ECE 385

Spring 2021

Final Project

Flappy Fish

Nicholas Logan & Abdulganiy Sunmola

Section ABE / 03-29-2021

TA: Wenjie Pan

Introduction

Our game, Flappy Fish, recreates the popular mobile game Flappy Bird. Much of the game mechanics is the same, with pipes coming at you from the right side of the screen, coins appearing at random, and frustrating controls. However, to add a little originality and uniqueness, we added the ability to glide down to better maneuver between the pipes. To counteract any ease of playing, we've also added 3 difficulty levels ranging from normal, impossible, to more impossible. This game is a continuation from lab 6.2, where we built the SoC on the NIOS-II processor.

Game Logic

In the Flappy Fish game, you will be controlling a fish which will either move up (using the W key) or down (using the S key). The three difficulties determine the speed of the game, which is indicated by the color of a circle icon located in the top right of the game screen (green = normal, orange = impossible, and red = more impossible) . A point is awarded to the player whenever the fish passes a pipe. To begin the game, you click key KEY[1] on the FPGA board when the game is in the off state (more details later). To set the difficulty, you can use the keyboard and select from numbers 1-3. The game will end once the fish hits either the pipe or the top/bottom of the borders of the game.

Implemented Features / Timeline

We were able to complete all of the baseline features in our game, on top of a few additional ones. For our first week, much of it was spent planning, researching, and coming up with pseudo code for our game logic. From there, we first started with the fish and gravity. To implement gravity, we altered the code for ball.sv and added a Counter variable that would increment on each positive edge of the clock. When Counter reaches 5, we increase the speed of the fish by 1, which simulates the motion of downward acceleration. From there, we moved to the pipe generation. To get pipes to generate at random heights, we again used a Counter variable, but this time increment it by a unique prime number for each pipe. The unpredictability of the Counter variable allowed us to simulate randomness. Next, coins were placed in between the pipes and collision logic was added so when the fish collided with the coin they would disappear. We used the same collision logic to change the game state when the fish hit a pipe or went outside the boundaries. Flappy Fish has 2 game states, off and on. This is shown in the figure below. We opted for this instead of a menu as we had difficulty with the sprite generation, and without text this would not be a feasible option. To keep track of the score, we use the FPGA board's HEX display and LEDs. This is done through software on the SoC. The left two HEX displays the current high score, while the right two displays the current score. The LEDs reflect the same thing as the right HEX display. Finally, our last feature was the difficulty levels. By pressing 1, 2, or 3 we can change the difficulty level of the game, which in our case is the speed at which the fish flies through the pipes.

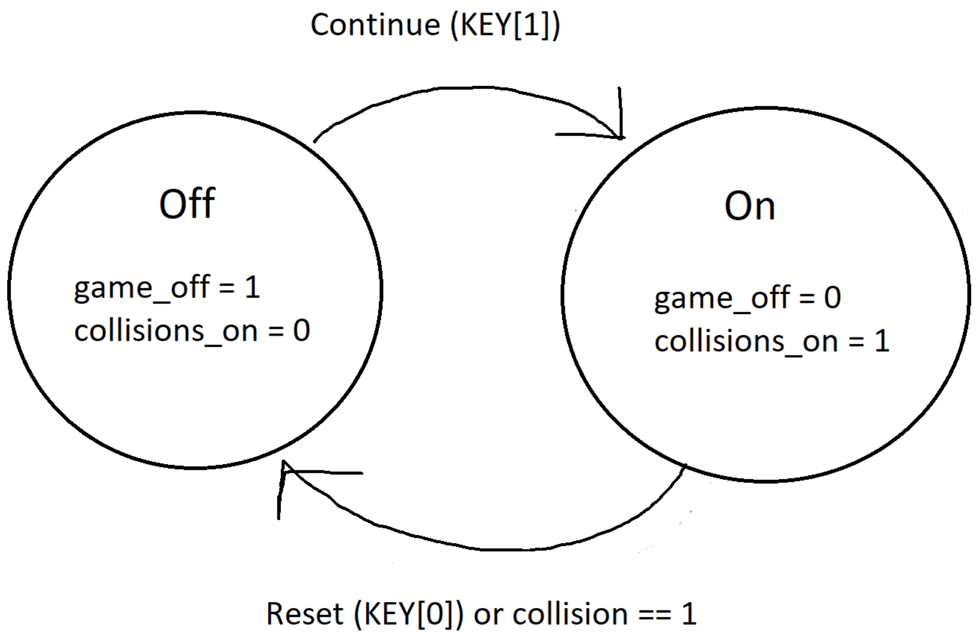
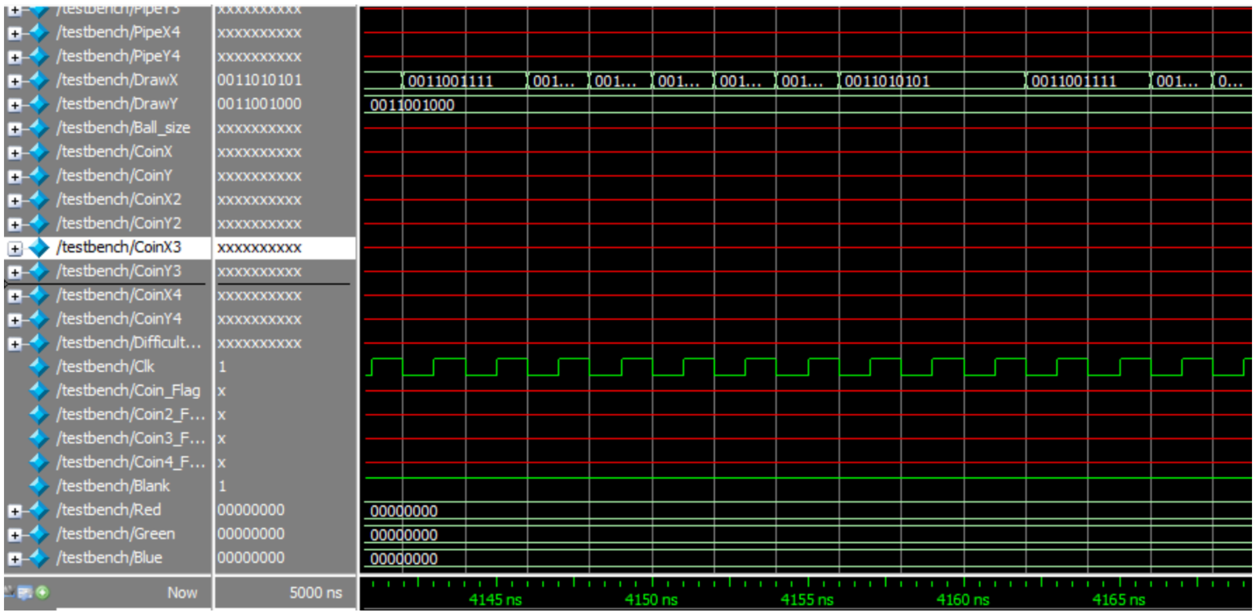


Figure 1: Game logic state machine

Challenges

Unfortunately, some of the advanced features we planned on implementing into our game could not be accomplished. The most challenging one was the **sprite data** we needed to make our pipes, fish and background look more realistic. We successfully converted the background and the fish to hexadecimal data which represents the index for the palette that will draw the colors of the animation in the color_mapper.sv file. We used the png_to_palette_resizer tool provided to us as a resource to convert the png files of the animation to the hexadecimal data. Before passing it through the tool, we picked out 6 unique colors from the png files to reduce memory as we saved the sprites on the chip and made the appropriate calculations to determine the input and output allocation space, combinational logic, and memory location in the ROM sv files.

For some unknown reason, we were unable to get the sprite to show up on the screen, it just appeared as a white box. We made a test bench to make sure the color_mapper reads the RGB data so as to ensure there were no errors in reading the data, we also double checked the hexadecimal data to make sure. We also tested out other variables individually such as DrawX and DrawY to make sure our variables are working properly. Instead of using sprites, we decided to manually input the colors for each of the objects such as the fish, background, coins, pipes and others in the game using computational logic in the always_comb in the RGB_Display section.



```
/testbench/PipeY3        xxxxxxxxxx
/testbench/PipeX4        xxxxxxxxxx
/testbench/PipeY4        xxxxxxxxxx
/testbench/DrawX         0011010101   0011001111  001...  001...  001...  001...  001...  0011010101   0011001111  001...  0...
/testbench/DrawY         0011001000   0011001000
/testbench/Ball_size     xxxxxxxxxx
/testbench/CoinX         xxxxxxxxxx
/testbench/CoinY         xxxxxxxxxx
/testbench/CoinX2        xxxxxxxxxx
/testbench/CoinY2        xxxxxxxxxx
/testbench/CoinX3        xxxxxxxxxx
/testbench/CoinY3        xxxxxxxxxx
/testbench/CoinX4        xxxxxxxxxx
/testbench/CoinY4        xxxxxxxxxx
/testbench/Difficult...  xxxxxxxxxx
/testbench/Clk           1
/testbench/Coin_Flag     x
/testbench/Coin2_F...    x
/testbench/Coin3_F...    x
/testbench/Coin4_F...    x
/testbench/Blank         1
/testbench/Red           00000000     00000000
/testbench/Green         00000000     00000000
/testbench/Blue          00000000     00000000
Now                      5000 ns      4145 ns   4150 ns   4155 ns   4160 ns   4165 ns
```

Figure 2: Testbench simulation displaying RGB for different positions within the sprite

SV Modules

Module: lab62
Inputs: MAX10_CLK1_50, [9:0] SW, [1:0] KEY, [15:0] DRAM_DQ, [15:0] ARDUINO_IO, ARDUINO_RESET_N
Outputs: [9:0] LEDR, {[7:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5 }, [12:0] DRAM_ADDR, [1:0] DRAM_BA, DRAM_CAS_N, DRAM_CKE, DRAM_CS_N, [15:0] DRAM_DQ, [1:0] DRAM_DQM, DRAM_RAS_N, DRAM_WE_N, DRAM_CLK, VGA_HS, VGA_VS, [3:0] VGA_R, [3:0] VGA_G, [3:0] VGA_B
Description: The module connects all of the components created in the Platform Designer to the wires to the physical switches, keys, and LEDs. All of the SDRAM connections, as well as I/O connections are passed in through this module. One thing we did have to change here was DRAM_LDQM and DRAM_UDQM, as well as the corresponding pin connections.
Purpose: Top-level module that integrates the NIOS II system with the rest of the hardware.

Module: HexDriver
Inputs: [3:0] In0
Outputs: [6:0] Out0
Description: This module converts 8-bit binary numbers from registers into hexadecimal by hardcoding them.
Purpose: Since the FPGA has limited LED's to display the output of the adder results, we use this module to display all our values.

Module: color_mapper

Inputs: {[9:0] BallX, BallY, PipeX, PipeY, PipeX2, PipeY2, PipeX3, PipeY3, PipeX4, PipeY4, DrawX, CoinX, CoinY, CoinX2, CoinY2, CoinX3, CoinY3, CoinX4, CoinY4, Difficulty_Flag }, Clk, Coin_Flag, Coin2_Flag, Coin3_Flag, Coin4_Flag, Blank
Outputs: { [7:0] Red, Green, Blue }
Description: The module takes in the location of each object and the location of the current draw pixel to determine what object is present and draws that color onto the screen. For the colors to appear properly, we take in the Blank flag from the VGA so we are not drawing in an invalid state.
Purpose: This module outputs everything to the VGA monitor including the 4 pipe instances, 4 coin instances, fish , sky, and ground.

Module: coin
Inputs: Reset, frame_clk, coin_stop, {[9:0] PipeX, PipeY }
Outputs: {[9:0] CoinX, CoinY}
Description: The location of the coin is determined by the location of the corresponding pipe, and moves at the same speed.
Purpose: This module controls the movement for the coin.

Module: collision_checker
Inputs: {[9:0] BallX, BallY, PipeX, PipeY, PipeX2, PipeY2, PipeX3, PipeY3, PipeX4, PipeY4, Ball_size, CoinX, CoinY, CoinX2, CoinY2, CoinX3, CoinY3, CoinX4, CoinY4}
Outputs: Collection_Flag, Collection2_Flag, Collection3_Flag, Collection4_Flag, collision
Description: The collision logic is similar to the drawing logic, in that it checks a position on the fish and sees what object is within the boundaries. We check three points on the fish : top, leftside, and bottom. This allows for less logic but all of the functionality. If the fish collides with the top or bottom of the screen or with a pipe, the collision variable is set. This notifies the game logic that it needs to move to the off state. However, if the fish collides with a coin, we instead set the collection flag of the corresponding coin and stop drawing it to the screen.
Purpose: Takes in position of all objects and sees if the fish is colliding with anything.

Module: pipe
Inputs: Reset, frame_clk, pipe_stop, {[9:0] pipe_num, pipe_speed, PipeX0, PipeY0}
Outputs: New_Pipe, {[9:0] PipeX, PipeY}
Description: The pipe moves from right to left at a constant speed. There are four pipe instances, and once a pipe passes the left boundary, we move it back to the right side with a different height. This makes the illusion that the pipe is infinitely generated.
Purpose: This module controls the movement for a pipe.

Module: vga_controller
Inputs: Clk, Reset

Outputs: hs, vs, pixel_clk, blank, sync, {[9:0] DrawX, DrawY}
Description: This consists of various computational logic that generates 525 X 800 pixels and uses the pixel clock (set at 25MHz) and the vertical and horizontal pulse signals to display the intended pixels required
Purpose: This gives precise instructions on how and which pixels the electron gun paints, the outputs for this module are essential inputs for the color_mapper module.

Module: bird
Inputs: Reset, frame_clk, bird_stop, [7:0] keycode
Outputs: {[9:0] BirdX, BirdY, BirdS}
Description: The bird (or fish) is the only module that takes user input. Thus, we pass in the keycode generated by the keyboard unlike the pipe or coin modules. If the user presses W, the internal velocity variable is set to -5. If the user presses S, this variable is set to 1. If the user holds S, this allows the bird to glide down. Furthermore, every 5 clock cycles we increment the speed by -1 to implement gravity. Finally, resetting centers the bird to the middle of the screen towards the left side.
Purpose: This module controls the movement for the player-controlled bird.

Module: frameRAM
Inputs: [8:0] read_address, Clk
Outputs: [5:0] data_Out
Description: This contains the input we calculated by using the ceiling of the logarithm of the memory obtained from multiplying the resolution of the fish image. It also has the output which are the unique colors we picked from the palette tool.
Purpose: This is the read only memory file that reads the hexadecimal data of the fish and sends the index of the palette to the color_mapper sv file to draw the image on the screen.

Module: BackgroundRAM
Inputs: [12:0] read_address, Clk
Outputs: [5:0] data_Out
Description: This contains the input we calculated by using the ceiling of the logarithm of the memory obtained from multiplying the resolution of the background image. It also has the output which are the unique colors we picked from the palette tool.
Purpose: This is the read only memory file that reads the hexadecimal data of the background and sends the index of the palette to the color_mapper sv file to draw the image on the screen.

Module: lab62_soc
Inputs: clk_clk, [1:0] key_external_connection_export, reset_reset_n,, [15:0] sdram_wire_dq, spi0_MISO, usb_gpx_export, usb_irq_export

Outputs: [15:0] hex_digits_export, [7:0] keycode_export, [13:0] leds_export, sdram_clk_clk, [12:0] sdram_wire_addr, [1:0] sdram_wire_ba, sdram_wire_cas_n, sdram_wire_cke, sdram_wire_cs_n, [1:0] sdram_wire_dqm, sdram_wire_ras_n, sdram_wire_we_n, spi0_MOSI, spi0_SCLK, spi0_SS_n, usb_rst_export
Description: This is an automatically generated module that instantiates all the various IP blocks that we used and interconnected (all of these are shown in the system level block diagram) in our platform designer.
Purpose: This acts as some form of top-level for the IP blocks and helps integrate it and system verilog.

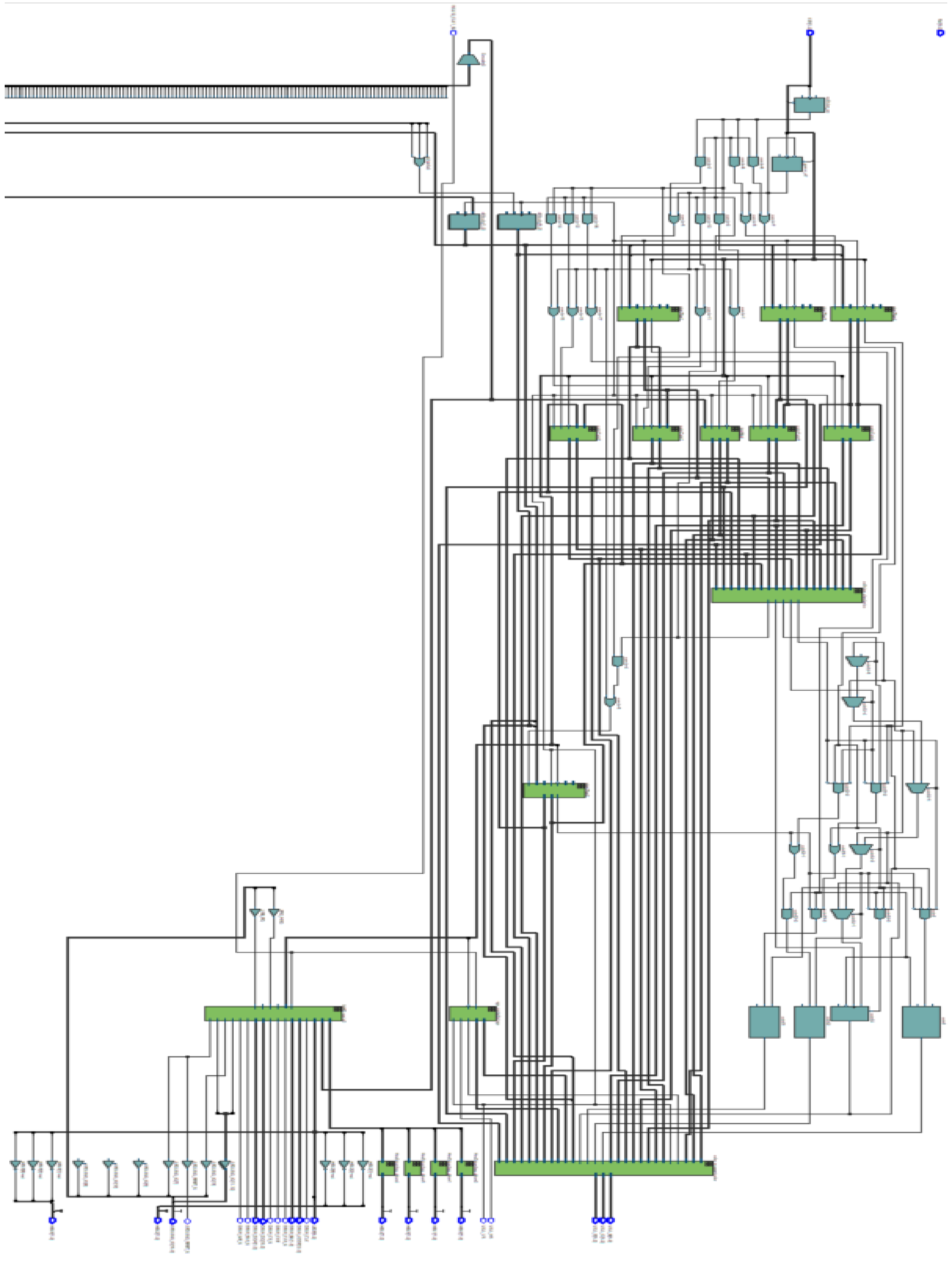
Design Statistics Table

LUT	5901
DSP	19
Memory (BRAM)	55296
Flip-Flop	2645
Frequency	138.85 MHz
Static Power	96.18mW
Dynamic Power	0.70mW
Total Power	106.18mW

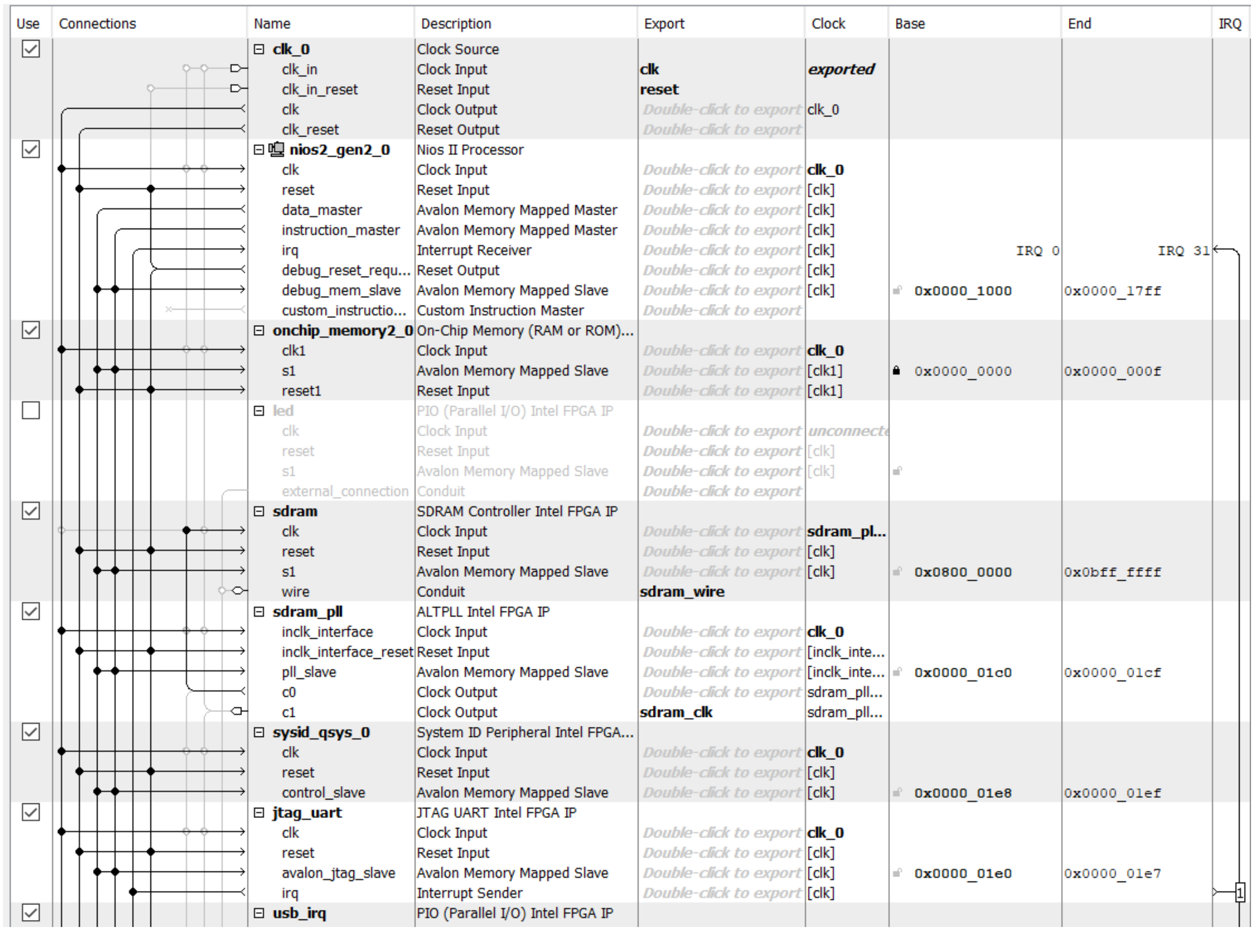
Conclusion

Our final project encompassed all we have learnt so far in this class and other resources we learnt as we researched how to approach the project. If our sprite data had worked , it would have been implemented using the on-chip memory so as to avoid the complexity of the SDRAM process of storing the data. Overall, our game came out as we had proposed in our project proposal but there's always room for improvement and anyone is welcome to use our project as a starting point to further improve the game. This was a great learning experience and helped to express our imaginative side.

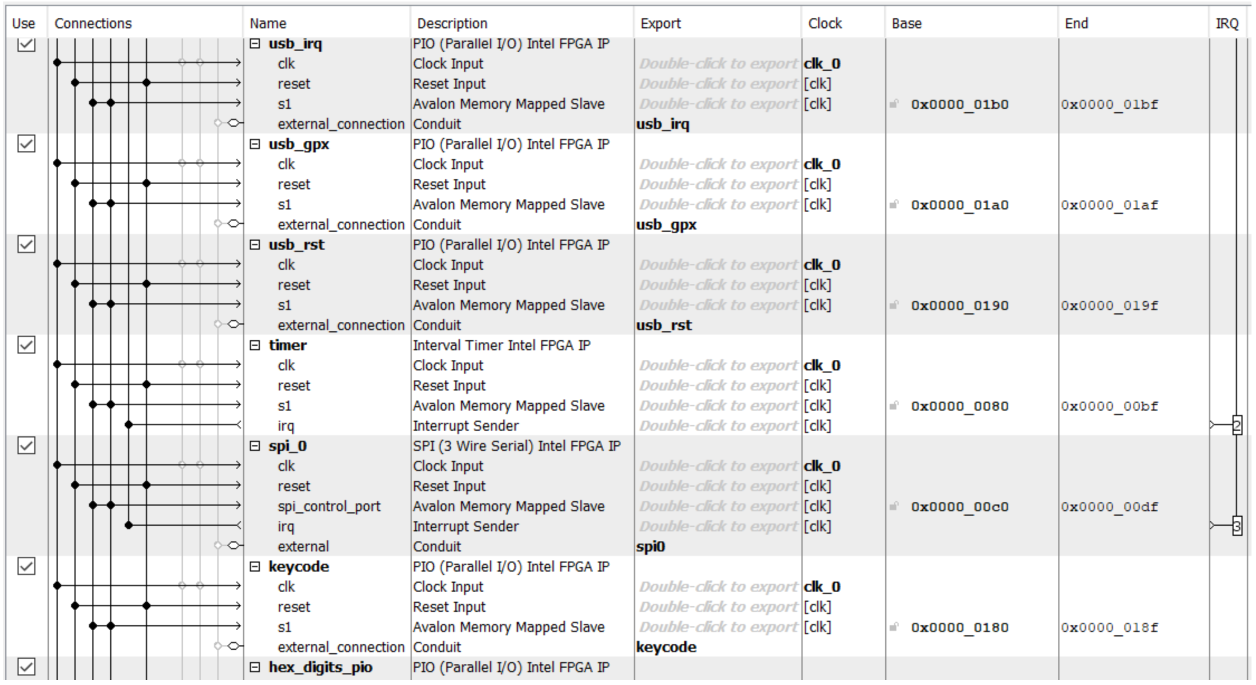
Top Level Block Diagram



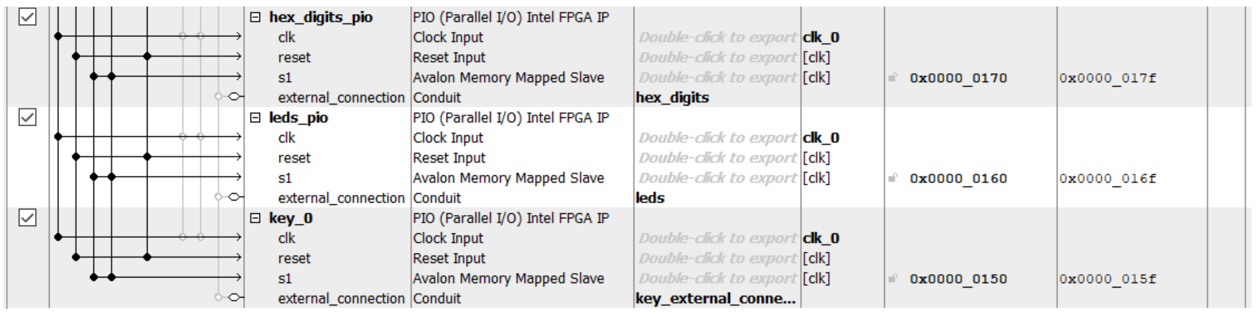
System Level Block Diagram



Use	Connections	Name	Description	Export	Clock	Base	End	IRQ
✓		⊟ **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				
✓		⊟ **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				
✓		⊟ **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]			
☐		⊟ led	PIO (Parallel I/O) Intel FPGA IP					
		clk	Clock Input	Double-click to export	*unconnecte*			
		reset	Reset Input	Double-click to export	[clk]			
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]			
		external_connection	Conduit	Double-click to export				
✓		⊟ **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]	0x0800_0000	0x0bff_ffff	
		wire	Conduit	**sdram_wire**				
✓		⊟ **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_01c0	0x0000_01cf	
		c0	Clock Output	Double-click to export	sdram_pll...			
		c1	Clock Output	**sdram_clk**	sdram_pll...			
✓		⊟ **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_01e8	0x0000_01ef	
✓		⊟ **jtag_uart**	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_01e0	0x0000_01e7	
		irq	Interrupt Sender	Double-click to export	[clk]			1
✓		⊟ **usb_irq**	PIO (Parallel I/O) Intel FPGA IP					



Use	Connections	Name	Description	Export	Clock	Base	End	IRQ
✓		⊟ **usb_irq**	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_01b0	0x0000_01bf	
		external_connection	Conduit	**usb_irq**				
✓		⊟ **usb_gpx**	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_01a0	0x0000_01af	
		external_connection	Conduit	**usb_gpx**				
✓		⊟ **usb_rst**	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_0190	0x0000_019f	
		external_connection	Conduit	**usb_rst**				
✓		⊟ **timer**	Interval Timer 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_0080	0x0000_00bf	
		irq	Interrupt Sender	Double-click to export	[clk]			2
✓		⊟ **spi_0**	SPI (3 Wire Serial) Intel FPGA IP					
		clk	Clock Input	Double-click to export	**clk_0**			
		reset	Reset Input	Double-click to export	[clk]			
		spi_control_port	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_00c0	0x0000_00df	
		irq	Interrupt Sender	Double-click to export	[clk]			3
		external	Conduit	**spi0**				
✓		⊟ **keycode**	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_0180	0x0000_018f	
		external_connection	Conduit	**keycode**				
✓		⊟ **hex_digits_pio**	PIO (Parallel I/O) Intel FPGA IP					



⊟ hex_digits_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_0170	0x0000_017f
external_connection	Conduit	hex_digits			
⊟ leds_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_0160	0x0000_016f
external_connection	Conduit	leds			
⊟ key_0	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_0150	0x0000_015f
external_connection	Conduit	key_external_conne...			

Platform Designer Blocks

Blocks: clk_0
Description: This is the module that takes care of the clock signal at which all the other IP blocks perform their operations. It is set at 50MHz for the FPGA.

Blocks: nios2_gen2_0
Description: This acts as the CPU that processes all the information required to perform this entire lab's expectation (i.e moving the ball through the keyboard). We use the Nios II/e for this.

Blocks: onchip_memory2_0
Description: This contains all the addresses and memory we need to call and save to perform the lab. This uses 16 chips, each having a data width of 32 bits producing a total memory size of 512Mbit.

Blocks: sdram
Description: This acts as an interface for the software side of the lab. The FPGA uses this to interface with various buses.

Blocks: sdram_pll
Description: This module creates the clock signal for the sdram.

Blocks: sysid_qsys_0
Description: This is used to verify that the software is loaded correctly into the appropriate hardware.

Blocks: jtag_uart
Description: This is an interface protocol that integrates the FPGA's switches and runs the accumulator program.

Blocks: usb_irq
Description: This acts as an interrupt key for the inputs from the keyboard

Blocks: usb_gpx
Description: This is an I/O module that simulates a multiplexer.

Blocks: usb_rst
Description: This module helps reset any inputs form the usb port/ keyboard.

Blocks: timer
Description: This is used to keep track of the keyboard input time-outs.

Blocks: spi_0
Description: This is described in detail in the report above.

Blocks: keycode
Description: This interprets the buttons pressed on the keyboard as 8 bit sized values

Blocks: hex_digits_pio
Description: This displays the keycode outputs on the hex display

Blocks: leds_pio
Description: This runs the leds according to the keycode outputs.

Blocks: key_0
Description: Resets the game and switches the game state to 0.