, and thought it would be a cool project to do.

Keyboard Controls

Link's movement is controlled with keys W, A, S, D, moving him up, left, down, and right respectively. Link can use his sword by pressing the Y key, but he will be unable to move while doing this. He will use his sword in the direction of the key last pressed (from W, A, S, D). The spacebar can be pressed to reset the game. This will take Link back to room one with full health. It will also respawn the enemy in the room. This is mainly used when Link dies or wins the game, but can be used at any time.

Game Rules

The game features four rooms, each has an enemy and an immovable block. The game has no score or points, you must simply get to the last room and defeat the enemy to win. Link must kill an enemy for the door to reveal itself, without dying himself. When you arrive at the third room, you will not be able to go through the door to the right, as it is locked. You will instead have to find the secret passage. After completing the secret passage, you will be able to go to the fourth room, and win the game.

Written Description of Project System

The NIOS processor has access to the addresses of the CY7 chip, allowing it read/write capabilities. From this, we can read the inputs from the USB chip, which we process with functions in IO_handler and usb files. By using a similar set of modules in platform designer to lab 7, we already set up clocks that ensure correct data transfers will occur. We additionally created modules for the otg_hpi variables we used in our C code, to ensure proper handling of interfacing with the CY7. The VGA output required that we had the VGA_controller module, which produced the timing signals needed to ensure proper synchronization with the DE2's VGA output. These timing signals ran at a frequency that, when used with the 50 MHz clock of the FPGA, allowed us a 60Hz framerate for the VGA output. We also used the Color_Mapper module to handle object rendering and coloring, such that we could see Link, the enemy, doors, the object, and the rest of the screen.

The overall flowchart of the project system instantiates a playable rectangular area in the center of the screen and from this we modified ball.sv to be the main character, Link. In Lab 8, we had a case statement governing movement based on a keypress. If W is pressed, we move up. If A is pressed, we move left, down for S and right for D. Our default case in Lab 8 was to

maintain the current motion. In our project, we changed this default case to set the motion back to zero, so that we only move when the keys are pressed. We included some additional code for room transitions, health calculation and interactions with an immovable object. Within this playable rectangular area there are available doors on the sides which serve as room transition zones. These rooms were handled via a state machine which tracked the room that Link was currently in and when he was transitioning. When the player enters the hitbox of the door after the enemy on the screen has been defeated they are teleported to the other side of the screen within the opposing door hitbox. This teleportation aspect of the project serves to simulate Link walking through the left dungeon room door and exiting the right door of the following room. When Link performs this room transition the enemy in the room is respawned and moves to a different location on the screen to identify the changing of the room and vary gameplay.

Our enemy module is very similar to that of Link's, but used a counter to determine which direction to move in. This counter allows for the enemy to move in a square on the screen which is similar to several enemies in the original game. Damage calculation for both Link and the enemy are handled within this module as the enemy sprite loading area serves as the active hitbox for the enemy as well. When the Link character sprite collides with this enemy hitbox he is damaged and thus dies.

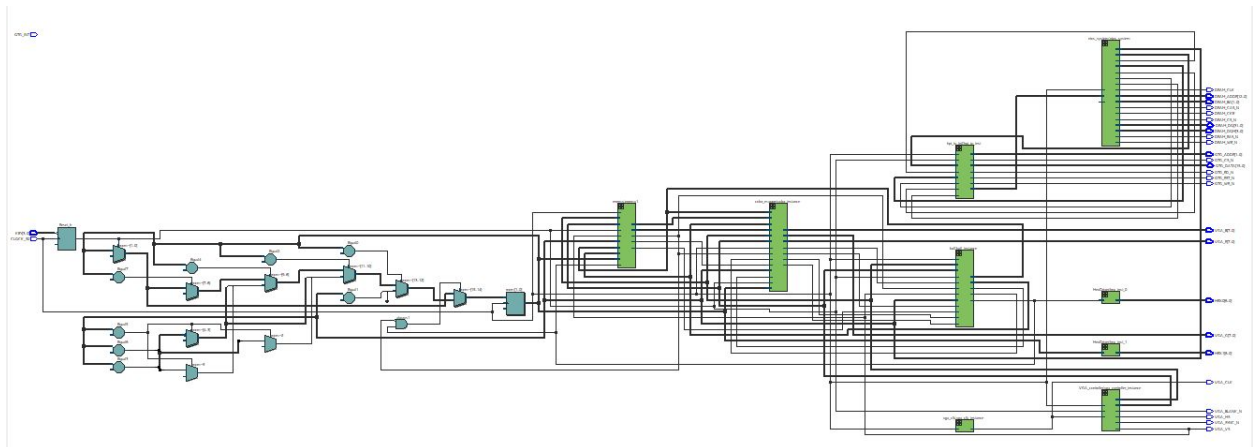
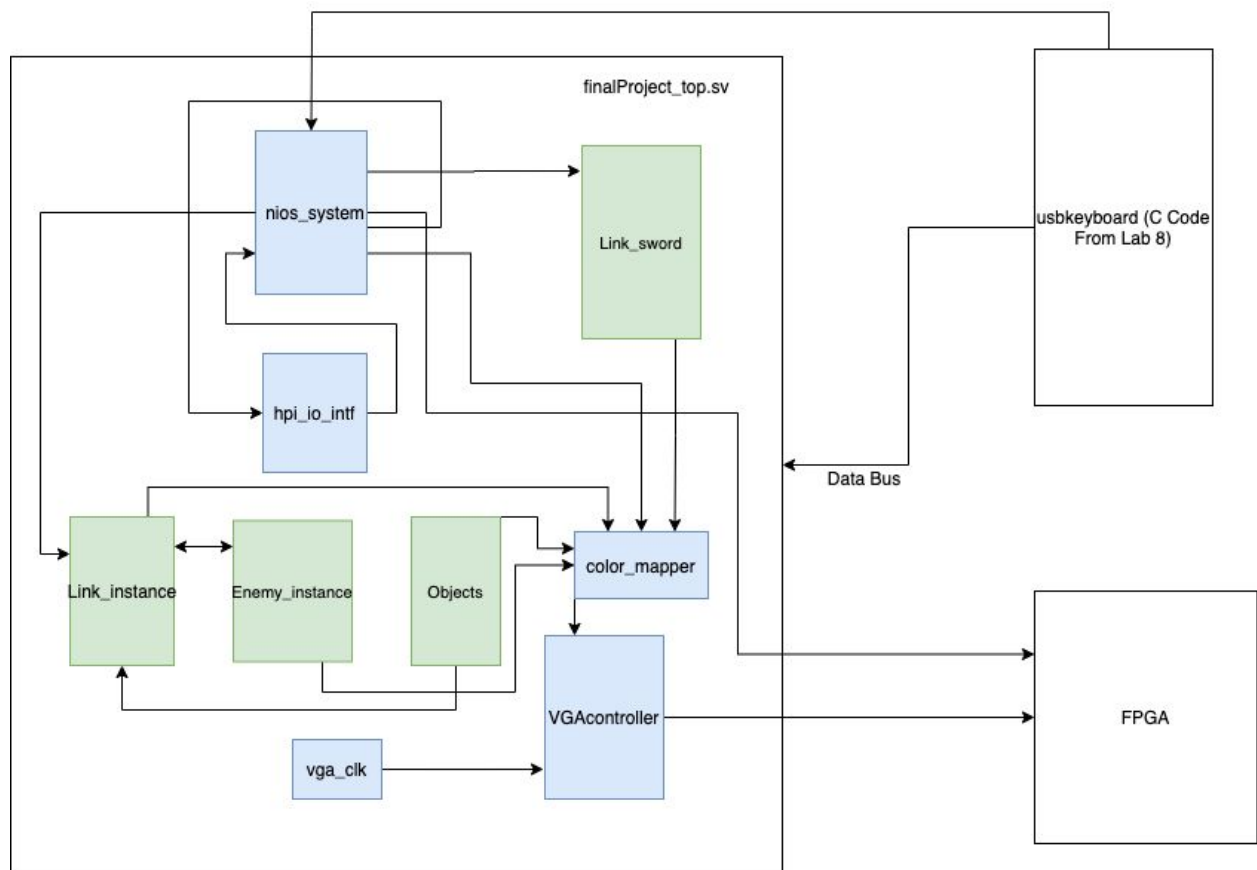
In order to kill the enemy Link must use his sword which can be activated using the "y" key on the keyboard. To do this Link must also be standing still which mimics the original game and simplifies the input reading and handling from the USB keyboard settings from Lab 8. Once the sword has been activated it activates both a block of white pixels loading in front of Link and a separate hitbox for damaging the enemy. From testing we realised that the hitbox for the sword needed to be different from that of the visual sword as it was inconsistent in damaging the enemy and felt frustrating to play with. To combat this issue we made a rectangular box hitbox in front of Link which was separated from the sword sprite loading pixels which worked way more consistently and smoothly.

After implementing one way sword hitboxes we adjusted the code for the hitbox to remember the last directional key pressed which would then tell us the direction that Link was facing. From that data we could then generate the sword in the forward facing direction of Link and thus Link was a functional and playable character.

For the final essential feature of the project we implemented an immovable object colored as a rock which Link could not pass through. To do this we identified that treating Link as a box and considering his sprite and hitbox as a series of four corners would be the best method as if one of the four corners of Link entered the rock then he could be bounced off of the surface of the rock. This logic was then duplicated for the enemy.

While we did not get to implement as many additional features as we wanted to, one feature that we were able to implement was a secret room in the third room which has a locked door on the right side of the screen after defeating the enemy. In order to clear this barrier the player must discover the hidden door not shown on the screen on the left side of the room and replay through the third room. After this the right door will open and allow for the player to enter the final room. After killing the enemy within this room Link is loaded in the center of the screen holding up the Triforce as a symbol of victory.

Block Diagram



Module Descriptions

hpi_io_intf.sv

Inputs: Clk, Reset, from_sw_address [1:0], sw_r, from_sw_w, from_sw_cs, from_sw_reset, from_sw_data_out [15:0]

Inouts: OTG_DATA [15:0]

Outputs: OTG_RD_N, OTG_WR_N, OTG_CS_N, OTG_RST_N, from_sw_data_in [15:0], OTG_ADDR [1:0]

Description: The variables containing sw are updated by outside modules, and the otg variables are updated to the sw variables as needed. otg variables represent the USB chip inputs.

Purpose: Variables in here need to be updated to ensure correct data reads and writes. The module updates the otg signals such that the C code can correctly read and write from the USB keyboard, or be reset.

VGA_controller.sv

Inputs: Clk, Reset, VGA_CLK

Outputs: VGA_HS, VGA_VS, VGA_BLANK_N, VGA_SYNC_N, DrawX [9:0], DrawY [9:0]

Description: Takes the data of the ball's current position on screen and updates it to the VGA display with the variables DrawX and DrawY.

Purpose: Serves as a compatibility file for the VGA display on the board to the available VGA display. Also serves as an available reset for said display with the reset button of the FPGA.

lab8.sv

Inputs: CLOCK_50, KEY [3:0], OTG_INT

Inouts: OTG_DATA [15:0], DRAM_DQ [31:0]

Outputs: HEX0 [6:0], HEX1 [6:0], VGA_CLK, VGA_SYNC_N, VGA_BLANK_N, VGA_VS, VGA_HS, VGA_R [7:0], VGA_G [7:0], VGA_B [7:0], OTG_ADDR [1:0], OTG_CS_N, OTG_RD_N, OTG_WR_N, OTG_RST_N, DRAM_BA [1:0], DRAM_DQM [3:0], DRAM_ADDR [12:0], DRAM_RAS_N, DRAM_CAS_N, DRAM_CKE, DRAM_WE_N, DRAM_CS_N, DRAM_CLK

Description: Serves as the top level module and provides the output data for the C code to assign to the board and the usb cable. Controls logic to update room variables.

Purpose: Instantiates all the other System Verilog modules and links together the entire program with the help of Qsys.

color_mapper.sv

Inputs: clk, reset_h, is_enemy, is_alive, link_alive, key [7:0], room [1:0], Ball_X_Pos [9:0], Ball_Y_Pos [9:0], Ball_X_Pos2 [9:0], Ball_Y_Pos2 [9:0], DrawX [9:0], DrawY [9:0]

Outputs: up_door, left_door, right_door, down_door, VGA_R [7:0], VGA_G [7:0], VGA_B [7:0]

Description: Used to determine what to draw at a given pixel, sending the necessary RGB values to the VGA outputs. It holds modules used to load sprites, and updates inputs into them (such as the enemy's alive status in the door module) to ensure certain sprites are not drawn before we want them to be displayed.

Purpose: Used to update VGA RGB values for printing to the screen.

ball.sv

Inputs: Clk, Reset, frame_clk, DrawX [9:0], DrawY [9:0], up_door, left_door, right_door, down_door, key [7:0], link_damage, enemy_alive

Outputs: room_trans, is_ball, is_alive, Ball_X_Pos [9:0], Ball_Y_Pos [9:0]

Description: Used to instantiate Link. DrawX and DrawY are used to determine whether or not to draw Link at a given pixel based on his x and y coordinates. This module contains his health, which is affected by the link_damage variable set in enemy.sv. The keypress is used to determine his movement. Variables enemy_alive, up_door, left_door, right_door, and down_door are used in room transition logic. When we want to transition, we set room_trans to high.

Purpose: Instantiated as Link character, this module designates the playable character and contains all variables and instructions for responding to various conditions including damage hitbox, movement, room transitions, and sprite loading ranges.

load_link.sv

Inputs: clk, reset_h, address [11:0], key [7:0]

Outputs: dataout [23:0]

Description: Module that holds all sprite data used for Link. Contains 4 sprites and uses the keypress to determine which to output.

Purpose: Used to draw Link in color_mapper.sv by mapping arrays of RGB sequences.

load_sword.sv

Inputs: clk, reset_h, key [7:0], Ball_X_Pos [9:0], Ball_Y_Pos [9:0], DrawX [9:0], DrawY [9:0]

Outputs: is_sword

Description: Uses current keypress to determine whether or not to draw a sword. Uses last keypress to determine which direction it should be drawn and Link's x and y coordinates to determine the relative pixel range.

Purpose: Used to draw Link's sword in color_mapper.sv and generate the hitbox for the sword.

enemy.sv

Inputs: Clk, Reset, frame_clk, DrawX [9:0], DrawY [9:0], room_trans, room [1:0], key [7:0], link_pos_y [9:0], link_pos_x [9:0]

Outputs: Ball_X_Pos [9:0], Ball_Y_Pos [9:0], is_ball, is_alive, link_damage

Description: Used to instantiate an enemy. DrawX and DrawY are used to determine whether or not to draw the enemy at a given pixel based on his x and y coordinates. This enemy respawns after a room transition, and the location of the spawn changes with a given room. Damage calculation occurs here. Link's x and y coordinates, as well as his size (equal to enemy, so we didn't need to include it here) are used to determine whether or not there is a collision. If there is a collision, link_damage is set to high. The keypress is used to determine whether or not Link used his sword on an enemy. If he did, the enemy will take damage. By having is_alive as an

output, we are able to use it in color_mapper.sv as well as other modules to assist in room transition logic.

Purpose: Instantiated as the enemy, this module creates an enemy with basic AI that is key to completing rooms and game progression.

load_enemy.sv

Inputs: clk, reset_h, address [11:0]

Outputs: dataout [23:0]

Description: Holds sprite data for the enemy.

Purpose: Used to draw enemy in color_mapper.sv and stores the RGB arrays for the enemy sprite.

door.sv

Inputs: clk, reset_h, door_enable, DrawX [9:0], DrawY [9:0], top_left_x [9:0], top_left_y [9:0], width [9:0], height [9:0]

Outputs: is_left_door, is_right_door, is_up_door, is_down_door

Description:

Purpose:

victory.sv

Inputs: clk, reset_h, address [11:0]

Outputs: dataout [23:0]

Description: Takes in coordinates DrawX and DrawY and determines whether or not there is a door present. If there is a door present, it outputs which door it is.

Purpose: Used to determine whether or not to draw a door pixel at a given coordinate.

Program Features and Implementation Details

Loading of Link's character sprite as well as the sprite of basic enemies Link needs to fight.

We calculated whether or not to draw a sprite pixel at a pixel coordinate (DrawX and DrawY) as we did in Lab8, checking if the distance from the center was less than the radius of the sprite. We did have to change the drawn pixel values. We created a module for Link's sprite and for the enemy. For the Link and enemy sprites, we created large registers to hold the RGB values for every pixel, and hard coded the values in. We did this by using the PNGToHex code from the final project website to convert the PNGs to text files. We then wrote a script to create a file that contained code to assign the RGB values to the registers we created, and we copied those into our modules. In the Link sprite file, we had 4 sprites for Link, we chose which one to print to the screen based on the current keypress.

Movement

For Link, we changed the movement slightly from Lab 8. The keypress logic largely remains the same, except that when a WASD key is not pressed, Link's movement is halted. In this way, we allow more control for the player. For the enemy, we set the Ball_X_Motion and

Ball_Y_Motion to be updated at 60 Hz, instead of 50 MHz. We also created a counter, which was updated at this frequency as well. Our movement logic dictates that the direction moved depends on the counter, and changes when the counter value leaves a set range. We set our enemy to move up when the counter is less than 32, right when counter is less than 64, down when counter is less than 96, and left when counter is less than 127. We set the counter back to 0 when it reaches 127.

Screen transitions when Link reaches the edge of the map.

Screen transitions and room variance are handled via a state machine which looks to see when all enemies in the room have been defeated and looks at Link's relative position on the screen compared to the door. When Link and the door have overlapping pixels and the player presses the directional input associated with the door (ie "a" key for left door) then Link's relative position on the screen is updated to the other side of the room which emulates moving into another room. This corresponds with all 4 door variations (up, left, right, down) and allows for the dungeon crawler gameplay loop.

Functional interaction between Link and immovable objects on each screen such as bushes and rocks.

This feature was implemented by treating the Link character as a rectangular box on the screen and gaining the relative positions of the four corners of Link's model. After attaining these corners we implemented logic similar to that of the bounce logic in Lab 8 and looked to see what direction Link had entered the object. Once this was determined Link was shifted and bounced in the opposite direction.

Link is able to damage and destroy enemies.

The implementation of this involved giving Link a sword attack which would generate a block of pixels in front of him as well as an active hitbox such that if the enemy would hit it they would die. For this enemy death we then despawned the enemy sprite and toggled a boolean variable which allowed the enemy to damage Link. In order for Link to function as he does in the original game Link must stand still while using his sword and the sword must appear in the direction that Link's sprite is facing. In order to distinguish Link's directional facing we made a variable which remembered the last movement key press from the player and had four available areas of generation for both the image and the active hitbox around Link. When this sword was activated using the "y" key on the FPGA keyboard, this hitbox and sprite loading are enabled and thus allow for the sword hitbox and enemy hurtbox to intersect. When they do the enemy is killed and all of its functions are disabled until the next room.

Enemies can move around and damage/destroy Link.

This feature of the game is a bit similar to the feature above, however, a key distinction is that when the enemy attacks Link simply by walking into him and overlapping their character sprites. Due to this key distinction we simply compared the relative outer edges of the balls that made up the enemy and Link and set conditionals when those relative positions are within one another's boundaries. An important thing to note here is that when Link kills the enemy in the

room the enemy hitbox despawns and thus cannot kill Link any longer. Link dies and deloads his sprite whenever he is hit by the enemy and his health bar at the top of the screen disappears.

End condition (Link collecting a piece of the TriForce).

The end condition for the game relied on the state machine for the room logic such that if the room was the last room within the current build of the game then killing the enemy spawning within the room would award Link with a piece of the TriForce. To give him the TriForce piece we deloaded his character model sprite and instead loaded a new sprite of him holding up the piece to signify victory. We also did not allow him to exit the room, so we would not run into undefined behavior.

Reset capability to bring you to the start of the dungeon.

Due to the implementation of Lab 8 the reset of the character position within the room was already handled. In order to reset the dungeon as a whole, however, this simply involved adjusting the state machine for room and door layout back to the first room when reset is pressed using the FPGA button.


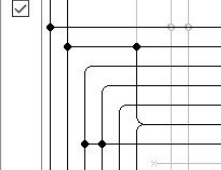
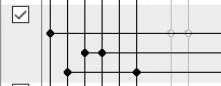
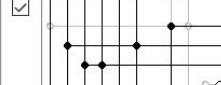
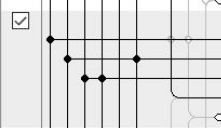
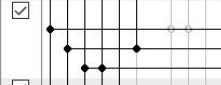
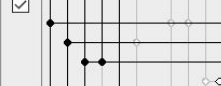
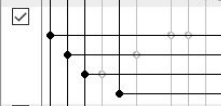
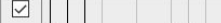
Transitions based upon entry.

Room transition logic is enabled via the variable for room transition going high within the state machine. This variable goes high when Link has defeated all the enemies within the room and identifies when Link's hitbox and the hitbox of the door overlap with one another. When this condition is met and the appropriate direction for each door is pressed, then Link teleports to the other side of the room and the room state is incremented (or decremented if he goes back) by one.

Secret room discovery.

The only additional feature we were able to implement is a result of the state machine which when entering the third room in the dungeon would lock the right door until the player discovered that Link can actually transition through the left wall of the screen to unlock a secret room. This secret room would then be able to unlock the final room within the dungeon.

Platform Designer and Qsys Modules

Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	Tags
<input checked="" type="checkbox"/>		clk_0	Clock Source						
		clk_in	Clock Input	clk	exported				
		clk_in_reset	Reset Input	reset					
		clk	Clock Output	Double-click to export	clk_0				
		clk_reset	Reset Output	Double-click to export					
<input checked="" type="checkbox"/>		nios2_gen2_0	Nios II Processor						
		clk	Clock Input	Double-click to export	clk_0				
		reset	Reset Input	Double-click to export	[clk]				
		data_master	Avalon Memory Mapped Master	Double-click to export	[clk]				
		instruction_master	Avalon Memory Mapped Master	Double-click to export	[clk]				
		irq	Interrupt Receiver	Double-click to export	[clk]			IRQ 0	IRQ 31
		debug_reset_requ...	Reset Output	Double-click to export	[clk]				
		debug_mem_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	# 0x0000_1000	0x0000_17ff		
		custom_instructio...	Custom Instruction Master	Double-click to export	[clk]				
<input checked="" type="checkbox"/>		onchip_memory2_0	On-Chip Memory (RAM or ROM)...						
		clk1	Clock Input	Double-click to export	clk_0				
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk1]	# 0x0000_0000	0x0000_000f		
		reset1	Reset Input	Double-click to export	[clk1]				
<input checked="" type="checkbox"/>		sdram	SDRAM Controller Intel FPGA IP						
		clk	Clock Input	Double-click to export	sdram_pl...				
		reset	Reset Input	Double-click to export	[clk]				
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	# 0x1000_0000	0x17ff_ffff		
		wire	Conduit	sdram_wire					
<input checked="" type="checkbox"/>		sdram_pll	ALTPLL Intel FPGA IP						
		inclk_interface	Clock Input	Double-click to export	clk_0				
		inclk_interface_reset	Reset Input	Double-click to export	[inclk_inte...				
		pll_slave	Avalon Memory Mapped Slave	Double-click to export	[inclk_inte...	# 0x0000_00b0	0x0000_00bf		
		c0	Clock Output	Double-click to export	sdram_pll...				
		c1	Clock Output	sdram_clk	sdram_pll...				
<input checked="" type="checkbox"/>		sysid_qsys_0	System ID Peripheral Intel FPGA...						
		clk	Clock Input	Double-click to export	clk_0				
		reset	Reset Input	Double-click to export	[clk]				
		control_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	# 0x0000_00d0	0x0000_00d7		
<input checked="" type="checkbox"/>		buttons_pio	PIO (Parallel I/O) Intel FPGA IP						
		clk	Clock Input	Double-click to export	clk_0				
		reset	Reset Input	Double-click to export	[clk]				
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	# 0x0000_00a0	0x0000_00af		
		external_connection	Conduit	buttons					
<input checked="" type="checkbox"/>		jtag_uart_0	JTAG UART Intel FPGA IP						
		clk	Clock Input	Double-click to export	clk_0				
		reset	Reset Input	Double-click to export	[clk]				
		avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	# 0x0000_00c8	0x0000_00cf		
		irq	Interrupt Sender	Double-click to export	[clk]				
<input checked="" type="checkbox"/>		keycode	PIO (Parallel I/O) Intel FPGA IP						

Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	Tags
<input checked="" type="checkbox"/>		buttons	Buttons (Parallel I/O) Intel FPGA IP Clock Input Reset Input Avalon Memory Mapped Slave Conduit	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk] [clk]	# 0x0000_00a0	0x0000_00af		
<input checked="" type="checkbox"/>		jtag_uart_0	JTAG UART Intel FPGA IP Clock Input Reset Input Avalon Memory Mapped Slave Interrupt Sender	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk] [clk]	# 0x0000_00c8	0x0000_00cf		
<input checked="" type="checkbox"/>		keycode	P10 (Parallel I/O) Intel FPGA IP Clock Input Reset Input Avalon Memory Mapped Slave Conduit	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk] [clk]	# 0x0000_0090	0x0000_009f		
<input checked="" type="checkbox"/>		otg_hpi_address	P10 (Parallel I/O) Intel FPGA IP Clock Input Reset Input Avalon Memory Mapped Slave Conduit	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk] [clk]	# 0x0000_0080	0x0000_008f		
<input checked="" type="checkbox"/>		otg_hpi_data	P10 (Parallel I/O) Intel FPGA IP Clock Input Reset Input Avalon Memory Mapped Slave Conduit	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk] [clk]	# 0x0000_0070	0x0000_007f		
<input checked="" type="checkbox"/>		otg_hpi_r	P10 (Parallel I/O) Intel FPGA IP Clock Input Reset Input Avalon Memory Mapped Slave Conduit	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk] [clk]	# 0x0000_0060	0x0000_006f		
<input checked="" type="checkbox"/>		otg_hpi_w	P10 (Parallel I/O) Intel FPGA IP Clock Input Reset Input Avalon Memory Mapped Slave Conduit	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk] [clk]	# 0x0000_0050	0x0000_005f		
<input checked="" type="checkbox"/>		otg_hpi_cs	P10 (Parallel I/O) Intel FPGA IP Clock Input Reset Input Avalon Memory Mapped Slave Conduit	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk] [clk]	# 0x0000_0040	0x0000_004f		
<input checked="" type="checkbox"/>		otg_hpi_reset	P10 (Parallel I/O) Intel FPGA IP Clock Input Reset Input Avalon Memory Mapped Slave Conduit	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk] [clk]	# 0x0000_0030	0x0000_003f		

Qsys Modules:

nios2_gen2_0 Block: The nios2 block handles the C code conversion over to system verilog which then is able to be executed within the hardware FPGA board.

Onchip_memory2_0: The memory block acts as available hardware memory for variables needed to execute the C code onto the board. This effectively acts as a compliment to the nios2 processor.

Clk_0 (not shown in screenshot): Serves as the functional clock for the entire system and acts as a reference for the other modules.

sdram: This block allows us to get available SDRAM on the FPGA board since on-chip memory is too small for the available program to be stored and updated successfully. SDRAM is useful due to its quick processing time and low update and output delay.

sdram_pll: This block is the method to account for the small delays within the transfer of the data in and out of the SDRAM and acts as a separate clock for the system.

sysid_qsys_0: This block verifies the correct transfer of the software and hardware by looking back and forth between the C code and SystemVerilog to assure the data transfer is done in the correct format.

buttons_pio: The inputs are the buttons on the FPGA device. This PIO allows for bidirectional data transfer from the FPGA to software and visa versa.

jtag_uart_0: Allows for terminal access for use in debugging the software.

keycode: Reads data, updates the USB chip and then sends it to the software for logic instruction.

otg_hpi_address: PIO that allows the desired address in memory of the SoC to be found that is sent from the software to the FPGA.

otg_hpi_data: PIO that allows for cross transfer of data from the FPGA to software and the other way around. This PIO has a width of 16 bits which allows for sizable data transfer between the two.

otg_hpi_r: PIO that allows the enable bit to read from the memory of the SoC that is sent from the software to the FPGA.

otg_hpi_w: PIO that allows the enable bit to write to the memory of the SoC that is sent from the software to the FPGA.

otg_hpi_cs: PIO that allows the enable bit turn on and off the memory of the SoC that is sent from the software to the FPGA.

otg_hpi_reset: PIO that allows for the reset of the memory of the SoC sent from the software to the FPGA.

Simulation



The image above shows the simulation waveform for Link in the first room where he must exist through the left side of the room. As seen in the picture when the enemy is alive Link bounces off of the left boundary of the screen and when the enemy dies the transition variable is set to high and Link teleports to the right side of the room.

Design and Resource Table

LUT	20487
DSP	30
Memory (BRAM)	135808
Flip-Flop	2322
Frequency	65.12 MHz
Static Power	106.74 mW
Dynamic Power	89.98 mW
Total Power	266.44 mW

Timeline Predictions and Results

Week 1 (4/3 - 4/9): Composition of project proposal as well as incorporating help from TA's. Finish Lab 9 Week 2 and begin sprite loading of Link and basic enemies.

We succeeded at loading 3 Link sprites, facing up, down, and to the side. We also loaded an enemy sprite, and put all of them on the screen. We modified the existing Lab 8 movement code to allow Link to move and stop according to active key presses, and we emulated it using a counter for the enemy. The enemy moves in a square, and changing Link's movement direction enabled sprite turnaround.

Week 2 (4/10 - 4/16): Finish Link and enemy character sprites and begin screen scrolling capability as well as Link vs enemy interaction including character and enemy death.

This week, we focused on room transition logic and debugging sprite loading. This week was mainly dedicated to creating a sword module. The sword took significant time and debugging, as we struggled to animate it in the desired direction and location (in front of Link depending on the direction he was facing). After that, we worked to include damage calculation

and a health stat. We also began working on our door sprites and room transitions. Loading the door was easy, but creating room transition logic and tying it into our enemy spawns was more challenging than we thought.

Week 3 (4/17 - 4/23): Demo character sprites and screen scrolling (Mid-Project Checkpoint Demo). Finish Link vs enemy and start interaction with stationary objects as well as unveiling secret rooms.

This week we focused on getting ready for our demo. We debugged any sprite issues, optimized the loading to improve compile time (we originally took 30 minutes to compile, we reduced this to 7), and started on room transitioning. Our health system broke our game, so we commented it out, leaving it to be done later. We were able to transport Link to the other side after walking through a door from any direction, but did not integrate that into enemy spawning and health reloading this week. By demo time we achieved our demo goals, but did not have as much done with regards to health and death as we planned.

Week 4 (4/24 - 4/30): Implementing several of the nonessential functions and cleanup of the essential ones for the final demo.

We began work on stationary objects, as well as debugged enemy movement to meet our design constraints. We had issues with the enemy turning around too fast, and it took us some time and unhelpful advice before we ended up at the desired speed. We got close to creating a stationary object that Link could not move through, but we were only able to deflect Link, instead of emulating the wall behavior in Lab 8. We finished up every essential feature, with the exception of stationary objects being bugged and the end condition.

Week 5 (5/1 - 5/8): Cleanup of any not fully functioning modules as well as adding music. Demoiing the project this week as well as composition of the final report.

This week we ended up having to do a lot of work for other classes, so we simply added the end condition (Link holding up the Triforce after beating the enemy in the last room) and fixed the object collision logic.

Reflection

Getting to make a simple dungeon from The Legend of Zelda was very enjoyable for us, and taught us how complex various features are to design. After making a health system and damage calculation, we found a greater appreciation for games that have invincibility frames and combos. We ended up making a very uniform room design for our dungeon, such that we only had minor changes that were simple to implement. One key takeaway in terms of game design was the difference of game hitboxes versus the loading of the character sprite on the screen. After encountering numerous errors within the sword loading for damaging enemies we learned this practice within office hours and implemented it within our project.

Conclusion

Given more time, we would have liked to expand on the features we implemented, such as variation of rooms and different enemies. By the time we finished our code for immovable objects, we needed to dedicate time to our other classes. We would have liked to create a module for immovable objects, with similar changes in location as we did the enemy spawns based on room. We were interested in creating different enemies, but scrapped the idea due to not wanting to create new AI due to complexity. Lastly, we considered adding invincibility frames to a character or enemy after they were hit, but we wanted to finish our main features beforehand and ended up not having time. We created a simple implementation of these features but would have liked to make our game more complex.

There are some small changes we wanted to include. We considered implementing a life system, such that Link has 3 tries to complete the game before resetting. Link would respawn in the same room he died in, but away from the enemy that killed him. After he lost all of his lives, he would stay down until the game was reset.

Despite not being able to flesh the game out as much as we wanted, we believe that we laid out the groundwork such that we could achieve our original goals without drastically reworking our code. We learned a lot about making a game fair instead of just hard, as well as the work that goes into creating user friendly mechanics. We created a very basic dungeon, and are proud that we applied what we learned in this class to make something we enjoy.

Code References

We used the Rishi's helper tool given to us from the final project page to convert the sprite PNGs to a text file we could use.

We also used the entirety of our Lab 8 code as a starting point. We only modified `color_mapper.sv`, `lab8.sv`, and `ball.sv`, and added additional modules.

Link's sprites: <https://www.wiizelda.net/ALinktothePast-Sprites.php>

En1qAQ\emy sprite: <http://wiki.zfgc.com/Images>

Rishi Helper Tools: <https://github.com/Atrifex/ECE385-HelperTools>

Nios II Code

We used the `usb` and `io_handler` files from Lab 8 to monitor keypresses for our game. We did not modify any C code in this project, so the NIOS II code functions exactly as it did in Lab 8.

Code used to create sprite file code

We used the code below to take the text files we obtained from Rishi's PNGToHex script and make SystemVerilog code we could copy and paste into our sprite modules. I also used a basic find/replace tool to replace the `RGB = x000000` with the background pixels.

```
#include <stdio.h>
```

```
#include <stdlib.h>
#define INPUT_FILE "link.txt"
#define OUTPUT_FILE "link_sprite.txt"

int main()
{
    FILE * fp;
    FILE * outFile;
    char * line = NULL;
    size_t len = 0;
    ssize_t read;
    int lineNum = 0;
    fp = fopen(INPUT_FILE, "r");
    outFile = fopen(OUTPUT_FILE, "w");
    if (fp == NULL)
        exit(EXIT_FAILURE);
    while ((read = getline(&line, &len, fp)) != -1) {
        fprintf(outFile, "data1[%d] = 24'h%s", lineNum, line);
        lineNum++;
    }
    fclose(fp);
    fclose(outFile);
    if (line)
        free(line);
    exit(EXIT_SUCCESS);
    return 0;
}
```