ECE 385

Fall 2020

Final Project

Two-Player Pacman

Deonna Flanagan, Murat Altindag

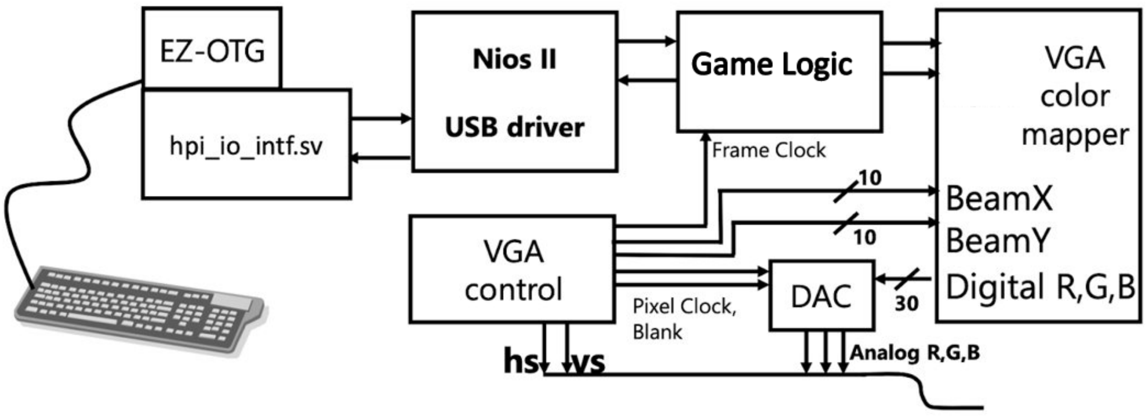
Section ABC Friday

Gene Shiue

Introduction

For our final project, we proposed to make a game of Pacman on the FPGA using our lab 8 code as a starting point. The Pacman is controlled by inputs from a USB keyboard (A-left, D-right, W-up, S-down) and the game is displayed on a VGA monitor. Upon building the game, however, we ran into bugs and changed the game a bit so it still worked (see "Bugs and Measures Taken" section). The game we developed is a two player game, consisting of two Pacman, the first is the yellow/main Pacman and the second is the red/evil Pacman. Both players use inputs from the keyboard to move their pacman. The objective for the yellow Pacman is to eat as many beans as it can before it is eaten by the red Pacman and the objective of the red Pacman is to eat the yellow pacman before he eats all the beans. The beans disappear as the yellow Pacman collides with or "eats" them. The score to keep track of the food eaten is stored in a variable, but we ran out of time to display it on the HEX display. Sprites are used to display each of the pacman, one yellow (xFFFF00) and one red (xFF0000), the walls are displayed in cyan (x2EB9F7) and the beans are displayed in a pink color (xDD44DD).

Written Description of Two-Player Pacman System



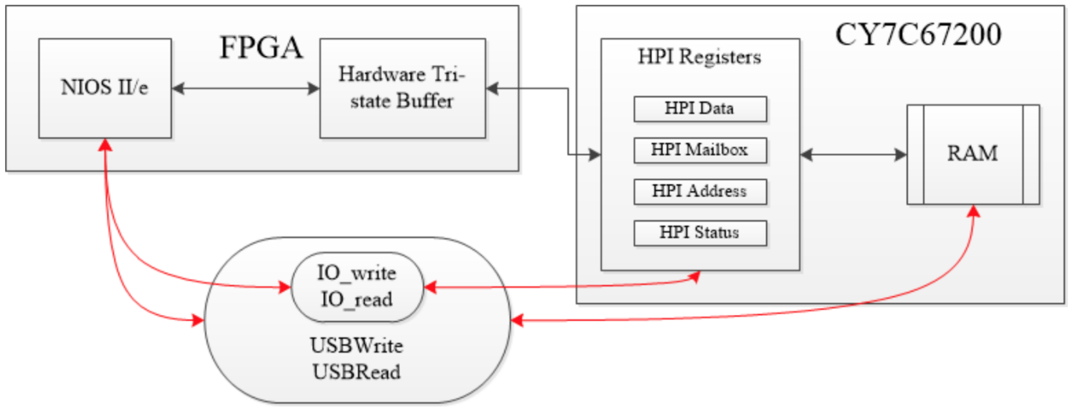
Written description of entire two-player Pacman system

We primarily built off of our lab 8 which consists of VGA components, MAX3421E USB chip and the SPI bus.

The VGA components are the SystemVerilog files that describe the display of the system. It takes inputs from the game logic to determine what is drawn where. The ball.sv module determines the movement of the main pacman, while two.sv determines the movements of the second pacman. When one of the pacman is being drawn, it reads from the sprite saved in memory to display the pacman. The walls and the food pellets are hard coded in color mapper. VGA_controller.sv regulates the display of the object, which pixels will be used, how the object will be shaped and where it will be located etc.

MAX3421E is the USB chip we are using for this lab which takes the usb inputs from the keyboard, sends them to the bus and receives the interrupt and reset outputs. It makes it possible for us to give directional commands to the Pacman.

SPI bus is the serial peripheral bus we are using in this lab. It connects the other two components and interacts with the top level to implement the functions.



Description of how NIOS interacts with both the MAX3421E USB chip and VGA components

NIOS includes and manages the processor, RAM, SPI, timing units and the PIOs of the system. It is connected to the same clock and reset inputs as well as using other inputs and outputs to interact with the USB chip and the VGA components. It has the USB inputs coming in from the MAX3421E, uses the SPI to do necessary modifications and outputs it to both the USB and the VGA. This way, the USB can get interrupted, reset, re-initialized etc. while the VGA can display the results of the commands coming from the MAX3421E chip.

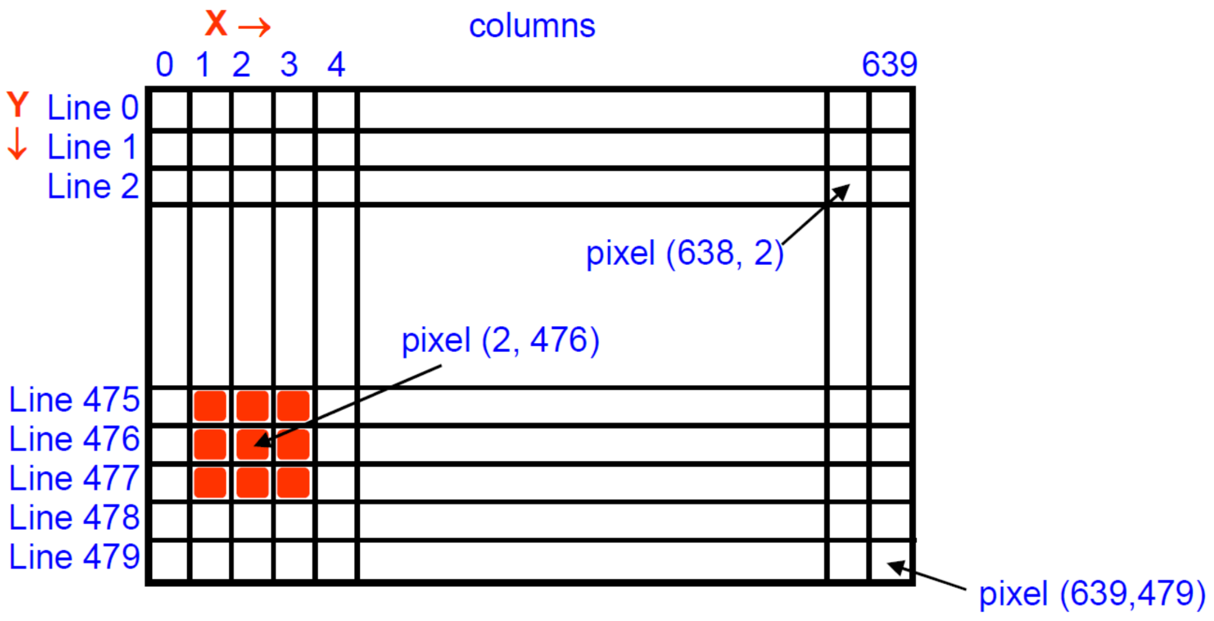
Written description of SPI protocol

The Serial Peripheral Interface is a synchronous serial communication bus that is generally used in embedded systems, just like the system of this lab. It is composed of a memory mapped master and at least one slave that interacts with each other and other sources inside the system. Master sets up the frame for reading and writing and the slaves are selected through the *slaveselect* register.

SCLK is the serial clock of the system.
MOSI is the Master-out, Slave-in command.
MISO is the Master-in, Slave-out command.
SS is the slave selection command.

Detailed description of VGA operation and how Ball, Color Mapper, and VGA controller modules interact

The VGA, video graphics array, is the type of connector port we use to display graphics on the monitor from the FPGA board. We can imagine the display working like an electron beam starting in the top left corner of the screen moving vertically until it reaches the end of the screen, then down one pixel to the far right again, similar to how a typewriter prints lines. X and Y coordinates are assigned with increasing X to the right and increasing Y downwards.



The Ball module tracks the position of the Pacman (BallX and BallY). It takes in the current position of the Pacman and the keycode (which can move the ball either 1 position vertically or 1 position horizontally). The Color Mapper module performs rendering and coloring of objects and outputs RGB values to the monitor. It takes in the X and Y coordinates of the center of the Pacman from the Ball module and creates a box around the coordinates that will represent the ball. When the beam crosses the box coordinates (within (center + box size) and (center - box size)), RGB will be set to (1023, 0, 0) to color the ball red. When the beam crosses pixels outside of (center + box size) and (center - box size), a value for the background color will be assigned for RGB such as (0,0,1023) for blue. This would create a red square with a blue background. To obtain a circle of radius r around the coordinates (x_0, y_0), we can determine if the beam coordinates (x, y) are within the circle if $(x - x_0)^2 + (y - y_0)^2 \leq r^2$. The pixel RGB values produced by the function are displayed using the VGA controller which gets sync pulses and sets the horizontal and vertical parameters.

Written Description of Software Components

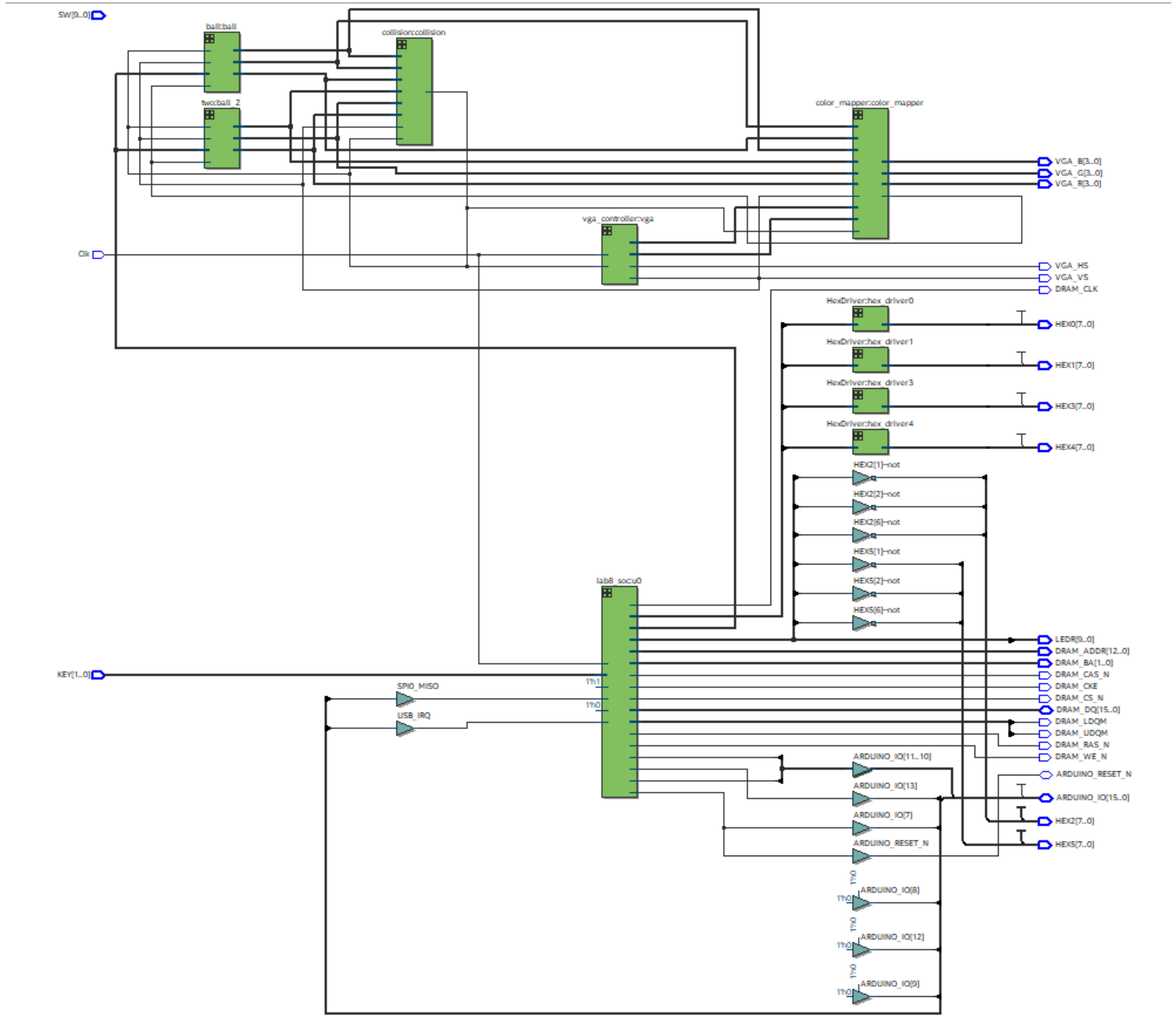
The software components consisted of the code to read in usb inputs from the keyboard necessary to move the Pacman. The movements of both Pacman rely on this interface. Data is written and read from specific addresses which allows for communication between the keyboard and the FPGA.

Written Description of Hardware Components

Most of the game was implemented in hardware, from the basic game logic to the VGA display. The sprites for the Pacman were stored in memory along with the wall position and food positions that were hard coded in. Signals are stored in memory that are used to determine what is being drawn at each pixel so we know the corresponding RGB values. The coordinates of Pacman one and Pacman two are done in hardware. For each Pacman we have to read in the sprite from memory to display it. For the wall or food, we see if the positions match and, if so, the appropriate color is drawn. To keep track of whether each piece of food has been eaten or not, we use a register. This register, in addition to keeping track of if each food is eaten or not, has a variable for score that will be set when each food pellet is eaten. This score can be set to display on the HEX display of our FPGA. For movement of each Pacman, we have the Pacman follow different rules. The yellow Pacman follows similar rules to the original Pacman: the Pacman can eat the beans and can’t go through walls. The red Pacman has some unique characteristics where it can go through walls and isn’t affected by the food, as its main objective is simply to eat the yellow Pacman.

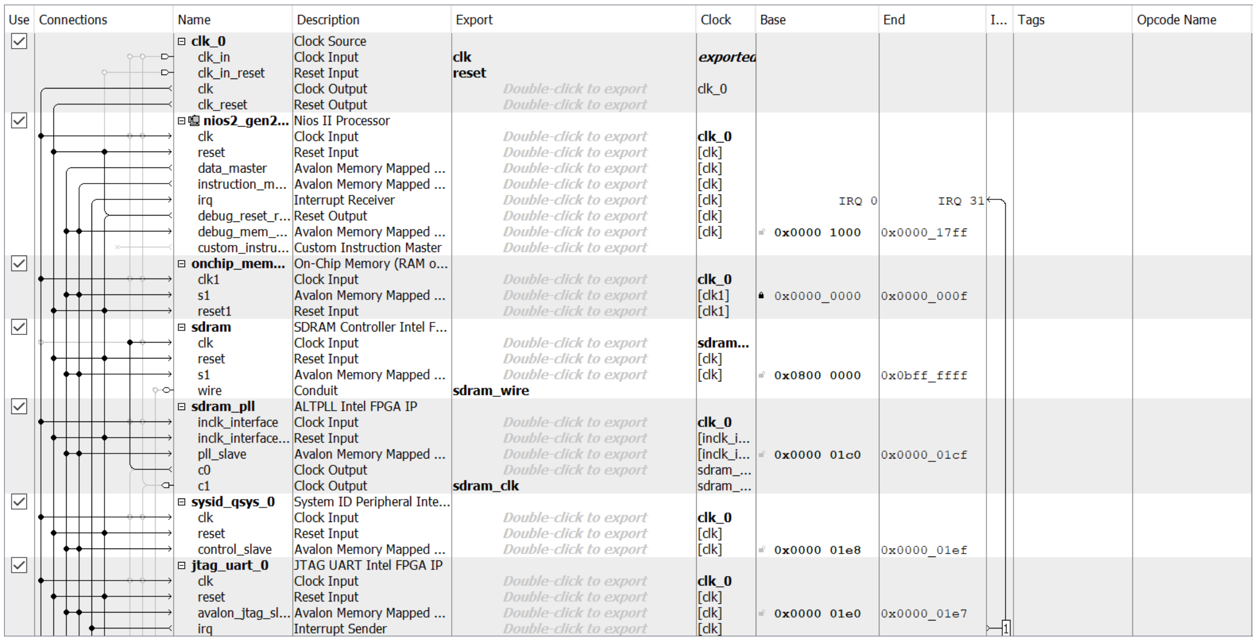
Block Diagram

This diagram should represent the placement of all your modules in the top level. Please only include the top-level diagram and not the RTL view of every module

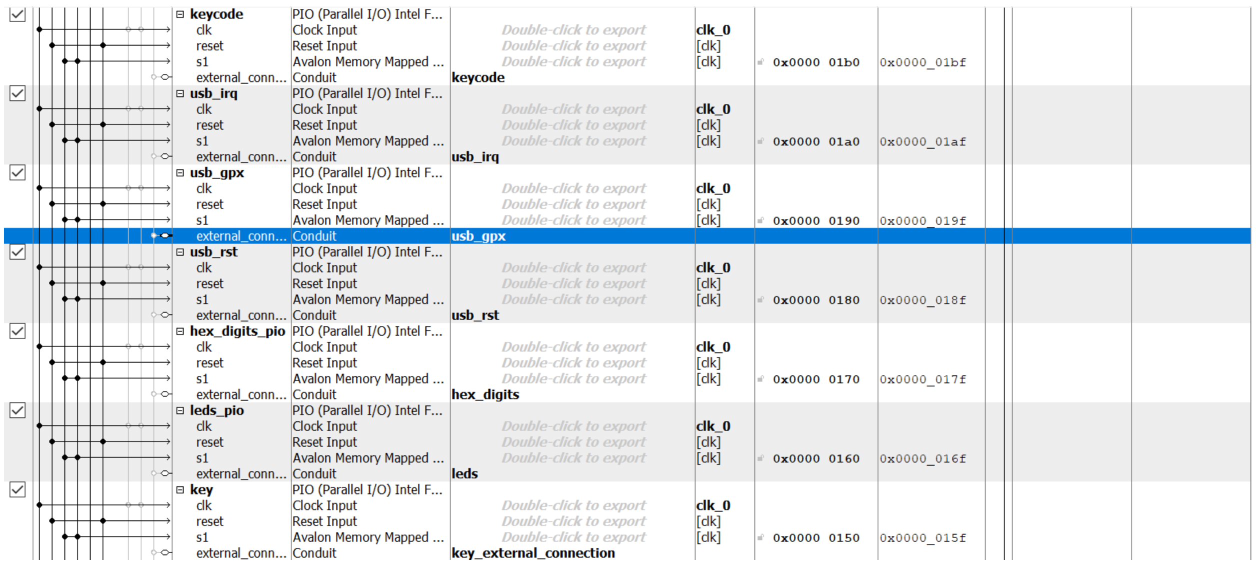


Module Descriptions

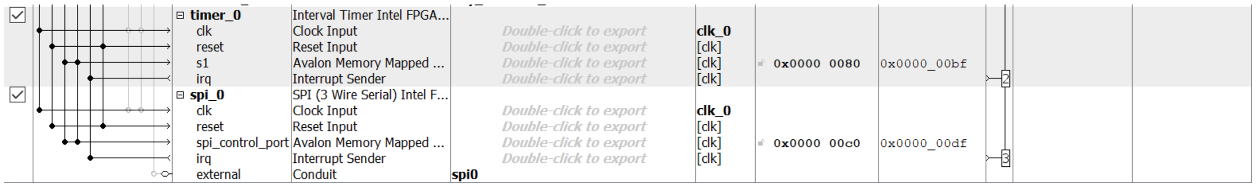
A guide on how to do this was shown in the Lab 5 report outline. Do not forget to describe the Platform Designer generated file for your Nios II system! When describing the generated file, you should describe the PIO blocks added beyond those just needed to make the NIOS system run (i.e. the ones needed to communicate with the USB chip and other components). The Platform Designer view of the Nios II system is helpful here.



Use	Connections	Name	Description	Export	Clock	Base	End	I...	Tags	Opcode Name
✓		⊟ **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...**	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 ...	*Double-click to export*	[clk]					
		instruction_m...	Avalon Memory Mapped ...	*Double-click to export*	[clk]					
		irq	Interrupt Receiver	*Double-click to export*	[clk]	IRQ 0	IRQ 31			
		debug_reset_r...	Reset Output	*Double-click to export*	[clk]					
		debug_mem_...	Avalon Memory Mapped ...	*Double-click to export*	[clk]	0x0000 1000	0x0000_17ff			
		custom_instru...	Custom Instruction Master	*Double-click to export*						
✓		⊟ **onchip_mem...**	On-Chip Memory (RAM o...							
		clk1	Clock Input	*Double-click to export*	**clk_0**					
		s1	Avalon Memory Mapped ...	*Double-click to export*	[clk1]	0x0000_0000	0x0000_000f			
		reset1	Reset Input	*Double-click to export*	[clk1]					
✓		⊟ **sdram**	SDRAM Controller Intel F...							
		clk	Clock Input	*Double-click to export*	**sdram...**					
		reset	Reset Input	*Double-click to export*	[clk]					
		s1	Avalon Memory Mapped ...	*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 Input	*Double-click to export*	[inclk_i...					
		pll_slave	Avalon Memory Mapped ...	*Double-click to export*	[inclk_i...	0x0000 01c0	0x0000_01cf			
		c0	Clock Output	*Double-click to export*	sdram_...					
		c1	Clock Output	**sdram_clk**	sdram_...					
✓		⊟ **sysid_qsys_0**	System ID Peripheral Inte...							
		clk	Clock Input	*Double-click to export*	**clk_0**					
		reset	Reset Input	*Double-click to export*	[clk]					
		control_slave	Avalon Memory Mapped ...	*Double-click to export*	[clk]	0x0000 01e8	0x0000_01ef			
✓		⊟ **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_sl...	Avalon Memory Mapped ...	*Double-click to export*	[clk]	0x0000 01e0	0x0000_01e7			
		irq	Interrupt Sender	*Double-click to export*	[clk]			1		



Use	Connections	Name	Description	Export	Clock	Base	End	I...	Tags	Opcode Name
✓		⊟ **keycode**	PIO (Parallel I/O) Intel F...							
		clk	Clock Input	*Double-click to export*	**clk_0**					
		reset	Reset Input	*Double-click to export*	[clk]					
		s1	Avalon Memory Mapped ...	*Double-click to export*	[clk]	0x0000 01b0	0x0000_01bf			
		external_conn...	Conduit	**keycode**						
✓		⊟ **usb_irq**	PIO (Parallel I/O) Intel F...							
		clk	Clock Input	*Double-click to export*	**clk_0**					
		reset	Reset Input	*Double-click to export*	[clk]					
		s1	Avalon Memory Mapped ...	*Double-click to export*	[clk]	0x0000 01a0	0x0000_01af			
		external_conn...	Conduit	**usb_irq**						
✓		⊟ **usb_gpx**	PIO (Parallel I/O) Intel F...							
		clk	Clock Input	*Double-click to export*	**clk_0**					
		reset	Reset Input	*Double-click to export*	[clk]					
		s1	Avalon Memory Mapped ...	*Double-click to export*	[clk]	0x0000 0190	0x0000_019f			
		external_conn...	Conduit	**usb_gpx**						
✓		⊟ **usb_rst**	PIO (Parallel I/O) Intel F...							
		clk	Clock Input	*Double-click to export*	**clk_0**					
		reset	Reset Input	*Double-click to export*	[clk]					
		s1	Avalon Memory Mapped ...	*Double-click to export*	[clk]	0x0000 0180	0x0000_018f			
		external_conn...	Conduit	**usb_rst**						
✓		⊟ **hex_digits_pio**	PIO (Parallel I/O) Intel F...							
		clk	Clock Input	*Double-click to export*	**clk_0**					
		reset	Reset Input	*Double-click to export*	[clk]					
		s1	Avalon Memory Mapped ...	*Double-click to export*	[clk]	0x0000 0170	0x0000_017f			
		external_conn...	Conduit	**hex_digits**						
✓		⊟ **leds_pio**	PIO (Parallel I/O) Intel F...							
		clk	Clock Input	*Double-click to export*	**clk_0**					
		reset	Reset Input	*Double-click to export*	[clk]					
		s1	Avalon Memory Mapped ...	*Double-click to export*	[clk]	0x0000 0160	0x0000_016f			
		external_conn...	Conduit	**leds**						
✓		⊟ **key**	PIO (Parallel I/O) Intel F...							
		clk	Clock Input	*Double-click to export*	**clk_0**					
		reset	Reset Input	*Double-click to export*	[clk]					
		s1	Avalon Memory Mapped ...	*Double-click to export*	[clk]	0x0000 0150	0x0000_015f			
		external_conn...	Conduit	**key_external_connection**						



✓	timer_0	Interval Timer Intel FPGA...					
	clk	Clock Input	Double-click to export	clk_0			
	reset	Reset Input	Double-click to export	[clk]			
	s1	Avalon Memory Mapped ...	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 F...					
	clk	Clock Input	Double-click to export	clk_0			
	reset	Reset Input	Double-click to export	[clk]			
	spi_control_port	Avalon Memory Mapped ...	Double-click to export	[clk]	0x0000 00c0	0x0000_00df	
	irq	Interrupt Sender	Double-click to export	[clk]			3
	external	Conduit	spi0				

Module: lab8_soc.v
Inputs: clk_clk, key_external_connection_export, reset_reset_n, spi0_MISO, usb_gpx_export, usb_irq_export
Outputs: sdram_clk_clk, sdram_wire_cas_n, sdram_wire_cke, sdram_wire_cs_n, sdram_wire_ras_n, sdram_wire_we_n,
[1:0] sdram_wire_ba, sdram_wire_dqm,
[7:0] keycode_export,
[12:0] sdram_wire_addr,
[13:0] leds_export,
[15:0] hex_digits_report,
Inouts: [15:0] sdram_wire_dq

Description: This module generates the HDL, therefore describes the hardware which includes the components and behaves as a sub-top level module, initialized inside the actual top level module. The inputs and outputs are exported to be connected to other modules.
Purpose: The purpose of this module is to connect the clock and reset inputs, other usb inputs and outputs, the CPU of the system, PLLs and the timers, Parallel IO units and the SPI to each other and the top level module. This makes up a Nios II/e processor that can be run through Eclipse.

Module: VGA_controller.sv
Inputs: Clk, Reset
Outputs: hs, vs, pixel_clk, blank, sync, [9:0] DrawX, [9:0] DrawY
Inouts: None

Description: This module provides us with all the VGA parameters necessary for displaying an output to the monitor. It takes in an input of Clk and Reset and provides us with a horizontal sync pulse, a vertical sync pulse, a 25 MHz pixel clock output along with counters to keep track of horizontal and vertical coordinates.
Purpose: The purpose of this module is to allow us to display an output to the VGA display. Vertical and horizontal syncs are used to keep track of the display to know where the pixel is being placed and when all the pixels in the frame are drawn and a new frame needs to be drawn.

Module: ball.sv
Inputs: Reset, frame_clk, [7:0] keycode
Outputs: [9:0] BallX, [9:0] BallY, [9:0] Balls
Inouts: None

Description: This module calculates the position of Pacman for the next frame given the keyboard input.
Purpose: The purpose of this module is to update the position of Pacman on the screen given the input from the keyboard so the new frame can be drawn to the monitor (controlled by WASD keys). It ensures that Pacman only moves in one direction at a time, either just vertically or just horizontally. It also stops Pacman all when it hits the hard coded walls on the map.

Module: two.sv
Inputs: Reset, frame_clk, [7:0] keycode
Outputs: [9:0] Ball2X, [9:0] Ball2Y, [9:0] Ball2S
Inouts: None

Description: This module calculates the position of the second (evil) Pacman for the next frame given the keyboard input.
Purpose: The purpose of this module is to update the position of the second Pacman on the screen given the input from the keyboard so the new frame can be drawn to the monitor (controlled by the arrow keys). It ensures that the ball only moves in one direction at a time, either just vertically or just horizontally. It also stops the second Pacman when it hits the hard coded walls on the map.

Module: Color_Mapper.sv
Inputs: [9:0] BallX, [9:0] BallY, [9:0] DrawX, [9:0] DrawY, [9:0] Ball_size
Outputs: [7:0] Red, [7:0] Green, [7:0] Blue
Inouts: None

Description: This module provides the VGA output with the RGB values for it to display on the monitor.
Purpose: The purpose of this module is to track the coordinates of the two Pacmans as well as their radii to know which pixels need to be colored to represent them. It also has hard coded

walls and beans which are also colored. The rest of the pixels on the monitor will be the background color. It instantiates food_register.sv to keep track of the beans eaten and the score.

Module: HexDriver.sv
Inputs: [3:0] In0
Outputs: [6:0] Out0
Inouts: None

Description: This module converts hex values into a 7 bit output that can be displayed on the Hex display of our FPGA board.
Purpose: The purpose of this module is to allow us to display hex values on our FPGA board.

Module: food_register.sv
Inputs: Reset, frame_clk, [5:0] Load,
Outputs: [44:0] Data, [5:0] Score
Inouts: None

Description: This module keeps track of the beans and the score.
Purpose: The purpose of this module is to take the Load signal from Color_Mapper.sv when the main Pacman collides with a bean and switches the bit on Data corresponding to that bean off. It also increments the score.

Module: collision.sv
Inputs: Reset, Clk, [9:0] BallX, BallY, BallS, Ball2X, Ball2Y, Ball2S,
Outputs: caught
Inouts: None

Description: This module checks collision of the two Pacmans.
Purpose: The purpose of this module is to reset the game (second Pacman wins) when the two Pacmans collide. It takes in coordinates of both Pacmans to check if they are on top of each other.

Post-lab

1. *Refer to the Design Resources and Statistics in IQT.30-32 and complete the following design statistics table.*

LUT	4759
DSP	0
Memory (BRAM)	11392
Flip-Flop	2587
Frequency	119.73 MHz
Static Power	96.18 mW
Dynamic Power	0.69 mW
Total Power	106.18 mW

Bugs and Measures Taken

Bugs	Measures
Walls were partially penetrable, and would only stop entities if their center hit the wall.	Wall collision code was modified to take the radius of the entities into account
After a bean was eaten, the last bean that was eaten would respawn.	Modified the register two have a second variable that can save the values.
Ghosts would rotate around walls, get stuck in walls, get out of the screen, etc.	Ghost movement was modified several times, they were eventually replaced by the evil Pacman.
Score could not be properly shown as a Pacman color change. Pacman color was never stable.	Score register and Load signals were modified and reverted. Score display feature was removed.
When the evil Pacman won, the game would	We decided to use the programmer to reload

not load back up.	the game.

Resolved Replaced Unresovled

Conclusion

Overall, the two-player Pacman game was functional and playable. Although we didn't add ghosts to our Pacman, we made an interesting two player version. This added an interesting take as one player is the original good Pacman and one gets to be the evil red pacman. This evil pacman has the ability to go through walls, making the game more challenging for good Pacman. One thing that we found particularly interesting was storing memory through the game. We had some variables in modules that would be set every clock cycle, but for knowing whether or not food pellets had been eaten we needed to hold memory. After many failed methods we decided on using a register that would also keep track of score. This method proved successful. Things we could improve on with this game would be the addition of ghosts and added sound effects. All in all, we learned a lot with this project and are happy with the game.

Citations

Rishi's code for creating sprites (https://github.com/Atrifex/ECE385-HelperTools)