ECE 385
Fall 2022
Final Project

Final project
Pacman on FPGA

Haocheng Yang((hy38)
Yicheng Zhou(yz69)

Introduction:

In this project, we designed and implemented a hardware-based Pacman. Similar to the original release, our designed has a map, and obstacles in the map (walls), and a “portal” at left and right sides of the map. There are “pacs” for the player to collect. A score board is present for score keeping, with each “pac” worth 1 point and a total of 80 points is avaliable. The game can be played with “W” “A” “S” “D” keys on a keyboard.

Written Description of the System:

1.) Summary of Operation
   a. Initializing program on FPGA
   b. Launch Eclipse to initialize NIOS-II for USB control (I/O)
   c. Press any control key to start the game “the gate will open and close after player existed the spawn location.”
   d. During game, if the player’s life run out or a “reset” key is pressed, the game will reset to state just after (b.)

2.) Description of hardware component
   a. The hardware used includes the NIOS-II processor, sdram controller, on chip memory and a clock phase shifter. The sdram controller is used to control and specify elements of the sdram. The on-chip memory is used to store values. The clock phase shifter is used to phase shift the clock signal so that the sdram is read at the correct window.

b. The I/O is implemented similarly to Lab 62, additionally, the input enable/disable is co-implemented with the “wall” (ie. When the player character hits a wall, the input for that direction is disable for the period where character remain in contact with the wall.)
c. A score counter is implemented with hardware due to eclipse keeps crashing. The counter utilizes a local hex-decimal translator to do the decimal wise conversion and output. The score keeper display is implemented via “virtual led” on screen.
d. A color mapper is used to determine what to display and where to display. It receives a large number of signals and transforms

3.)Description of software component

a. The software component of this system is identical to the implementation of the software in lab6.2. The software features a usb_kb file which feeds the keyboard inputs in terms of “keycodes” -- hexadecimal codes representing each key pressed by a stack of individual hex codes--into the external conduits of the hardware components through SPI and the Avalon services provided by intel.
b. Regarding the details already discussed in lab6.2, we will skip here. There is not point copy and pasting such details.

Diagrams

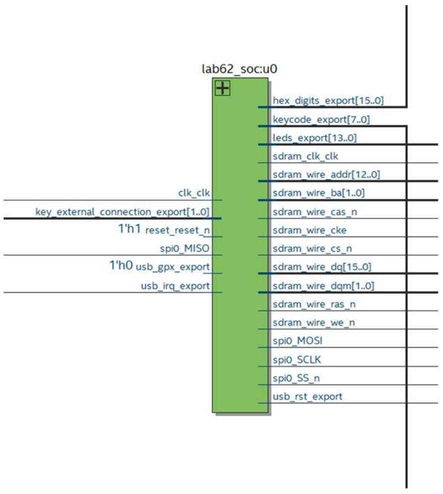
1.)Block diagram:

a. RTL view for the system:

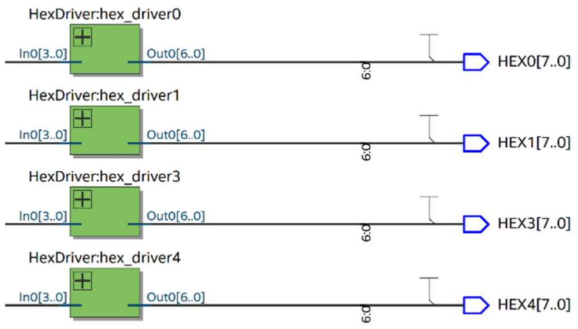
i. (There's more than one page to the RTL View, and all have miniature characters that are non-readable. Thus, I would separate the view into multiple parts):

ii. Overview on the right (Not helpful, just to assess the situation):

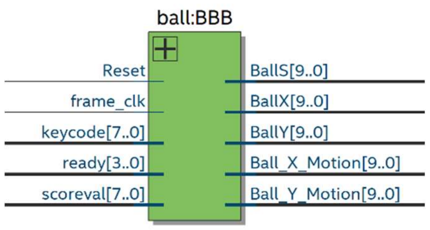
iii. The SoC:



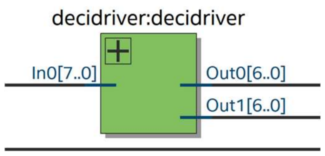
iv. The Hex Drivers:



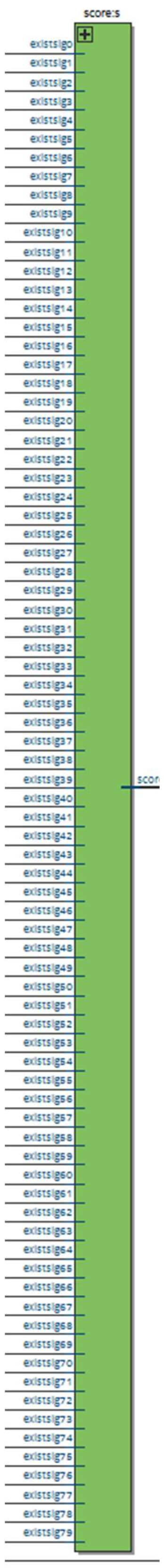
v. The Ball:

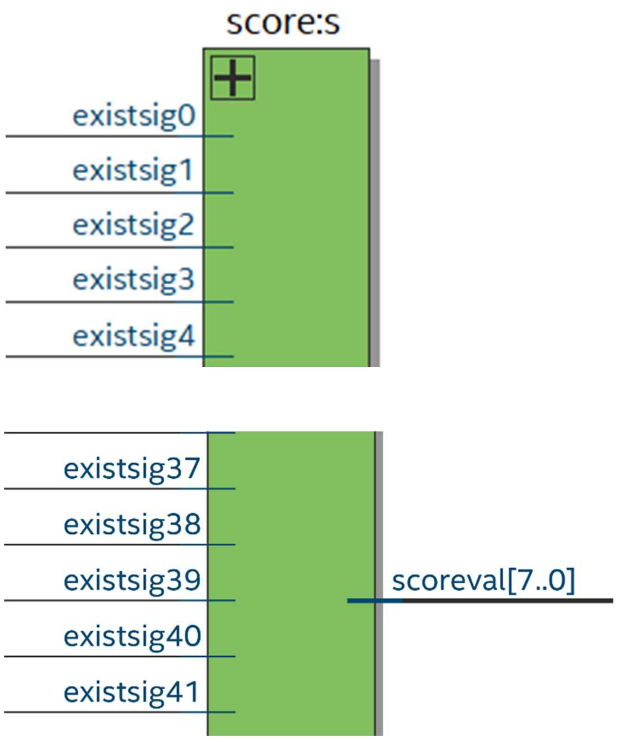


vi. The "Deci" driver:

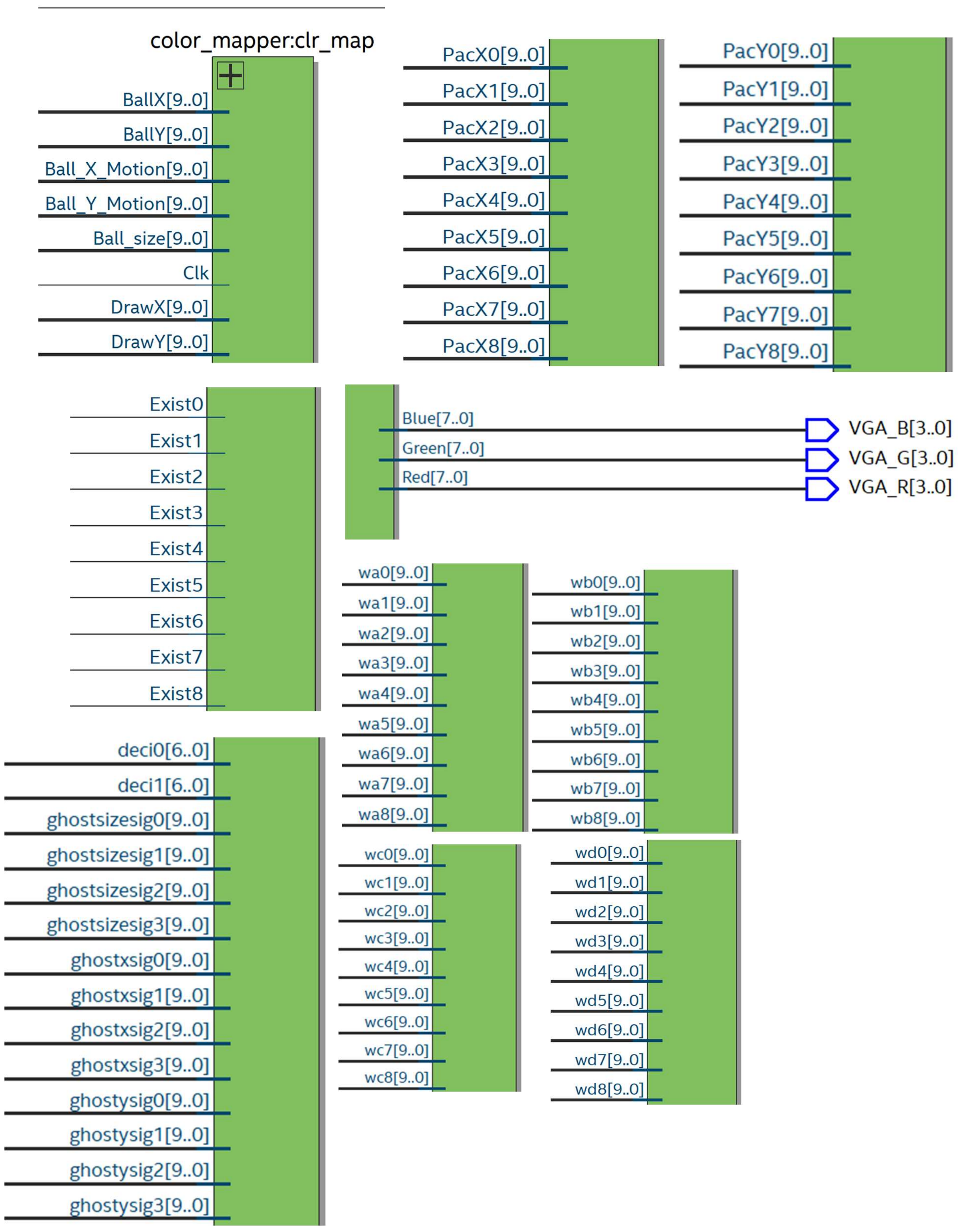


vii. The Score Module (Extras for ease viewing):

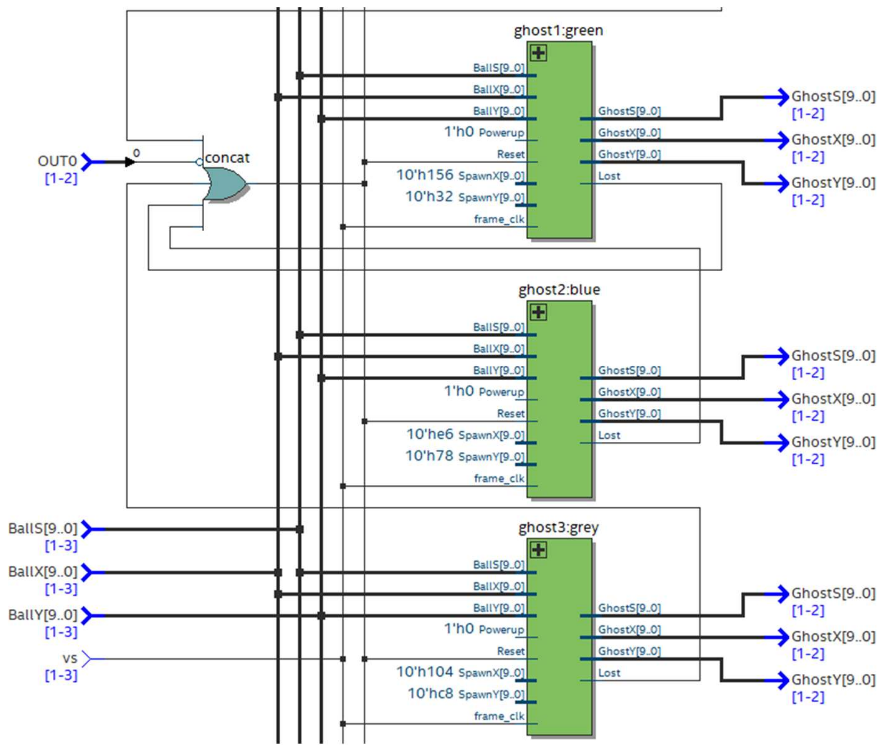




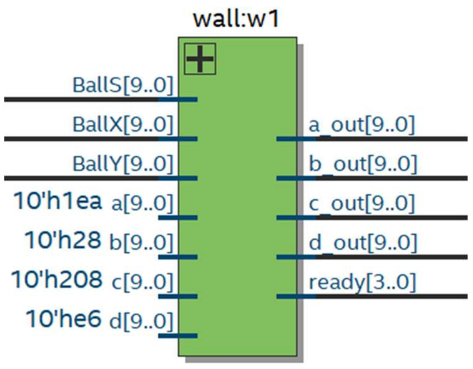
viii. Color Mapper(separated for better view):



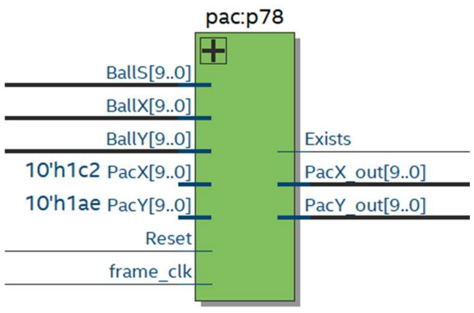
ix. Ghosts:



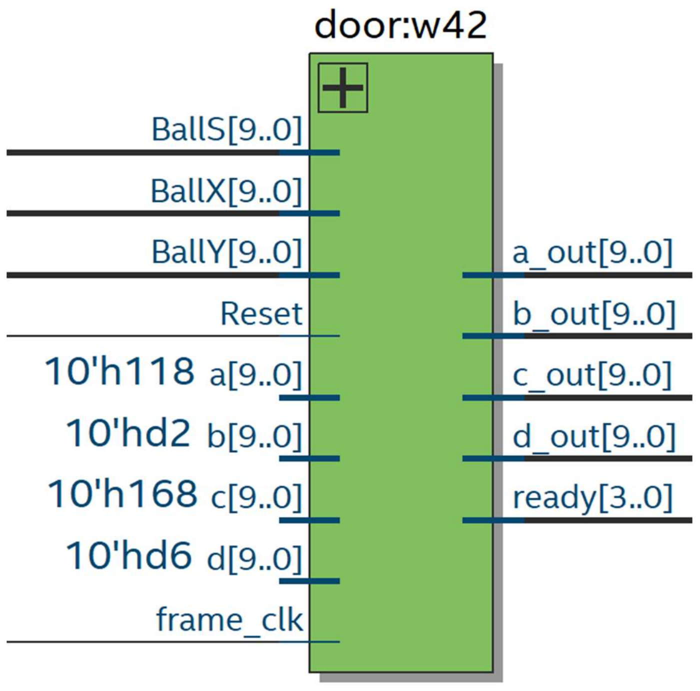
x. Wall, example:



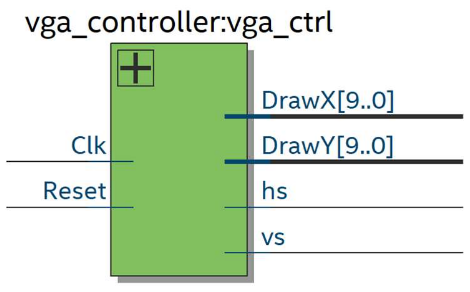
xi. Pac, example:



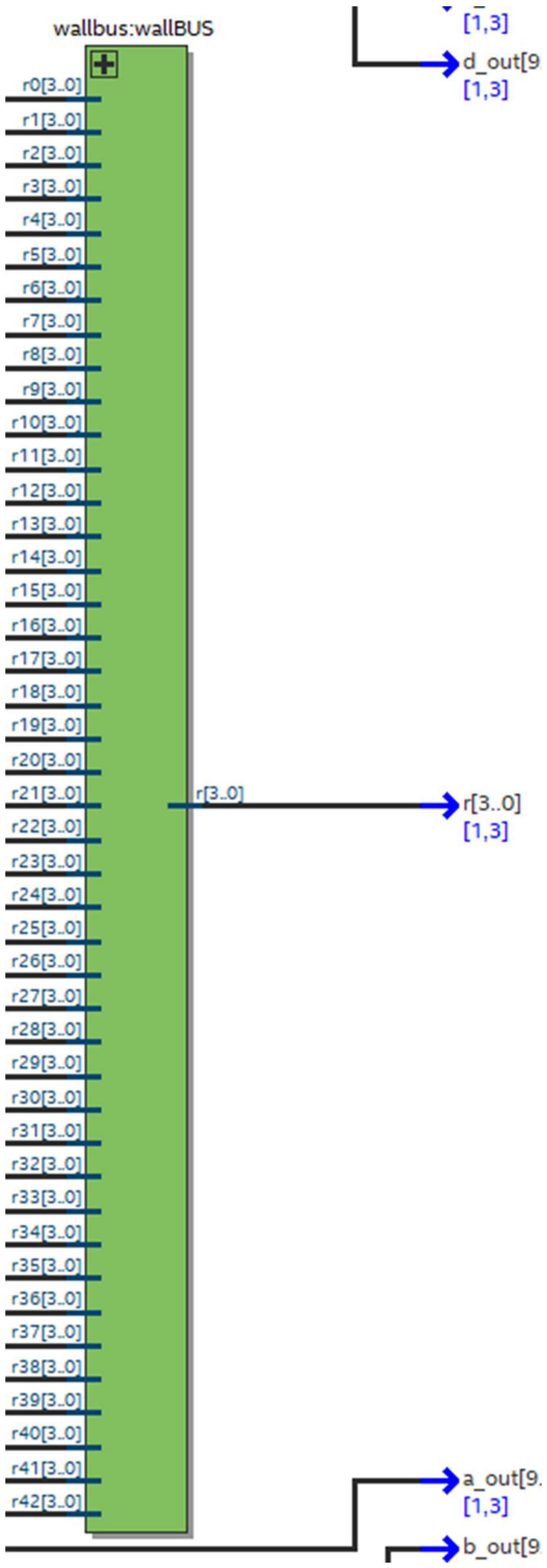
xii. Door:



xiii. VGA Controller:

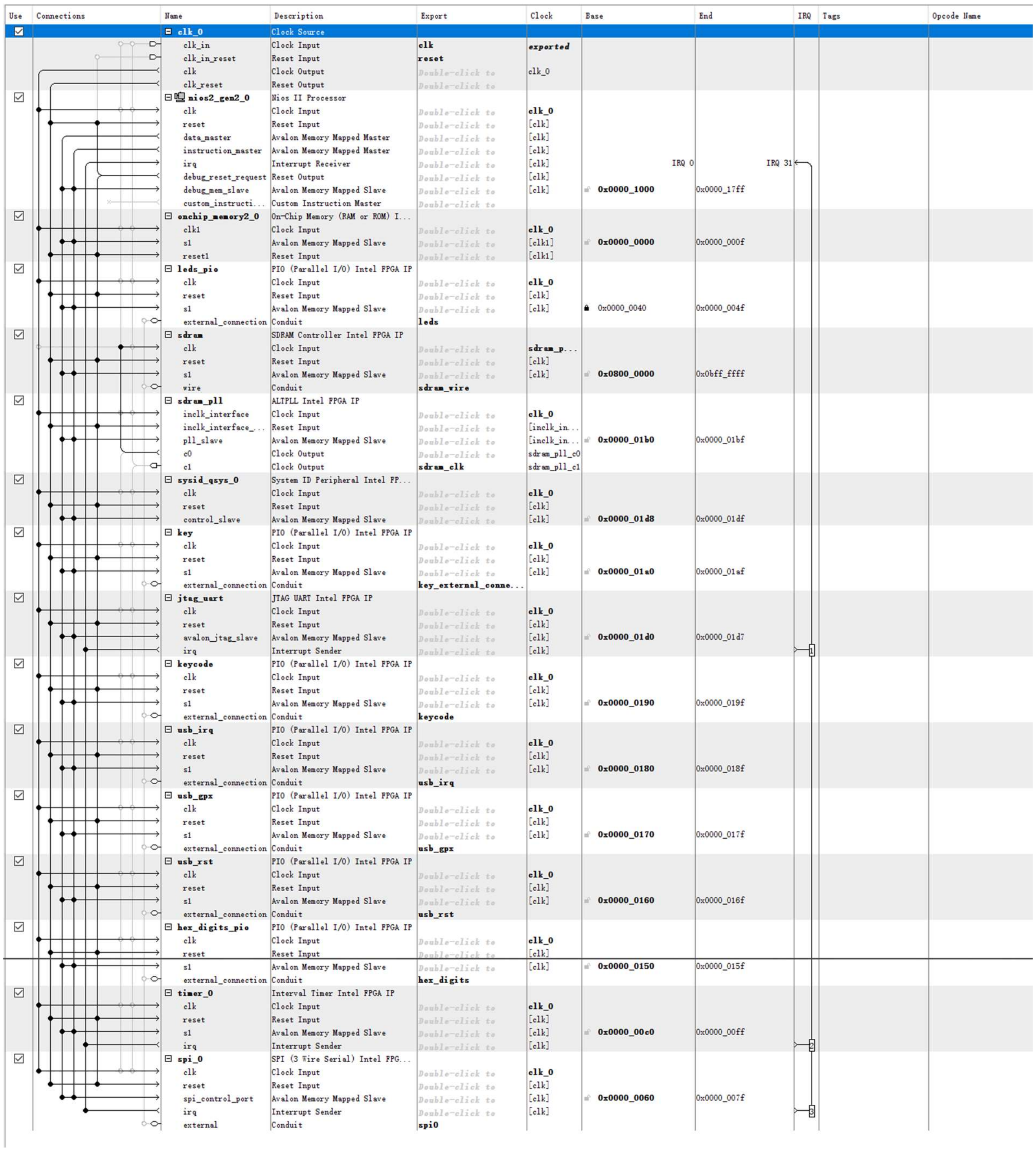


xiv. Wall Bus:



wallbus:wallBUS
r0[3..0]
r1[3..0]
r2[3..0]
r3[3..0]
r4[3..0]
r5[3..0]
r6[3..0]
r7[3..0]
r8[3..0]
r9[3..0]
r10[3..0]
r11[3..0]
r12[3..0]
r13[3..0]
r14[3..0]
r15[3..0]
r16[3..0]
r17[3..0]
r18[3..0]
r19[3..0]
r20[3..0]
r21[3..0]
r22[3..0]
r23[3..0]
r24[3..0]
r25[3..0]
r26[3..0]
r27[3..0]
r28[3..0]
r29[3..0]
r30[3..0]
r31[3..0]
r32[3..0]
r33[3..0]
r34[3..0]
r35[3..0]
r36[3..0]
r37[3..0]
r38[3..0]
r39[3..0]
r40[3..0]
r41[3..0]
r42[3..0]
r[3..0]
[1,3]
d_out[9
[1,3]
r[3..0]
[1,3]
a_out[9.
[1,3]
b_out[9

2.)Diagram of platform designer:



Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	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*	clk_0					
		clk_reset	Reset Output	*Double-click to*						
✓		⊟ **nios2_gen2_0**	Nios II Processor							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		data_master	Avalon Memory Mapped Master	*Double-click to*	[clk]					
		instruction_master	Avalon Memory Mapped Master	*Double-click to*	[clk]					
		irq	Interrupt Receiver	*Double-click to*	[clk]	IRQ 0	IRQ 31			
		debug_reset_request	Reset Output	*Double-click to*	[clk]					
		debug_mem_slave	Avalon Memory Mapped Slave	*Double-click to*	[clk]	**0x0000_1000**	0x0000_17ff			
		custom_instructi...	Custom Instruction Master	*Double-click to*						
✓		⊟ **onchip_memory2_0**	On-Chip Memory (RAM or ROM) I...							
		clk1	Clock Input	*Double-click to*	**clk_0**					
		s1	Avalon Memory Mapped Slave	*Double-click to*	[clk1]	**0x0000_0000**	0x0000_000f			
		reset1	Reset Input	*Double-click to*	[clk1]					
✓		⊟ **leds_pio**	PIO (Parallel I/O) Intel FPGA IP							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		s1	Avalon Memory Mapped Slave	*Double-click to*	[clk]	0x0000_0040	0x0000_004f			
		external_connection	Conduit	**leds**						
✓		⊟ **sdram**	SDRAM Controller Intel FPGA IP							
		clk	Clock Input	*Double-click to*	**sdram_p...**					
		reset	Reset Input	*Double-click to*	[clk]					
		s1	Avalon Memory Mapped Slave	*Double-click to*	[clk]	**0x0800_0000**	0x0bff_ffff			
		wire	Conduit	**sdram_wire**						
✓		⊟ **sdram_pll**	ALTPLL Intel FPGA IP							
		inclk_interface	Clock Input	*Double-click to*	**clk_0**					
		inclk_interface_...	Reset Input	*Double-click to*	[inclk_in...					
		pll_slave	Avalon Memory Mapped Slave	*Double-click to*	[inclk_in...	**0x0000_01b0**	0x0000_01bf			
		c0	Clock Output	*Double-click to*	sdram_pll_c0					
		c1	Clock Output	**sdram_clk**	sdram_pll_c1					
✓		⊟ **sysid_qsys_0**	System ID Peripheral Intel FP...							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		control_slave	Avalon Memory Mapped Slave	*Double-click to*	[clk]	**0x0000_01d8**	0x0000_01df			
✓		⊟ **key**	PIO (Parallel I/O) Intel FPGA IP							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		s1	Avalon Memory Mapped Slave	*Double-click to*	[clk]	**0x0000_01a0**	0x0000_01af			
		external_connection	Conduit	**key_external_conne...**						
✓		⊟ **jtag_uart**	JTAG UART Intel FPGA IP							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		avalon_jtag_slave	Avalon Memory Mapped Slave	*Double-click to*	[clk]	**0x0000_01d0**	0x0000_01d7			
		irq	Interrupt Sender	*Double-click to*	[clk]			1		
✓		⊟ **keycode**	PIO (Parallel I/O) Intel FPGA IP							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		s1	Avalon Memory Mapped Slave	*Double-click to*	[clk]	**0x0000_0190**	0x0000_019f			
		external_connection	Conduit	**keycode**						
✓		⊟ **usb_irq**	PIO (Parallel I/O) Intel FPGA IP							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		s1	Avalon Memory Mapped Slave	*Double-click to*	[clk]	**0x0000_0180**	0x0000_018f			
		external_connection	Conduit	**usb_irq**						
✓		⊟ **usb_gpx**	PIO (Parallel I/O) Intel FPGA IP							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		s1	Avalon Memory Mapped Slave	*Double-click to*	[clk]	**0x0000_0170**	0x0000_017f			
		external_connection	Conduit	**usb_gpx**						
✓		⊟ **usb_rst**	PIO (Parallel I/O) Intel FPGA IP							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		s1	Avalon Memory Mapped Slave	*Double-click to*	[clk]	**0x0000_0160**	0x0000_016f			
		external_connection	Conduit	**usb_rst**						
✓		⊟ **hex_digits_pio**	PIO (Parallel I/O) Intel FPGA IP							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		s1	Avalon Memory Mapped Slave	*Double-click to*	[clk]	**0x0000_0150**	0x0000_015f			
		external_connection	Conduit	**hex_digits**						
✓		⊟ **timer_0**	Interval Timer Intel FPGA IP							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		s1	Avalon Memory Mapped Slave	*Double-click to*	[clk]	**0x0000_00c0**	0x0000_00ff			
		irq	Interrupt Sender	*Double-click to*	[clk]			2		
✓		⊟ **spi_0**	SPI (3 Wire Serial) Intel FPG...							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		spi_control_port	Avalon Memory Mapped Slave	*Double-click to*	[clk]	**0x0000_0060**	0x0000_007f			
		irq	Interrupt Sender	*Double-click to*	[clk]			3		
		external	Conduit	**spi0**						

Written description of .sv files

1.) VGA_controller.sv:

Inputs: clk, Reset

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

Description: this is the VGA controller which sends signals through VGA port to the monitor.

Purpose: This is the module which determines the boundary of the screen, with it's on board clock, generates the pixel clock. It also generates the horizontal and vertical sync.

2.) Lab62.sv:

Inputs: MAX10_CLK1_50, [1:0] KEY, [9:0]SW, [15:0]DRA,_DQ, [15:0] ARDUINO_IO, Arduino_reset_n.

Outputs: [9:0] LEDR, [7:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, DRAM_CLK, DRAM_CKE, [12:0] DRAM_ADDR, [1:0] DRAM_BA, DRAM_LDQMM DRAM_UDQM, DRAM_CS_N, DRAM_WE_N, DRAM_CAS_N, DRAM_RAS_N, VGA_HS, VGA_VS, [3:0]VGA_R, [3:0]VGA_G, [3:0] VGA_B

Description: This is the top-level entity of lab 6

Purpose: this calls on all other modules for functions and bridge data between other modules. Many local wires are present just to make the process easier.

3.) HexDriver.sv:

Inputs: [3:0] In0

Outputs: [6:0] Out0

Description: This module converts 4bit-binry input and convert it to 7 bits for hex-display.

Purpose: This module converts 4bit binary numbers from registers into hexadecimal so LEDs can display outputs.

4.) Color_Mapper.sv:

Inputs: Clk, [9:0] BallX, BallY, DrawX, DrawY, Ball_size, Ball_X_Motion, Ball_Y_Motion, Exist[0-79], [9:0] PacX[0-79], PacY[0-79], [9:0] wa[0-42], wb[0-42], wc[0-42], wd[0-42], [6:0] deci0, deci1, [9:0] ghostxsig0,ghostysig0,ghostsizesig0, [9:0] ghostxsig1,ghostysig1,ghostsizesig1, [9:0] ghostxsig2,ghostysig2,ghostsizesig2, [9:0] ghostxsig3,ghostysig3,ghostsizesig3,

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

Description: This module generates the RGB values for each pixel for monitor.

Purpose: The module consolidates the display "demands" from all different elements, and with integrated algorithm to determine the priority of elements to eventually send out the RGB values for VGA_controller for output.

5.) Ball.sv:

Inputs: Reset, frame_clk, [3:0] ready, [7:0] keycode, [7:0] scoreval,

Outputs: [9:0] BallX, BallY, BallS, Ball_X_Motion, Ball_Y_Motion

Description: logic elements for the ball (player character) that has integrated logic to determine the relative position of the ball, player and pacs.

Purpose: this module is to implement all logic surrounding the player character that takes in keyboard input for control.

6.) pac.sv:

Inputs: Reset, frame_clk, [9:0] BallX, BallY, BallS, PacX, PacY,

Outputs: Exists, [9:0] PacX_out, PacY_out

Description: the logic elements of the pacs scattered on the map.

Purpose: this is to implement the pacs for the player to collect and in conjunction with socre.sv to implement score keeping.

7.) wall.sv:
Inputs: [9:0] BallX, BallY, BallS, a, b, c, d,

Outputs: [3:0] ready, [9:0] a_out, b_out, c_out, d_out

Description: the logic elements of the wall on the map.

Purpose: this is to implement the walls for map design.

8.) wallbus.sv:
Inputs: [3:0] r [0-42]

Outputs: [3:0] r

Description: a data bus.

Purpose: a data bus for the walls on the map. Is practically a giant or gate for all the input r signals. The output basically sums all the r signals and gains a result of what keys to allow on the input side, restricting the ball (Pacman)'s movements.

9.) Score.sv:

Inputs: exists[0-79]

Outputs: [7:0] scoreval

Description: a adder to keep track of how much pac is left on the map

Purpose: for scorekeeping, this module will return a hex score for decidriver.sv for conversion.

10.) Decidriver.sv:
Inputs: [7:0] In0

Outputs: [6:0] Out0, Out1

Description: similar to the hexdriver, to determine a 2digit decimal output from a 2digit hex input.

Purpose: for hex to decimal conversion for each digit so that the score display is not in hex but in decimal.

11.) ghost.sv, ghost1.sv, ghost2.sv:

Inputs: Reset, frame_clk, [9:0] BallX, BallY, BallS, [9:0] SpawnX, SpawnY, Powerup,

Outputs: Lost, [9:0] GhostX, GhostY, GhostS,

Description: an adaptation from the original ball.sv from lab62, this contains the logic elements of the ghosts, they are fixed path logic and will "kill" the player if touched

Purpose: to implement the ghosts in the original game, however, all ghosts are a unified logic unlike the actual game. This is due to a design defect which will be discussed in the conclusion part.

12.) door.sv

Inputs: Reset, frame_clk, [9:0] BallX, BallY, BallS, a, b, c, d, Outputs: [3:0] ready, [9:0] a_out, b_out, c_out, d_out

Description: an adaptation from the wall.sv, which allows a single pass per reset, and then it will become impassible.

Purpose: to implement the spawn location door, once the door is crossed, the game starts.

A Brief Description on the SoC system:

1. Clk_0: global clock signal, connects to most subsystems on this

platform, with the exception of the sdram module using the c0 output

of sdram_pll with a -1ns phase shift to compensate for the short time

window of the sdram when it generates a valid value for read/write.

This module also serves as an reset signal for most modules. The

module takes input from external connections, where in the top level

we connect to buttons.

2. Nios_gen2_0: Nois II processor. Using the /e variant, this is a

economical version of the processor. This module supports JTAG

Debug and ECC RAM Protection only, as opposed to many other features offered in the /f version. This module is the processor of the

system as a modified Harvard machine, providing 2 buses,

data_master and instruction_master, where they are the bus for data

and instruction. Notice this module also have an Interrupt Receiver

where this drives the Keyboard Inputs.

3. Onchip_memory_2_0: On chip RAM/ROM.

4. Leds_pio: An output pio connection, serves to light the LEDs in the

accumulator design, was not removed for convenience.

5. Sdram: Memory of whole system, a 16bits, 1 CS, 4 Bank memory,

with 13 Row and 10 Column of addresses.

6. Sdram_pll: This module outputs a special, phase shifted clock for

sdram, the reason for us to use this feature in this module is described

in Clk_0.

7. Sysid_qsys_0: This is a System ID peripheral which generates a

System ID per connection.

8. Key: This is a 2-bit width pio input, reading the buttons on the FPGA

board, key0, and key1, positioned on the side of the board.

9. Jtag_uart: A block to support the communication with the FPGA with

the computer regarding software data.

10.Keycode: This is an 8-bit pio Output where the 7 segment display will

display the keyboard input in hexadecimal numbers. Note: this is the

number, not the actual display driver.

11.Usb_irq, Usb_gpx, Usb_rst: Interrupt request line: generates a

interrupt when there is data to transfer in this serial bus, gpx and rst

was not introduced in the lab manual, but they are pio input and

output with 1 bit width, respectively.

12.Hex_digits_pio: 7 segment display output pio port. Works with the

HexDriver.sv.

13.Timer_0: Interval timer with 1ms timeout.

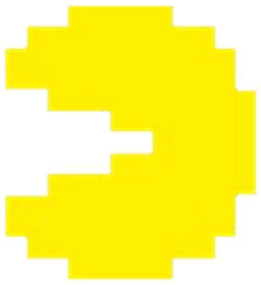
14.Spi_0: a 3 wire serial communication module.

Resource usage:

LUT	5639
DSP	8
BRAM	11392b
FF	2692
Freq	124.47Mhz
Static PWR	96.31mW
Dynamic PWR	0.73mW
Total PWR	106.22mW

Conclusion:

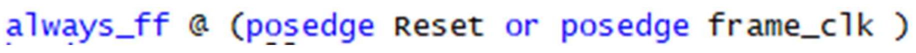
This final lab is supposed to be the lab where we utilize most of the techniques we learned during the semester and use it to build something that shows more or less the information from most labs. However, we were in a rush to complete the project and not to get into any issues, so we used an older platform as in lab6.2. This proved to be very stable, but very hard to do anything beyond basic geometric drawing. This leads us to the intricacies section, where I share some details into the Color _mapper.sv design.



First off, this is a standard Pacman:

Where the implementation is obviously doing a sprite, and then by recognizing Pacman's motion details, changing the sprite display, what we did was to take the output of ball.sv, and draw a triangle towards the Pacman's direction of travel. Granted this is a very useful method, the Pacman we drew looks unlike the original because we are basically enable to emulate the edges.

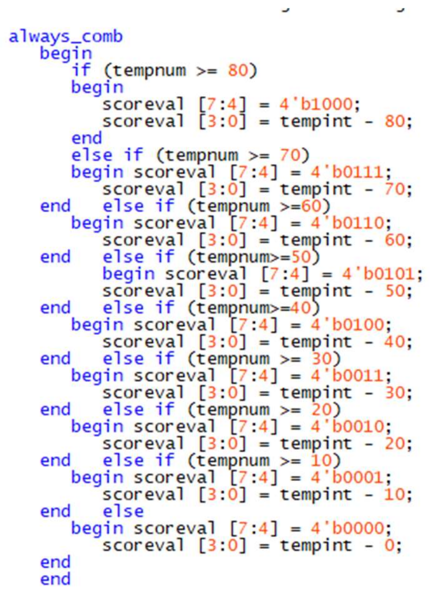
Second, the original game features 3 lives, however, the basic design of lab62.sv means we need to either reset the board when the ghosts touch Pacman, or we would need to reset all the game elements. The game elements are extremely hard to reset due to all the always_ff's only reacting to the global reset and changing it can be troublesome and also frustrating. Example:



```
always_ff @ (posedge Reset or posedge frame_clk )
```

Third, when we designed the wall.sv and wallbus.sv modules, we did not think of how we are going to implement the ghosts. This proved to be a big problem when we're trying to make the ghosts move on their own. The original Pacman, and many arcade games with a built-in monster, would be using a grid system where all of the target locations are stored as grids, and each grid would be a (x,y) but with a certain relation to other grids. For example, the Pacman is travelling from (10, 10) to (10, 20), now the ghost reasonably has to target (10,20), and to follow a certain path to get there. However, because of the wall.sv definition in our program, we basically cannot figure out what path the ghost needs to take, and by then it is too late to implement a rom system to store all the locations that are impassable.

Finally, some thoughts on the good side of using an older, more intuitive platform. This platform, lab62.sv, is extremely easy to use and to add primitive functionality to. Anything we want to have can be done at the hardware level when designed with hardware intentions in mind (i.e. the score counter is just a lot of modules where they receive input from the Deci driver decoder and feed to the color mapper.) This is extremely easy for us to finish a functionality in a short timeframe, but extremely



```
always_comb
   begin
      if (tempnum >= 80)
      begin
         scoreval [7:4] = 4'b1000;
         scoreval [3:0] = tempint - 80;
      end
      else if (tempnum >= 70)
      begin scoreval [7:4] = 4'b0111;
         scoreval [3:0] = tempint - 70;
   end   else if (tempnum >=60)
      begin scoreval [7:4] = 4'b0110;
         scoreval [3:0] = tempint - 60;
   end   else if (tempnum>=50)
         begin scoreval [7:4] = 4'b0101;
         scoreval [3:0] = tempint - 50;
   end   else if (tempnum>=40)
      begin scoreval [7:4] = 4'b0100;
         scoreval [3:0] = tempint - 40;
   end   else if (tempnum >= 30)
      begin scoreval [7:4] = 4'b0011;
         scoreval [3:0] = tempint - 30;
   end   else if (tempnum >= 20)
      begin scoreval [7:4] = 4'b0010;
         scoreval [3:0] = tempint - 20;
   end   else if (tempnum >= 10)
      begin scoreval [7:4] = 4'b0001;
         scoreval [3:0] = tempint - 10;
   end   else
      begin scoreval [7:4] = 4'b0000;
         scoreval [3:0] = tempint - 0;
   end
   end
```

unfriendly towards system of larger scales. For example, when we uncarefully design conduits such as the wall modules or the pac modules, we end up copy pasting the same lines of code 80 or 100 times before we are tired and realize that a rom system could have done it in one time coding and still be easily edited.

All in all, this is a really fun project that we actually learned a lot in the process with, and we will explore the field of FPGA even more in the future.