ECE 385

Spring 2023

Final Lab

# The Binding of an ECE Major

Gustavo Fonseca, Hunter Baisden

TA: Hongshuo Zhang

## Introduction

For our final project we created a top-down RPG shooter inspired by the game “The Binding of Isaac”, hence the name “The Binding on an ECE Major”. The idea behind the game was to have an ECE major travel around the ECE Building exploring iconic areas of the buildings, such as the Main Lobby, the Grainger Auditorium, the TI Design Room, the Daily Byte, and the mysterious basement. The player must fight common enemies of an ECE major, such as a Business major and Tests. The goal of this game is to poke fun at various aspects of ECE Culture and create a completely unique game that could only be created by an actual ECE major at UIUC.

## Main Gameplay Walkthrough

The game begins with the player, an ECE Major, spawning in the middle of the ECE Building Main Lobby. You first encounter the classic Lobby Tables where every ECE Major has nervously sat before an exam. These tables are currently advertising the Spring Jam Concert that occurred in April. The player’s inventory can be seen at the bottom left of the screen, currently all three items are blacked out as they are not collected yet. The player’s health is also displayed in the bottom right of the screen. The player has 5 health to begin with, hence there are 5 hearts being displayed.

The Player is then presented with a choice, what room to enter. There are signs directing the player to each available room: the Auditorium, Design Room, Daily Byte, and the Basement. The player then must go to the desired room and press the F to enter each room. If the player tries to enter the basement now, they will be denied access, the player must first collect the two keys that are being guarded in the Auditorium and the TI Design Room.

The player can also go into the Daily Byte, where they will see a corridor with a star at the end. The player can then pick up this star and put it in their inventory, you can now also see the star appear on the right of your inventory as well. This star unlocks the very powerful ability of invincibility. Now, whenever the player presses the E key, the player will be invulnerable for 1 second, there then will be a 5 second cooldown before the player can use the ability again.

The player can also go into either one of the rooms containing the necessary keys for the basement. If the player enter the auditorium, they will be greeted with a Business Major and Fraternity Brother that is equipped with a jetpack, allowing him to fly over the many chairs blocking the player’s path. The player then must proceed to shoot at this enemy, using the massive pencil in their hand to shoot lead whenever the player presses SPACE. The player can then control the direction of the shot independently of the way that they are moving using the arrow key. The player can also curve bullets around the walls and chairs by changing the direction the player is aiming and pressing SPACE while the bullet is midair. The moment the player enters the room, the Frat Bro will begin to chase the player, using the most efficient path to do so, while also shooting F’s at the player, something they will never have to worry about in their major. The player can take damage in 2 different ways, with the bullets being constantly by the enemy or by the enemy touching them. If the enemy touches the player there is a half a second interval where the player cannot be damaged, preventing the player from instantly dying if they are touched. The enemy only has 3 health; thus the player only has to shoot the enemy 3 times for the enemy to stop moving and stop shooting, the player then has the choice to spare the enemy or shoot them again and make them disappear. The key will appear in the middle of the room after the player has shot the enemy twice, allowing players looking to complete the game as fast as possible the option to skip killing the entirely and leave after grabbing the key. Picking up the key acts in a very similar way to the Star in the Daily Byte. The player is free to leave the room at any time, even without the key; however, if they try to go back into the auditorium the enemy will have recovered, even if they were killed in the first place. In contrast, once you picked up the key within the Auditorium, it will not come back after you picked it.

The player now has to get the second key from the TI Design Room. In this room the player will have to face two enemies now, both being the thing that all ECE majors hate, Exams. The first version of the exam, colored with white paper, moves slowly following the player while shooting. The second version of the exam, colored with blue paper, mover almost as fast as the player, quickly chasing the player down, to keep the game fair we decided to stop this test from shooting at the player. In order to get the key to spawn in this room, the player only has to shoot the white test twice. These enemies again act in the same way as the Frat Bro, only having 3 health before they stop moving and shooting. Grabbing this key has the same behavior as the first key. Once the player grabs the key, they can go back to the Main Lobby. The enemies will again recover if you come back into the room, just like how there are always more exams to come as an ECE Major.

Now that the player has both keys, they can now open the dreaded basement. When they enter, they are greeted with a large slime monster, representing the terror of humidity towards electronics and ECE Majors everywhere. There are no walls within the room, the player must face this enemy head-to-head, nowhere to hide. Unlike the other room, the player cannot leave the room until the slime monster is defeated. Once the player is victorious, the sign directing the player back to the main lobby appears, and the player is free to return to safety. The player is then free to explore any room of their choosing, but be weary that the enemies will always be there to torment to player, as is the life of an ECE Major.

## Hardware Component

## I/O

1. **Inputs:**
    a. **KEY[0]**
       Used as the hardware reset for program
    b. **MAX10_CLK1_50**
       The 50MHz clock that controls the system
    c. **LEDR**
       Controls the LED display to display keyboard keycodes and general debugging

For more detailed inputs and outputs, refer to the System Level Block Diagram section, at the bottom of which has the overall Platform Designer.

## NIOS Interactions

1. **MAX3421E USB**
   This connection connects the keyboard to NIOS II. There are 4 pins in a USB: VDD, D-, D+, GND. The data is transmitted through the D- and D+ pins. In order for the USB device to be enumerated, there are four steps to be performed. Firstly, the MAX3421E chip must be initialized, which is done by the MAX3421E.c program. Secondly, the USB must be reset. Thirdly, the USB address must be assigned. Finally, the device descriptor is gotten. The last three steps are performed by the transfer.c file.
2. **VGA**
   This connection outputs the data from the NIOS II to the monitor. The VGA output has 5 key outputs: Red, Green, Blue, Horizontal Sync (HS), and Vertical Sync (VS). The values from Red, Green, and Blue (RGB) define what color will be output to the selected pixels. The Horizontal Sync is a pulse that tells the screen which pixels on the screen should be colored with the selected RGB value. The Vertical Sync is a pulse that tells the screen which Lines on the screen should be colored with the selected RGB value. For every value of HS and VS that match, the values defined by RGB are drawn on the pixels defined by those values on the screen.

## SPI Protocol

The SPI Protocol is made up of 4 signals: CLK, MOSI, MISO, and SS. The CLK is sent from the Master device, in this case NIOS II, to the be used as the clock for the slave device, MAX3421E. MOSI, Master-Out Slave-In, is used for sending data from NIOS II to the MAX3421E, using the CLK signal as a sync. MISO, Master-In Slave-Out, is used for sending from MAX3421E to NIOS II, again using the CLK signal as a sync.

## C Code Functionality

1. **MAXreg_wr**
   The purpose of this function is to write a single byte into a register within the MAX3421E through the SPI Protocol. We select the desired register and value to write into it based on the parameters. We then use the alt_avalon_spi_command function to connect and write to the selected register.
2. **MAXbytes_wr**
   The purpose of this function is to write multiple bytes into a register within the MAX3421E through the SPI Protocol. We select the desired register, the number of bytes, and the actual data to write into the register based on the parameters. We again use the alt_avalon_spi_command function to connect and write into the selected register. The function also returns the pointer of the last byte stored into the register.
3. **MAXreg_rd**
   The purpose of this function is to read a single byte from a register within the

MAX3421E through the SPI Protocol. We then select the desired register based on the parameters. We then use the alt_avalon_spi_command function to connect and read from the selected register and save its value. The function will then output this saved value.

4. **MAXbytes_rd**
The purpose of this function is to read multiple bytes from a register within the MAX3421E through the SPI Protocol. We select the desired register, the number of bytes, and the location where the read data will be stored based on the parameters. We then use the alt_avalon_spi_command function to connect and read from the selected register and save its value. The function will then return the pointer of the last byte stored into memory.

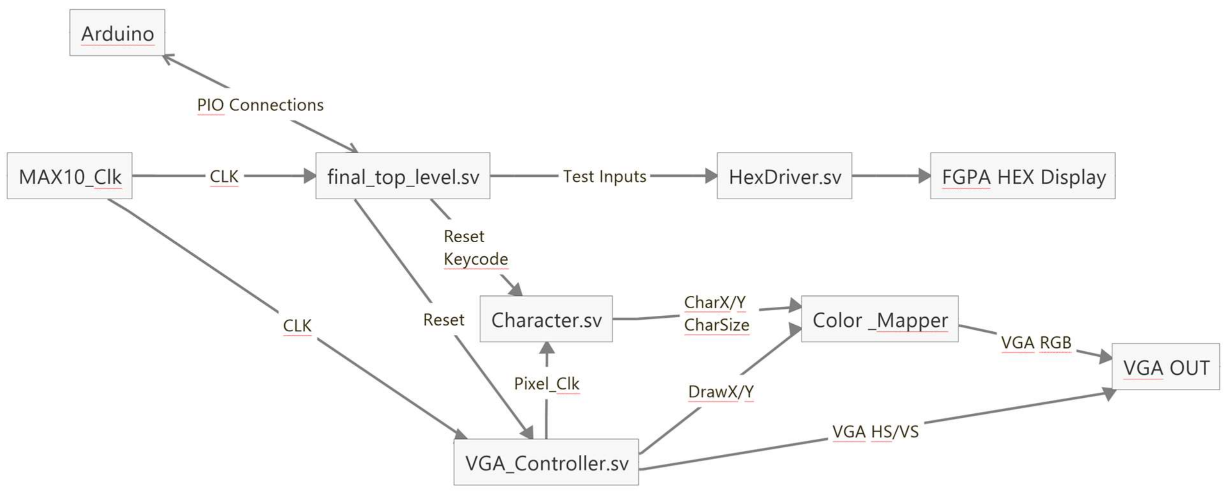
## VGA Operation

The VGA works by using an electron beam to paint each pixel, starting from the left side of the monitor until the bottom right side. There are 640 horizonal pixels and 480 vertical lines. The VGA needs three main signals: HS, VS, and RGB.

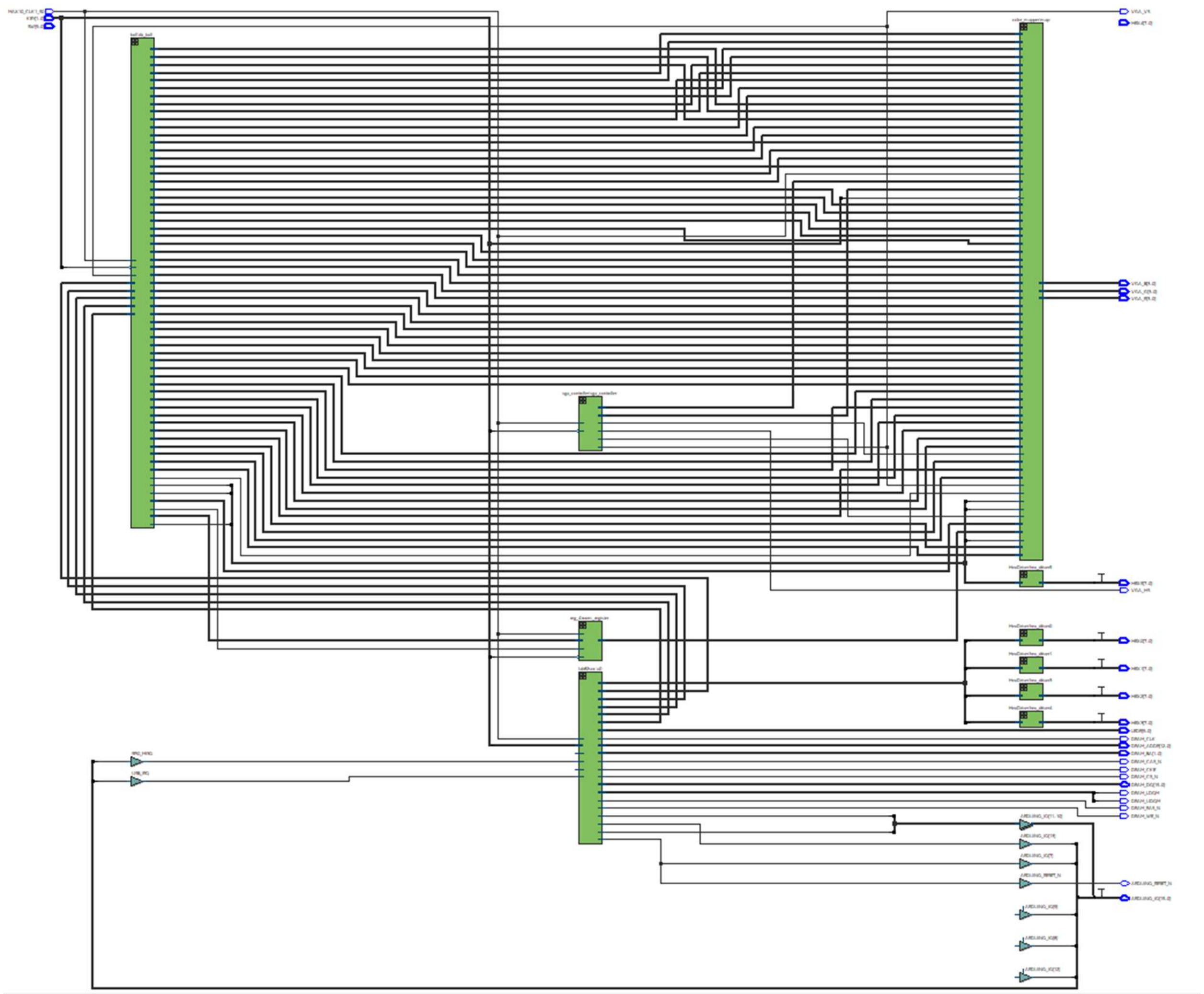
The main interaction between the vga_controller, ball, and color_mapper modules involve checking if the current pixel should be colored as the ball or the background. To start, the vga_controller orients the system to what is the current pixel we are modifying, sending the current HS and VS to the VGA as well. The ball module then takes in the current keycode from the keyboard, it then outputs the current position of the ball. The color_mapper then checks if the current pixel matches with the ball's current position and desired radius. If they match, then the current pixel will be colored with selected ball's color, if not, the pixel will be colored with the selected background color. Finally, the color_mapper outputs the current RGB values to the VGA.

## Top Level Block Diagram

This is the simplified Block Diagram of the lab.



This is the top-level diagram of the full lab.



## .SV Modules

**Module**: final_top_level (in final_top_level.sv)

**Inputs**: MAX10_CLK1_50, [1:0] KEY, [9:0] SW

**Outputs**: [9:0] LEDR, [7:0] HEX (0-5), DRAM_CLK, DRAM_CKE, [12:0] DRAM_ADDR, [1:0] DRAM_BA,  DRAM_LDQM, DRAM_UDQM, DRAM_CS_N, DRAM_WE_N, DRAM_CAS_N, DRAM_RAS_N

**Inouts:** [15:0] DRAM_DQ, ARDUINO_IO, ARDUINO_RESET_N

**Description**: This is the top-level module for this project. It declares the 4 submodules, as well as the SOC. These include vga_controller, character controller, color_mapper, and HexDriver. It takes inputs for the DRAM to be used by the SOC.

**Purpose**: This module serves as the top level for all the interactions with the FPGA. It is used as the top level in compilation when we program the FPGA. A diagram for this function can be seen above.

**Module**: color_mapper (in Color_Mapper.sv)

**Inputs**: [9:0] BallX, [9:0]  BallY, [9:0] DrawY, [9:0] Ball_size, [9:0] BulletX, [9:0] BulletY, [9:0] Bullet_size, [9:0] BaddieX, [9:0] BaddieY, [9:0] Baddie_size, [9:0] BaddieX1, [9:0] BaddieX2, [9:0] Baddie1_size, [9:0] BaddieX2, [9:0] BaddieY2, [9:0], Baddie2_size, [9:0] Baddie_BulletX, [9:0] Baddie_BulletY, [9:0] Baddie_bullet_size, [9:0] WallX(0-7), [9:0] WallY(0-7), [9:0]WallH(0-7), [9:0] WallL(0-7), [3:0] room, Clk, Reset, blank, pixel_clk, frame_clk, [1:0] direction, [1:0] fraddie_direction, [1:0] test_direction, invincible, star, key1, key2, [2:0] player_health, [1:0] fraddie_health,[1:0] test_health

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

**Description**: A short list of what everything in this module does:

- Inputs
    - BallX, BallY, and Ball_size are the same names from Lab6, they are used to describe our main character, Gus
    - BulletX, BulletY, and Bullet_size are used to describe the bullet that Gus shoots
    - BaddieX, BaddieY, and Baddie_size are used to describe the villain that we have in the game.
        - Baddie1 and Baddie2 varaibles are for additional villians
    - Baddie_bulletX, Baddie_bulletY, and Baddie_bullet_size are used to describe the bullet of our shooting enemy
    - WallX and WallY are used to determine the top left corner of a rectangular wall
        - There are 7 others named in convention WallX2 and WallY2 for example, for a total of 8 walls
    - WallL and WallH are used to describe the height and length of the rectangular wall
        - There are 7 others named in convention WallL2 and WallH2 for example, for a total of 8 walls
    - Clk is the Max10 50Mhz clock
    - Reset is the Reset signal coming from the key on the FPGA
    - blank is the blanking interval coming from the vga_controller, we need to respect this in the color mapper for a smooth refresh
    - pixel_clk is the pixel_clk coming from the vga_controller as well
    - frame_clk is the VGA_VS, or the vertical refresh rate of VGA, 60hz
    - direction is the 2 bit value of the direction of the player used to place the sprites accordingly
    - fraddie_direction is the direction of the first baddie
    - test_direction is the direction of the second baddie
    - invincible tells whether the player is using the invincible powerup
    - star indicates whether the player has picked up the star powerup

    - key1 is an indicator that the player has picked up the first key
    - key2 is an indicator that the player has picked up the second key
    - player_health is a register to hold the 5 lives of the player
    - fraddie_health is the health of the first enemy
    - test_health is the health of the second enemy
- Outputs
    - Red, Green, and Blue are used to display the proper hex values to the desired pixel

**Purpose**: This module oversees putting everything on the screen at the right time and in the right spot, respecting the blanking interval. Without it the screen would just be blank. This module also instantiates and indexes all the sprites used in this project.

**Module**: character.sv (in character.sv)

**Inputs**: Reset, frame_clk, [7:0] keycode, [7:0] keycode1, [7:0] keycode2, [7:0] keycode3, [7:0] keycode5, Clk

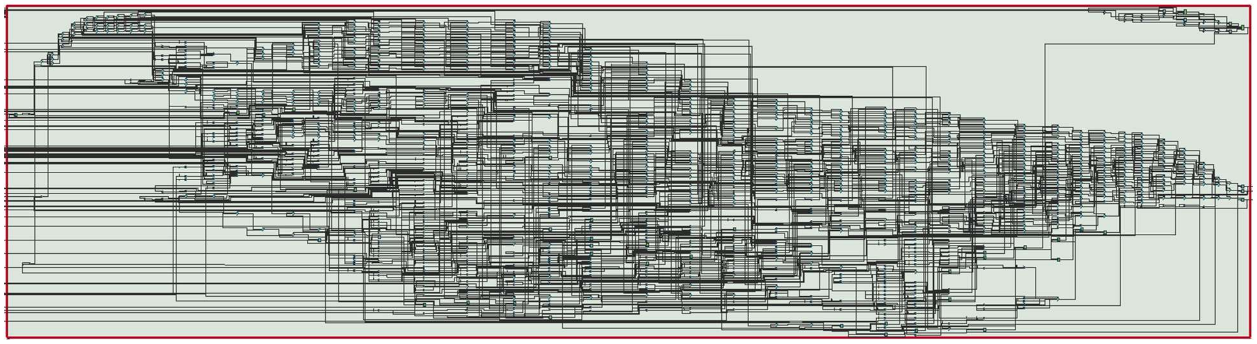
**Outputs**: [9:0] BallX, [9:0]  BallY, [9:0] BallS, [9:0] BulletX, [9:0] BulletY, [9:0] BulletS, [9:0] BaddieX, [9:0] BaddieY, [9:0] BaddieS, [9:0] BaddieX1, [9:0] BaddieX2, [9:0] BaddieS1, [9:0] BaddieX2, [9:0] BaddieY2, [9:0], BaddieS2, [9:0] Baddie_BulletX, [9:0] Baddie_BulletY, [9:0] Baddie_bulletS, [9:0] WallX(0-7), [9:0] WallY(0-7), [9:0]WallH(0-7), [9:0] WallL(0-7), [3:0] room, roomChange,  Clk, Reset, blank, pixel_clk, frame_clk, [1:0] direction_out, [1:0] baddie_dir_out, [1:0] baddie_dir_out2, invincible, star, key1, key2, [2:0] player_health, [1:0] health_out,[1:0] health2_out

**Description**:  A short list of what everything in this module does:

- Inputs

  - Reset is the Reset signal coming from the key on the FPGA
  - Clk is the Max10 50Mhz clock
  - frame_clk is the VGA_VS, or the vertical refresh rate of VGA, 60hz
  - All the keycodes come from the keyboard input, used to enable multiple key presses
- Outputs
  - BallX, BallY, and BallS are the same names from Lab6, they are used to describe our main character, Gus
  - BulletX, BulletY, and BulletS are used to describe the bullet that Gus shoots
  - BaddieX, BaddieY, and BaddieS are used to describe the villain that we have in the game.
    - Baddie1 and Baddie2 variables are for additional villains
  - Baddie_bulletX, Baddie_bulletY, and Baddie_bulletS are used to describe the bullet of our shooting enemy
  - WallX and WallY are used to determine the top left corner of a rectangular wall
    - There are 7 others named in convention WallX2 and WallY2 for example, for a total of 8 walls
  - WallL and WallH are used to describe the height and length of the rectangular wall
    - There are 7 others named in convention WallL2 and WallH2 for example, for a total of 8 walls
  - direction_out is the 2 bit value of the direction of the player used to place the sprites accordingly
  - baddie_dir_out is the direction of the first baddie
  - baddie_dir_out2 is the direction of the second baddie
  - invincible tells whether the player is using the invincible powerup
  - star indicates whether the player has picked up the star powerup
  - key1 is an indicator that the player has picked up the first key
  - key2 is an indicator that the player has picked up the second key
  - player_health is a register to hold the 5 lives of the player
  - health_out is the health of the first enemy
  - health2_out is the health of the second enemy

**Purpose**: This module is where all the core interaction logic happens, all of the damage, collisions, instantiations of all of the sub modules happen here.



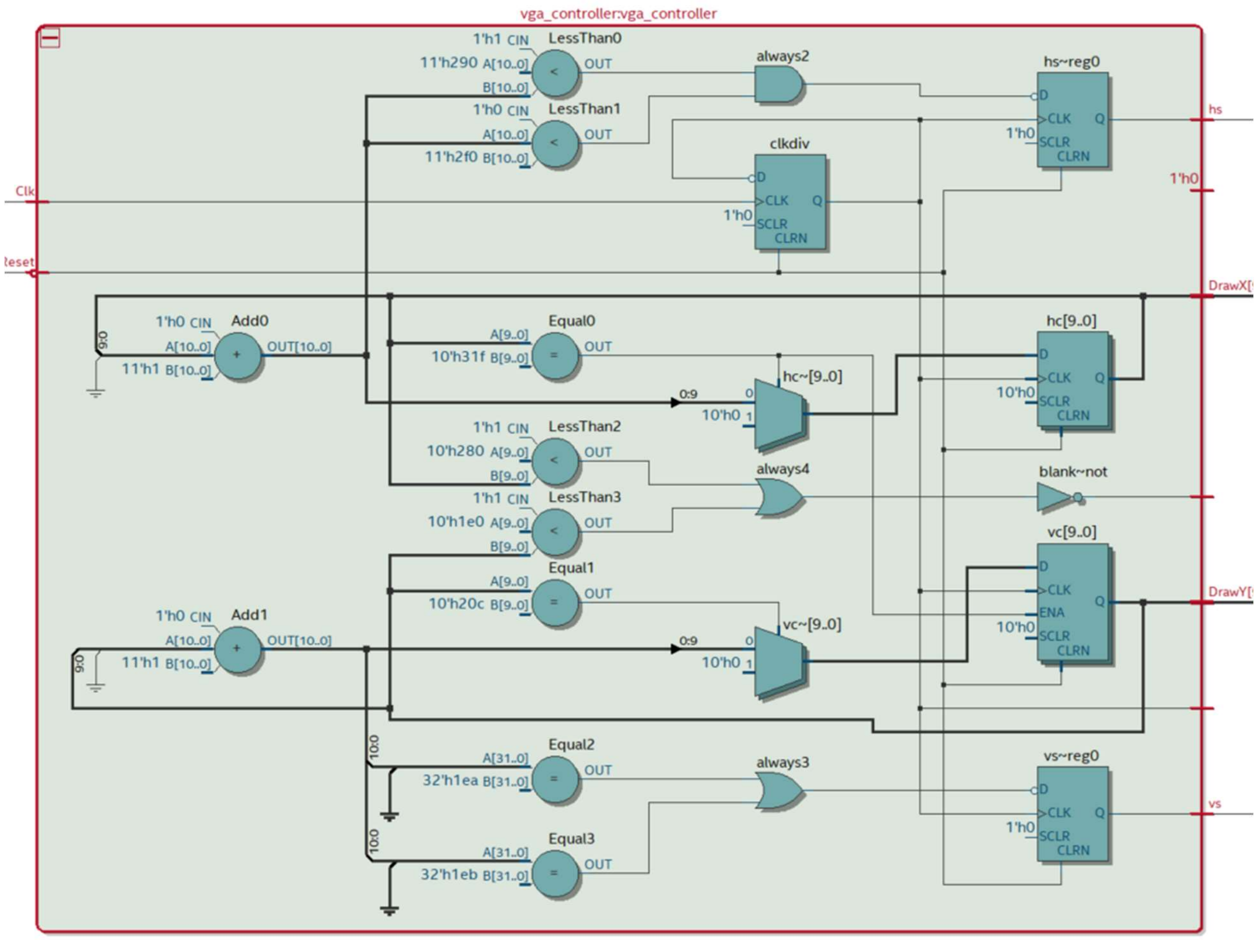
**Module**: vga_controller (in vga_controller.sv)

**Inputs**: Clk, Reset

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

**Description**: This module controls the timings and locations on the vga output itself. It simply takes in the Clk and Reset and outputs the vsync and hsync, as well as the pixel_clk, sync, blank, and the coordinates DrawX and DrawY.

**Purpose**: This module is important to making sure the vga output it properly timed and also not blurring and displaying incorrect pixels.



**Module**: countdown (in timers.sv)

**Inputs**: Clk, Reset, [7:0] seed, Enable

**Outputs**: Done, [7:0] Temp

**Description**: This module takes in the frame_clk at 60 hz and uses the frame_clk to count down from the seed to 0, 60 times a second. We were able to take advantage of this by sending in a seed, and counting down to create a timer

**Purpose**: This module was used as an invincibility timer, a enemy hit timer, and an animation timer

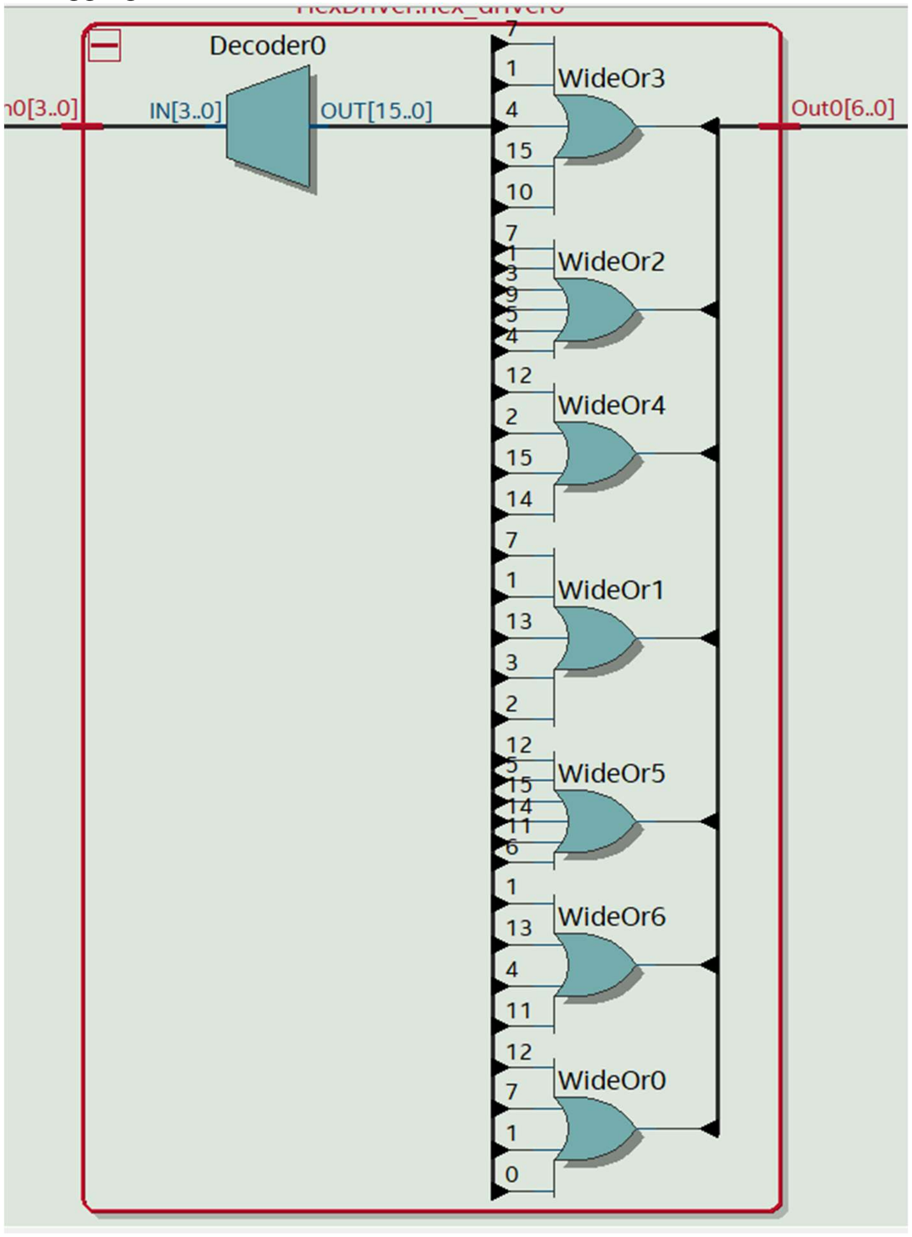
**Module**: HexDriver (in HexDriver.sv)

**Inputs**: [3:0] In0

**Outputs**: [6:0] Out0

**Description**: This module pairs every unique case of 4-bits (16 in total) into a 7-bit value used to illuminate the hex displays.

**Purpose**: This was used to output the values we were working with to the hex displays, both for debugging and for the demo itself.



**Modules**: ECE_life_soc (in ECE_life_soc.qip)

**Description**: Below I will describe the PIO blocks added in addition to the ones needed for the NIOS II processor to complete the platform designer.

**Purpose:** All these modules come together to make a working NIOS II processor with memory and USB abilities.

**Module**: ram modules (in gus.sv, bus.sv, backgrounds.sv, and signs.sv)

**Inputs**: [variable:0] write_address, [variable:0] read_address, we, Clk

**Outputs**: [23:0] data_Out

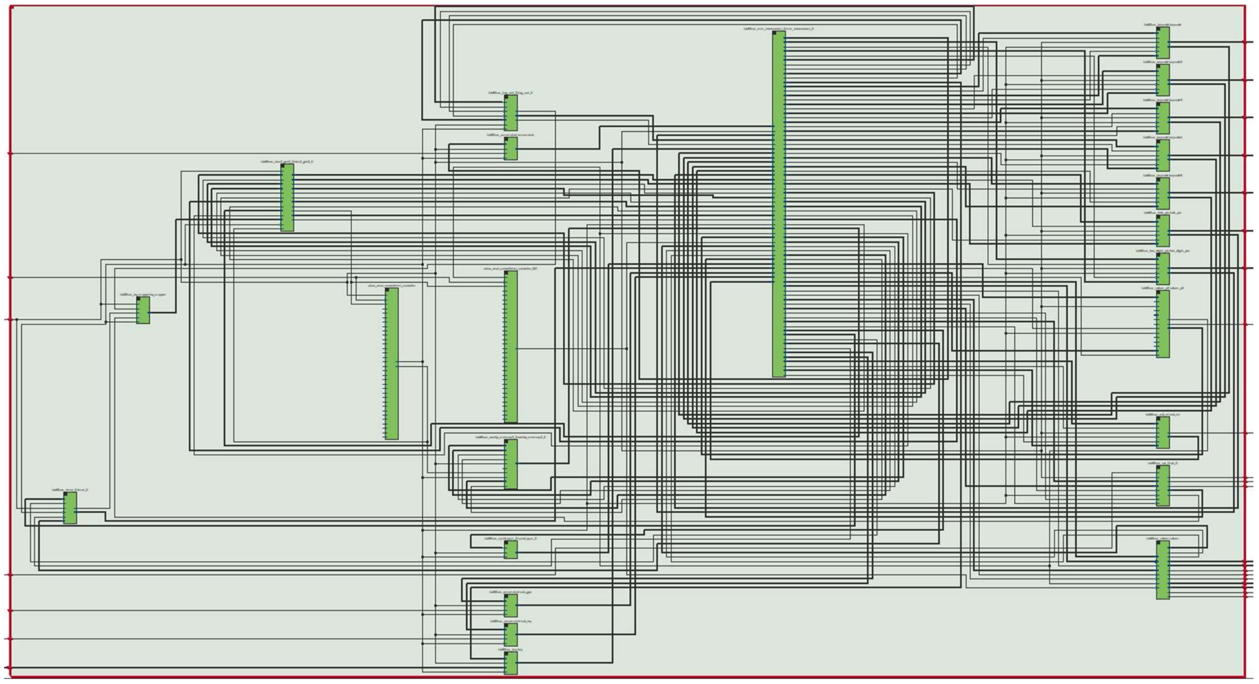
**Description**: We instantiated the ram the same way for every single sprite to index into it:

- write_address: we did not use this because we treated the rams as roms
- read_address: this was set as the number of bits needed to index to the bottom of the sprite text file
- we: this was always set to 0 so we never wrote to the ram
- Clk: used to clock the ram
- data_Out: 24 bits used to make output the section of the text file we are using in order to output the hex of the sprite

**Purpose**: These modules were used to instantiate all of the ROMS that we needed for all of the sprites in our project

## System Level Block Diagram

If not explicitly mentioned, the SOC component was used in both parts of the lab. An overview of the SOC is shown below:



The rest of this section goes into further detail about the different IP's within the platform designer, generated within the SOC.

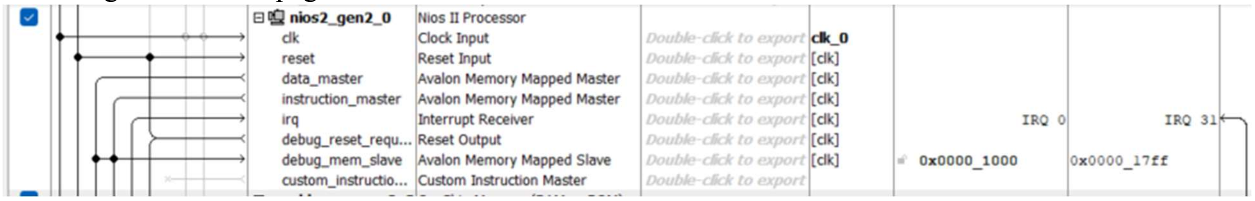
**Module:** nios2_gen2_0 (in ECE_life_soc.qip)

**Connections:** clk, reset, data_master, instruction_master, irq, debug_reset, debug_mem_slave

**Exports:** None

**Description:** This is the central processor that everything is connected to. It is a NIOS II Processor from Intel.

**Purpose:** This module is what everything else is connected to. The block diagram for this module is far to large to fit into a page.



| | | | | | |
|---|---|---|---|---|---|
| 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 | | | |

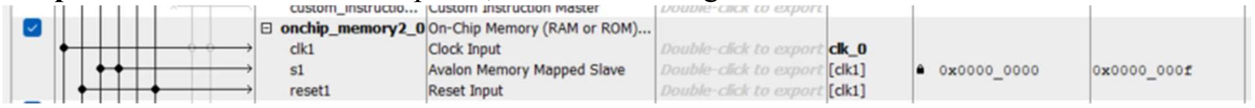
**Module:** onchip_memory2_0 (in ECE_life_soc.qip)

**Connections:** clk1, reset1, Nios II Data bus

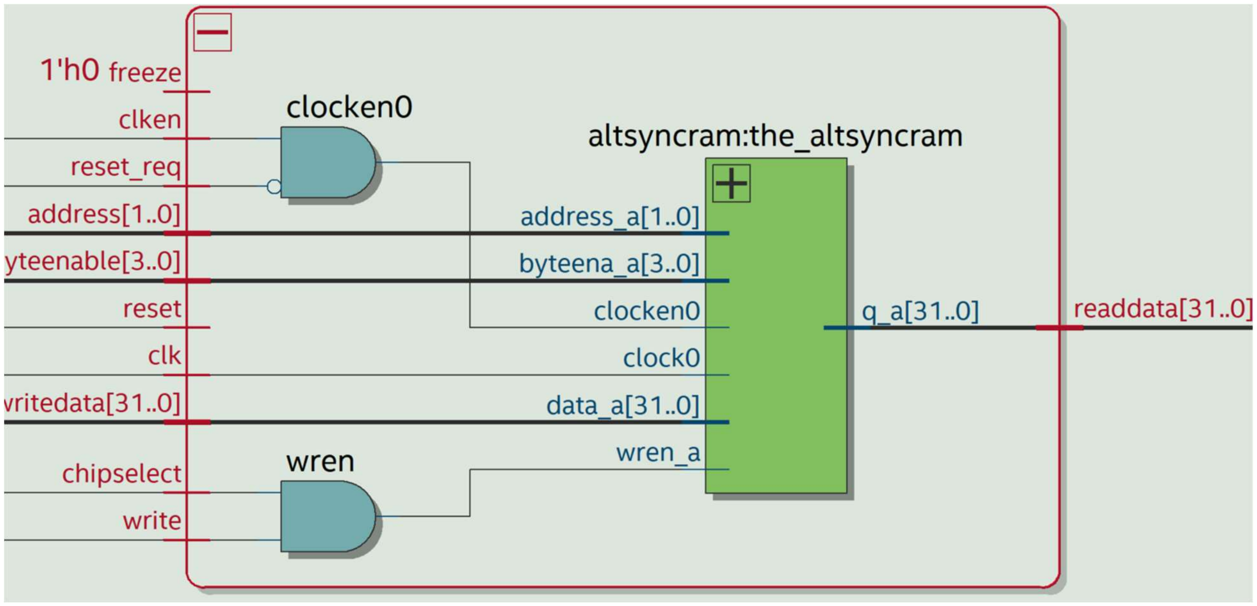
**Exports:** None

**Description:** This is an On-chip memory module used for the on chip memory on the FPGA

**Purpose:** This module is not required, but makes storage easier



| | | | | | |
|---|---|---|---|---|---|
| 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] | | |



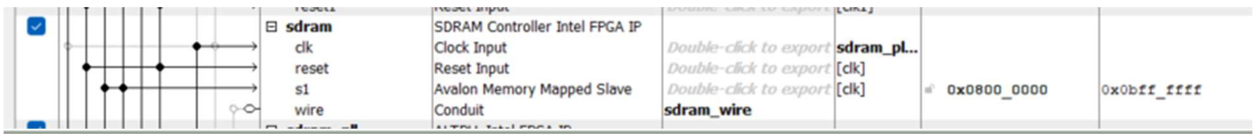
**Module:** sdram (in ECE_life_soc.qip)

**Connections:** clk1, reset1, Nios II Data bus

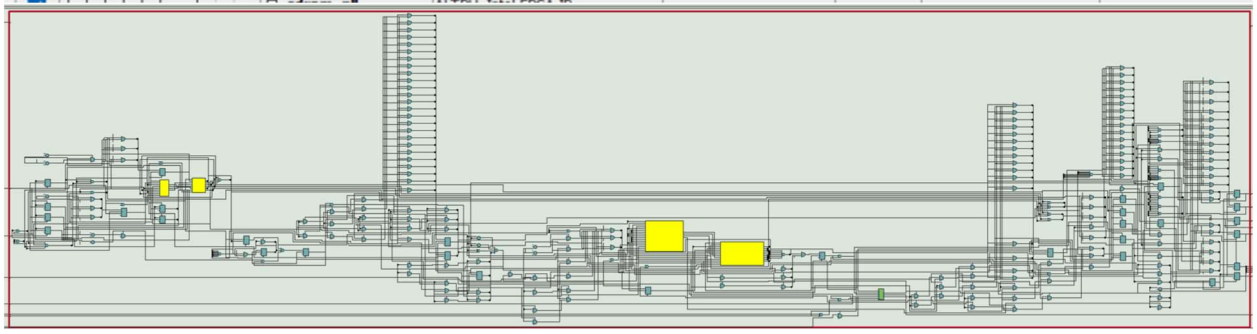
**Exports:** sdram_wire

**Description:** This is the functional synchronous memory for the NIOS II processor.

**Purpose:** Without this, the processor would not be able to store any data.



| | | | | |
|---|---|---|---|---|
| ⊟ **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** | | |



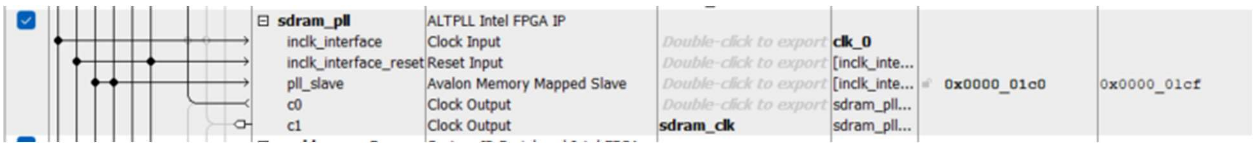
**Module:** sdram_pll (in ECE_life_soc.qip)

**Connections:** clk, reset, Nios II Data bus

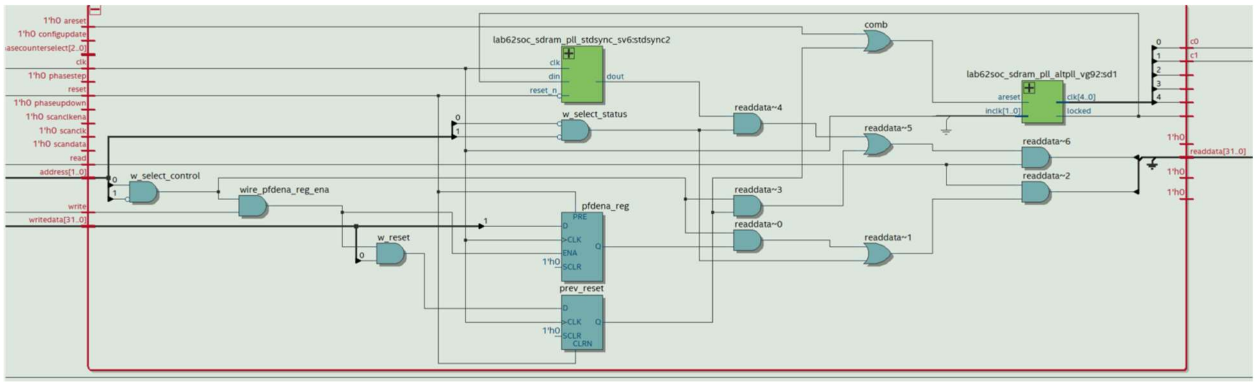
**Exports:** sdram_clk

**Description:** This is the module that is used to control the SDRAM, specifically clocking it at the right frequency.

**Purpose:** Without this module, the SDRAM would not work well, if at all, because of asynchronous clocking times.



| | | | | | |
|---|---|---|---|---|---|
| ⊟ **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... | | |



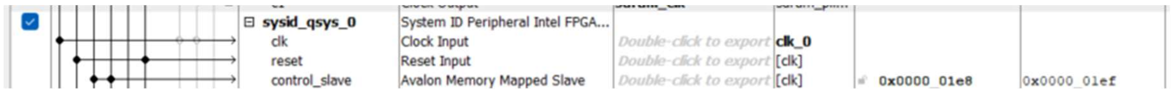
**Module:** sysid_qsys_0 (in ECE_life_soc.qip)

**Connections:** clk, reset, control_slave

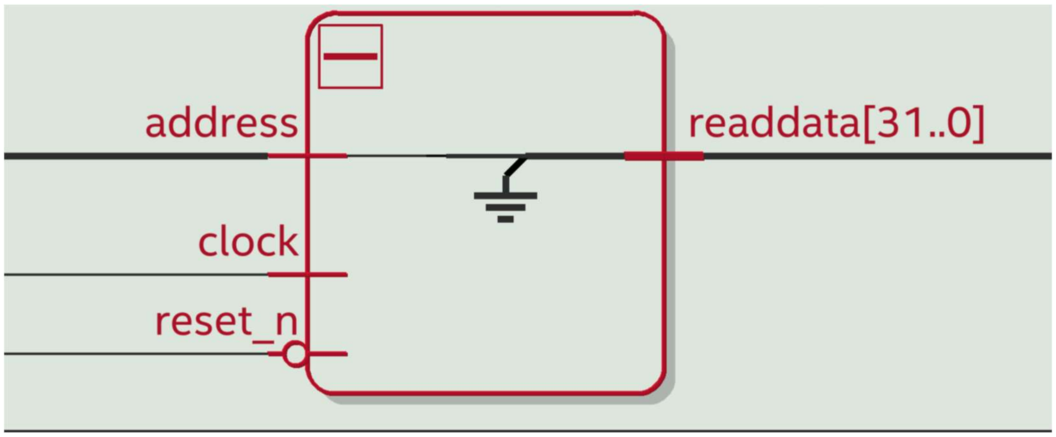
**Exports:** None

**Description:** This module acts as a manager between the hardware and the BSP that is generated.

**Purpose:** Without this module, the BSP would not work, and in turn the c code could not be run on the FPGA .



| | | | | |
|---|---|---|---|---|
| ⊟ 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 |



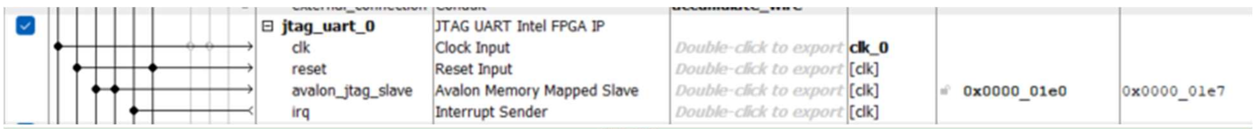
**Module:** JTAG UART Intel FPGA(in ECE_life_soc.qip)

**Connections:** clk, reset, Nios II Data bus, IRQ

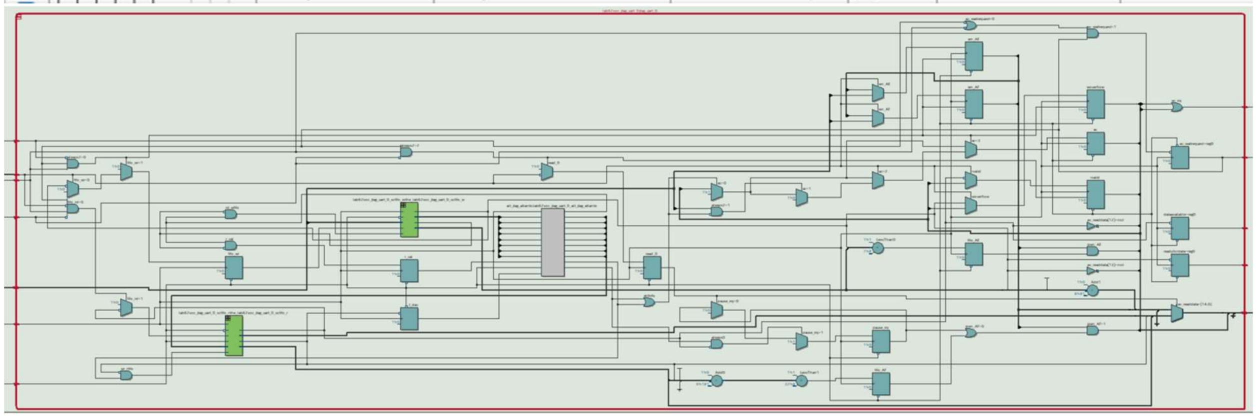
**Exports:** None

**Description:** This is a module within the platform design that is specifically used for USB connections.

**Purpose:** This allows for the printf() command to be used in the eclipse IDE.



| | | | | |
|---|---|---|---|---|
| ⊟ **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_01e0 | 0x0000_01e7 |
| irq | Interrupt Sender | Double-click to export | [clk] | |



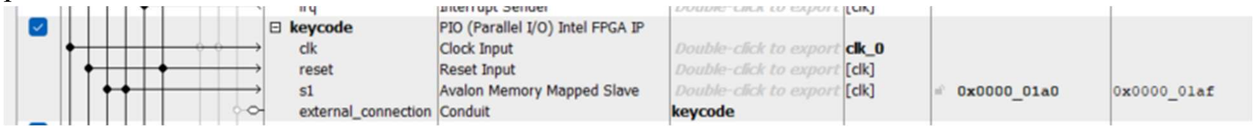
**Module:** keycode (in ECE_life_soc.qip)

**Connections:** clk, reset, Nios II Data bus, IRQ

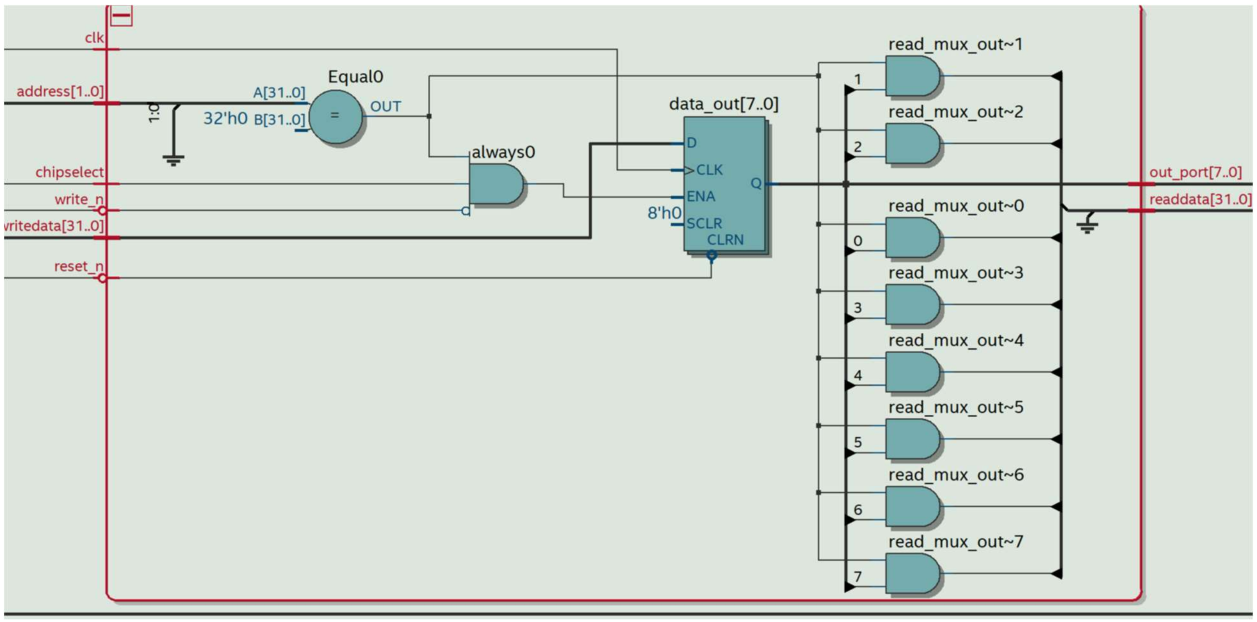
**Exports:** keycode

**Description:** This is a PIO designed to take in a keycode from a peripheral and send it to the NIOS II processor. It should be noted that we made 5 of these in our platform designer to have 5 keycodes.

**Purpose:** This PIO is imperative to get the proper signal from the keyboard to the NIOS II processor.



| | | | | | |
|---|---|---|---|---|---|
| ⊟ **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_01a0 | 0x0000_01af |
| external_connection | Conduit | **keycode** | | | |



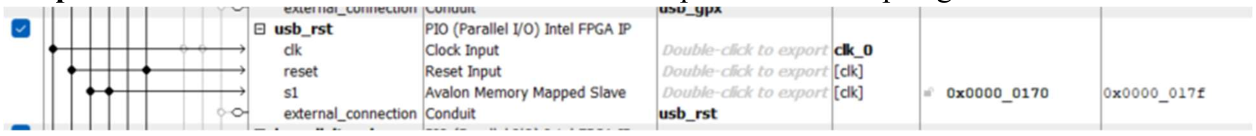
**Module:** usb_rst (in ECE_life_soc.qip)

**Connections:** clk, reset, Nios II Data bus

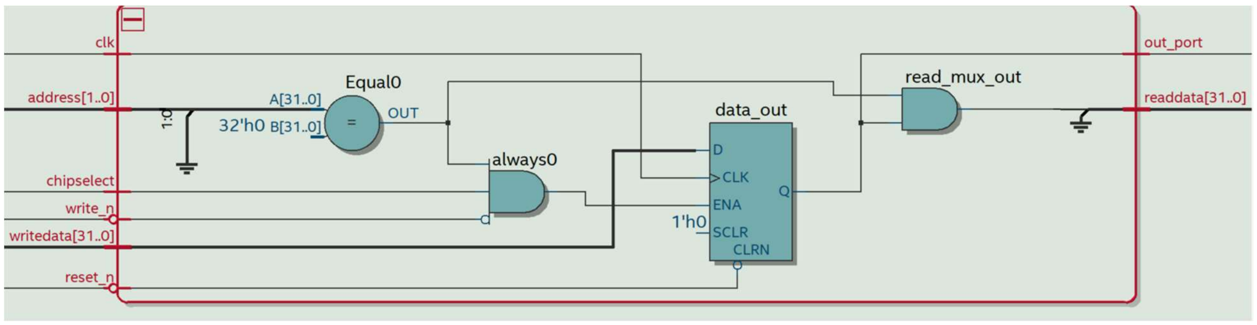
**Exports:** usb_irq

**Description:** This is a PIO used within the USB.

**Purpose:** This PIO is used to make sure the USB responds to interrupt signals.



| | | | | | |
|---|---|---|---|---|---|
| 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_0170 | 0x0000_017f |
| external_connection | Conduit | usb_rst | | | |



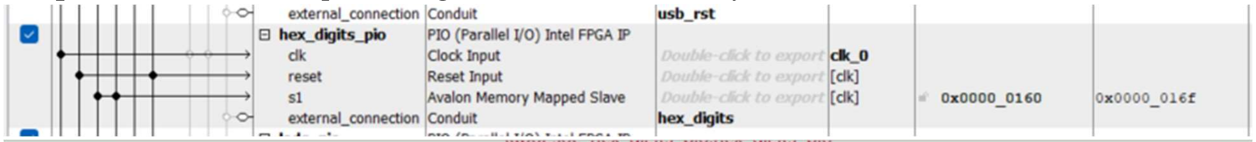
**Module:** hex_digits_pio (in ECE_life_soc.qip)

**Connections:** clk, reset, Nios II Data bus

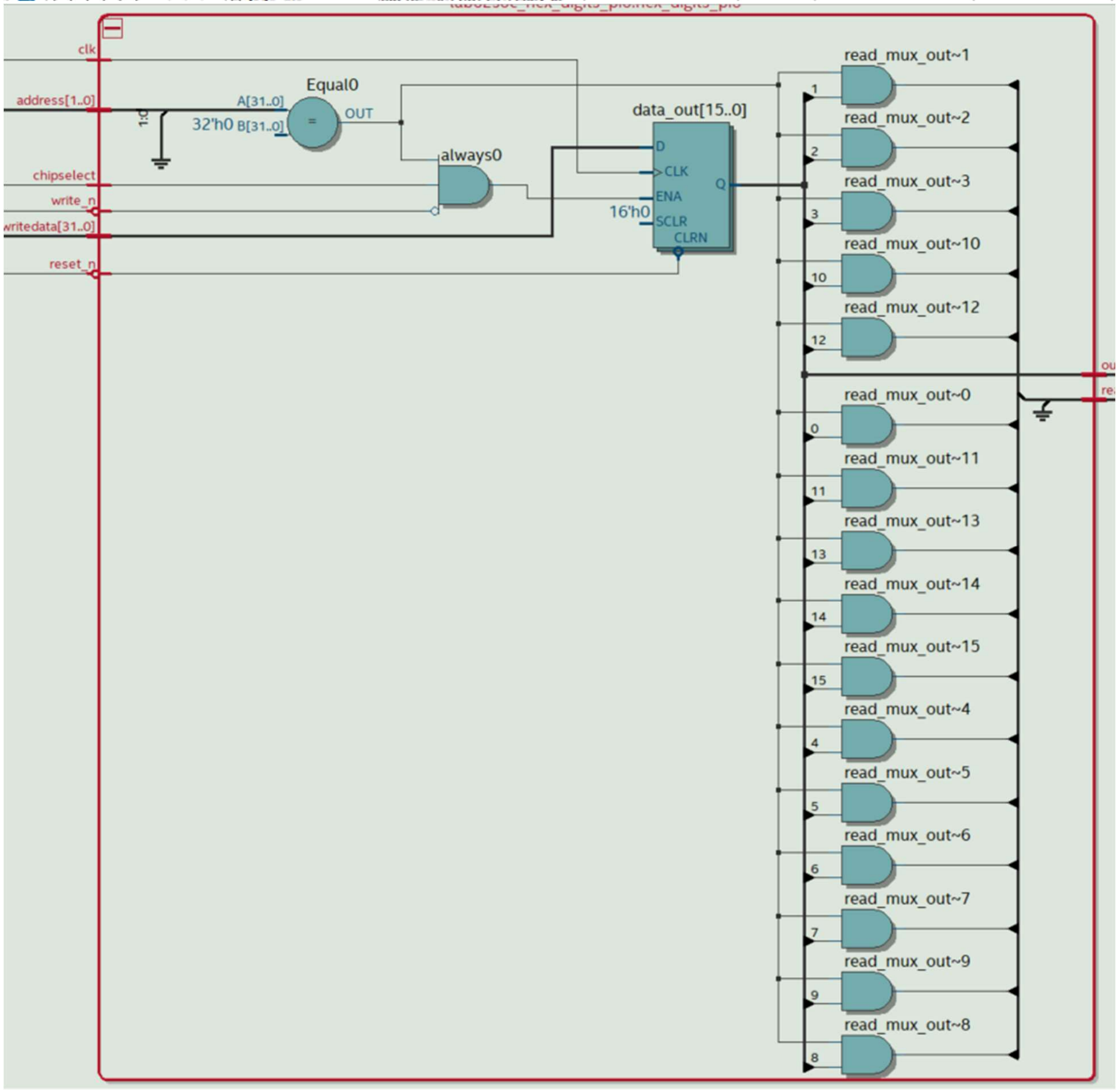
**Exports:** hex_digits

**Description:** This is the PIO used to print values to the hex displays coming from the keyboard. This was for part 2 of the lab.

**Purpose:** This PIO is required to get the hex from the keycodes to the Hex Drivers.



| | | | | |
|---|---|---|---|---|
| external_connection | Conduit | usb_rst | | |
| ⊟ 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_0160 | 0x0000_016f |
| external_connection | Conduit | hex_digits | | |



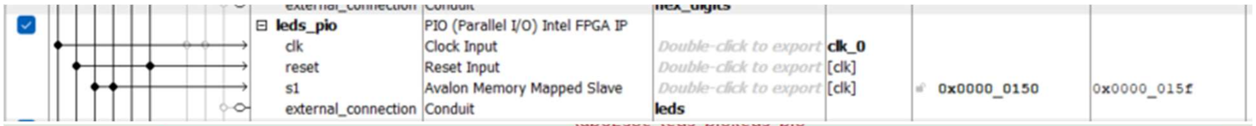
**Module:** leds_pio (in ECE_life_soc.qip)

**Connections:** clk, reset, Nios II Data bus

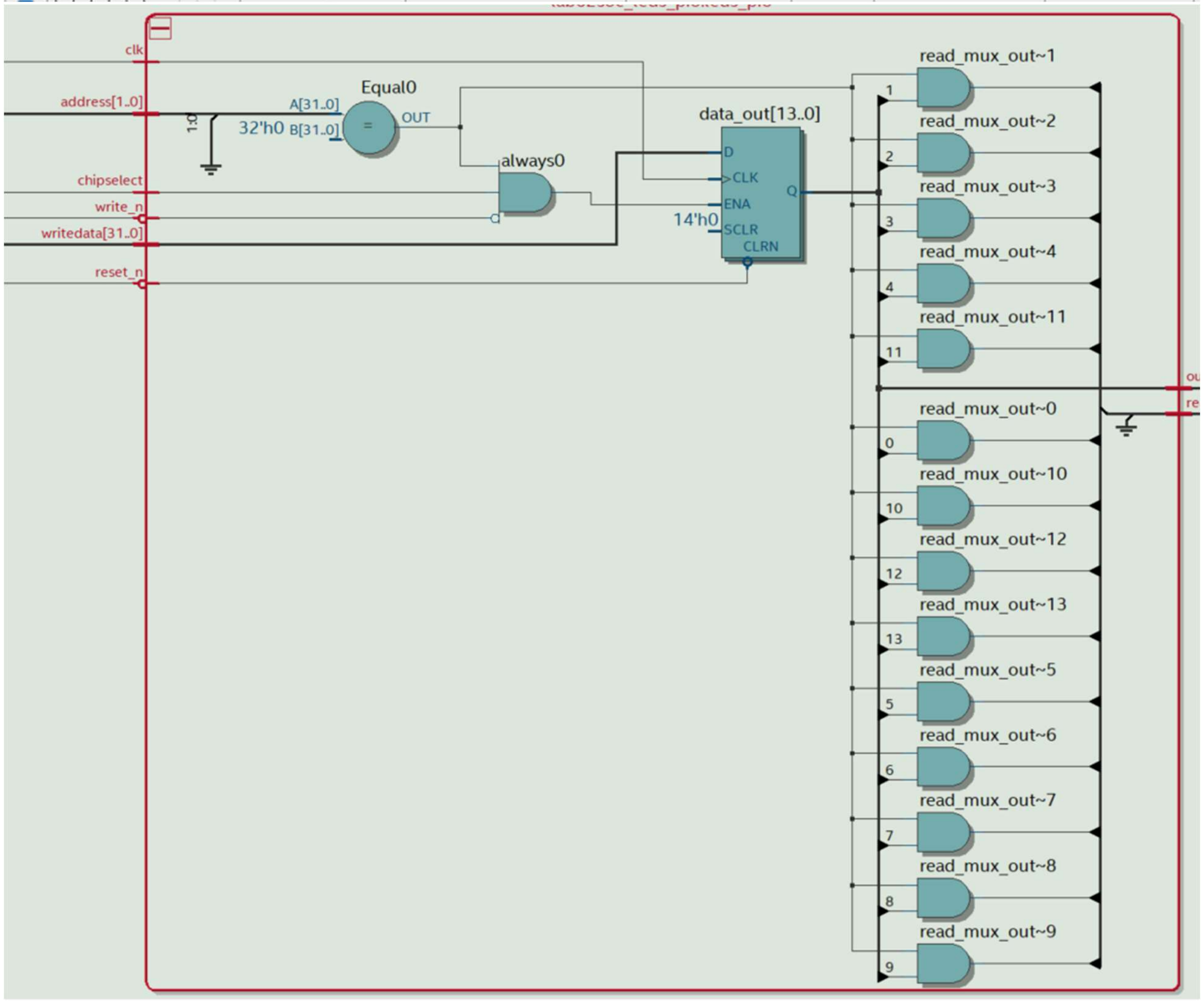
**Exports:** leds

**Description:** This is the PIO used to display onto the LEDs. This was for Part 1 of the lab.

**Purpose:** Without this PIO, it would not be possible to display the running sum from part one on the LEDs.



| | | | | | |
|---|---|---|---|---|---|
| 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_0150 | 0x0000_015f |
| external_connection | Conduit | leds | | | |



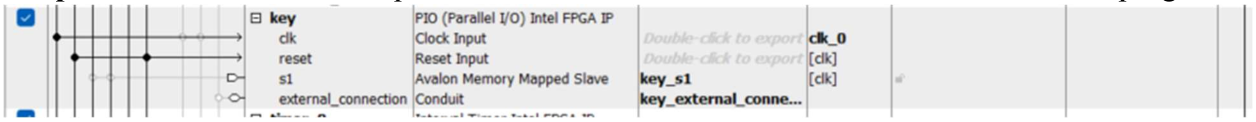
**Module:** key (in ECE_life_soc.qip)

**Connections:** clk, reset, Nios II Data bus

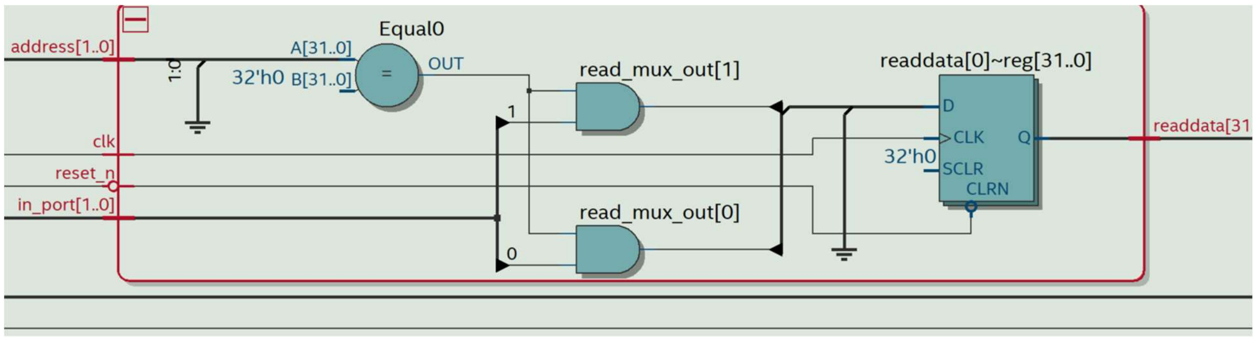
**Exports:** key_s1, key_external_connection

**Description:** This is the PIO used to detect key presses on the FPGA. KEY[0] is assigned to interact in different ways with the FPGA.

**Purpose:** This module was required to make sure the KEY0 is usable as a reset for the program.



| | | | |
|---|---|---|---|
| key | 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 | key_s1 | [clk] |
| external_connection | Conduit | key_external_conne... | |



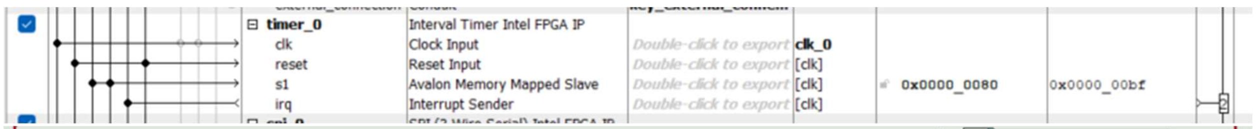
**Module:** timer_0 (in ECE_life_soc.qip)

**Connections:** clk, reset, Nios II Data bus, IRQ

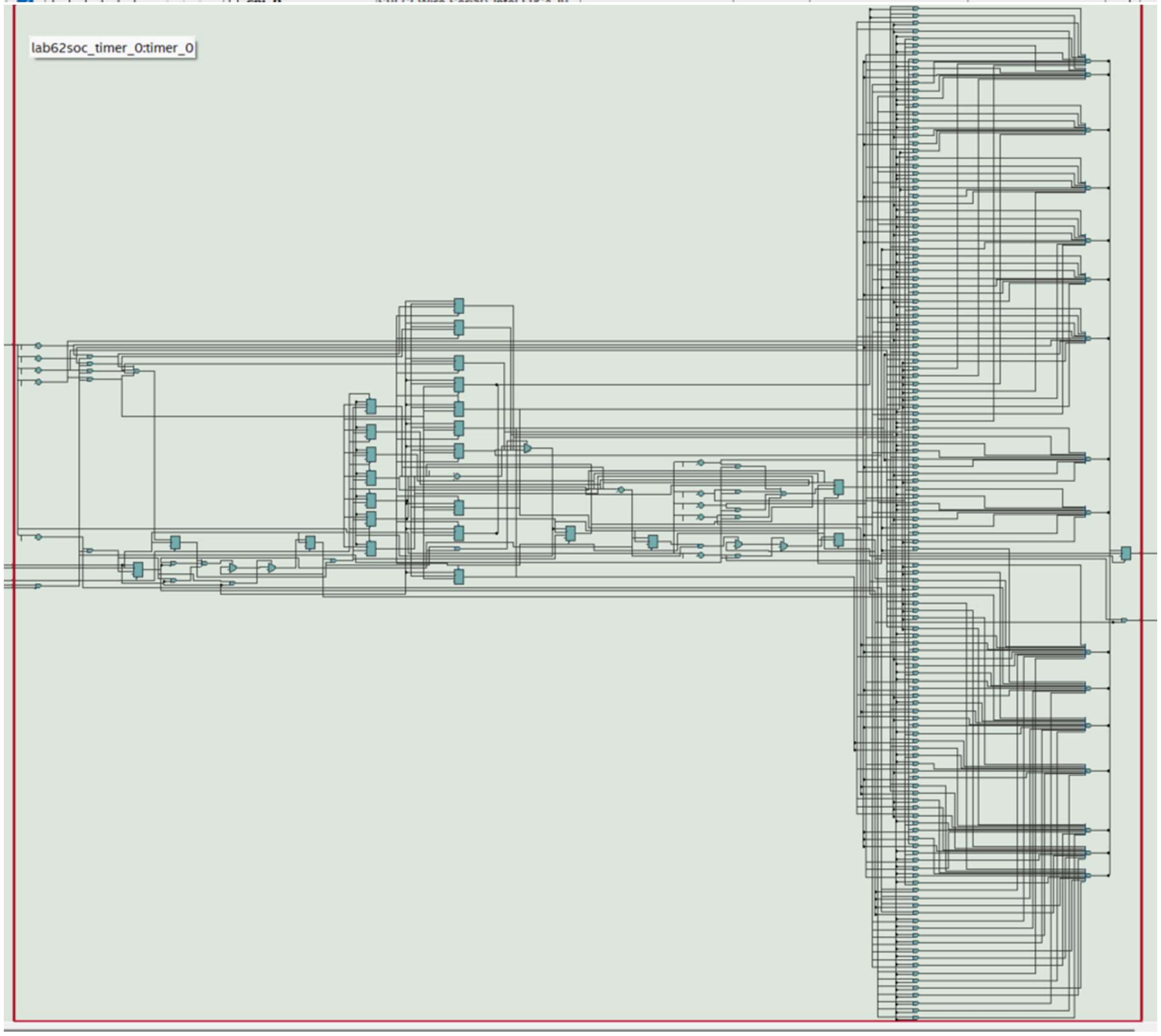
**Exports:** None

**Description:** This Intel timer IP is used to keep the timings in order. It is required for USB timings to be exact.

**Purpose:** This time is used to line up the timings with the IRQ as well as with the SPI.



| | | | | | |
|---|---|---|---|---|---|
| timer_0 | 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] | | |



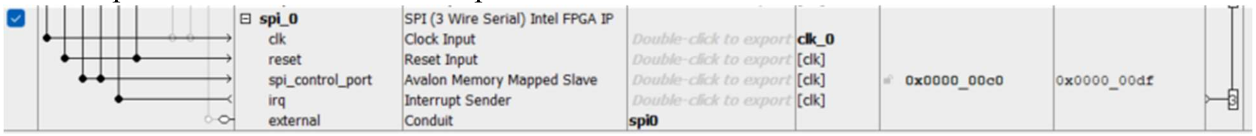
**Module:** spi_0 (in ECE_life_soc.qip)

**Connections:** clk, reset, Nios II Data bus, IRQ

**Exports:** spi0

**Description:** This is a SPI(3 Wire Serial) Intel IP and is responsible for the majority of the driver work between the USB and the NIOS II.

**Purpose:** This IP is the powerhouse between the NIOS II and the USB, and without it, a connection between the 2 would be infeasible. The block diagram for this section is far too large and complicated to fit into this lab report.



|  |  |  |  |  |  |
|---|---|---|---|---|---|
| ⊟ 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] |  |  |
| external | Conduit | spi0 |  |  |  |

## Software Component

1. **MAXreg_wr**
   This function writes a single bit to the MAX3421E register via the SPI. We created this function by creating a 2-length array X: {register + 2, value to written}. We then use the function "alt_avalon_spi_command(base, slave, write_length, write_data, read_length, read_data, flags)" to connect the MAX3421 to NIOS II. We use the address of the SPI, SPI_0_BASE, as the base, length of the write data is 2, the write data itself is the address of X, the other parameters are all 0. We then check if the return_code is not 0, if there was an issue with the connection, the return_code would be 0, thus we print an error message if this happens.
2. **MAXbytes_wr**
   This function writes multiple bits to the MAX3421E register via the SPI. We created this function by creating a (1 + number of bytes to write)-length array X. To create this array, we set the first value as register + 2, then we created a for loop to take each value from the data array and input into the array. The final X array looks as follows: {register + 2, data[0], data[1], … data[number of bytes – 1]}. We then use the function "alt_avalon_spi_command" to connect the MAX3421 to NIOS II. We use the address of the SPI, SPI_0_BASE, as the base, length of the write data is the number of bytes to write plus one, the write data itself is the address of X, the other parameters are all 0. We then check if the return_code is not 0, if there was an issue with the connection, the return_code would be 0, thus we print an error message if this happens. We then return the pointer of the last byte stored into the register, (data + number of bytes to write).
3. **MAXreg_rd**
   This function reads a single byte from the MAX3421E register via the SPI. We created a local variable to store the read value, val. We then use the function "alt_avalon_spi_command" to connect the MAX3421 to NIOS II. We use the address of the SPI, SPI_0_BASE, as the base, length of the write data is 1, the write data itself is the address of register, the read length is 1, and the place to store the read data is the address of val. We then check if the return_code is not 0, if there was an issue with the connection, the return_code would be 0, thus we print an error message if this happens. We then return the value read from the register, val.
4. **MAXbtyes_rd**
   This function reads multiple bytes from the MAX3421E register via the SPI. We created a local variable to store the read value, val. We then use the function

"alt_avalon_spi_command" to connect the MAX3421 to NIOS II. We use the address of the SPI, SPI_0_BASE, as the base, length of the write data is 1, the write data itself is the address of register, the read length is the number of bytes to be read, and the place to store the read data is the address of the data arry. We then check if the return_code is not 0, if there was an issue with the connection, the return_code would be 0, thus we print an error message if this happens. We then return the pointer of the last byte stored into memory, (data + number of bytes to write).

## Design Resources and Statistics

| | |
|---|---|
| LUT | 15,330 |
| DSP | 0 |
| Memory(B-Ram) | 974,992 |
| Flip-Flop | 280 |
| Frequency | 50 Mhz/ 60Hz |
| Static Power | 97.34mW |
| Dynamic Power | 226.34mW |
| Total Power | 345.50mW |

## Unique Features and Complexities

- Our design has 5 unique keycodes that all operate independently, allowing for it to take up to 5 different inputs at once
- Every single sprite in our design is completely custom and drawn by us
- Every interactable object in this game has collision detection with everything else, should they ever come in contact
- Internal timers on animation, collisions, and invincibility powerup
- Ability to not kill the enemies and still win the game, promoting speed running
- 5 completely unique rooms with different wall layouts and sprite backgrounds
- Animations on the winged enemies that are independent of each other
- Invincibility final can be found in the Daily Byte
- Fully working inventory with 2 keys and star invincibility powerup
- Player shot bullets can be curved by the player
- Enemies have a player tracking algorithm and follow the player no matter where the player is inside of the room
- When the player dies the enemies regain full health, but the players inventory does not get reset
- When the player dies, enemies run offscreen as their job is done
- Sean Kingston makes an appearance in the game
- Both keys are needed in the inventory to open the boss battle door
- Doors do not automatically open but rather need to be opened by clicking "F"
- There are 8 full functional walls of any size that can be placed anywhere within a room

- When enemies die, they stay still as a corpse, and then when shot again disappear
- The keys and star only show up once, meaning once the player has them, even if they go back and kill the enemy, the key will not show up again

## Baseline Features

- A working ECEB foyer with a student walking around based on input given by the user on a keyboard
    - Created a ECE Major sprite that moves with WASD inputs
    - Created ECE foyer background with tables and signs to other rooms
- The ability for the student to through different rooms including the Daily Byte, TI Design Room, Grainger Auditorium, and the Basement
    - Created a room structure with a total 5 rooms to explore
- Small Minigames or interactable items within each sub-area
    - Have to fight unique enemies in each sub-area
        - Tests, Frat Bro, and Slime monster
    - There are keys and a star to pick up through the ECE Building
- An ability to keep an inventory and check when the game should be over
    - There is an inventory that stores all 3 items, the 2 keys and star
    - Once the Slime monster is defeated and the basement is unlocked, the game is over
- A cursor controlled by the mouse can be used to interact with environment elements
    - Did not implement this as after looking at the inspiration for our game closer we realized that there was no point to use the mouse for what our game wanted to be

## Additional Features

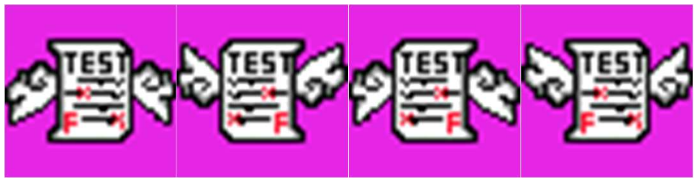
- Complex minigames that include a switch from a “point and click” to” platformer” or other types of games depending on the room
    - The main gameplay of the game stays the same throughout the room but each room has unique backgrounds and enemy types
- The addition of a secondary player that is controlled by a second player, used solely to create obstacles for the main character as they progress through the game
    - Not implemented

- Fluid animation between the rooms rather than cutting to black screen and then coming back from black screen
    - The character fluidly moves from room to room, i.e if the player went into a door on the left the player would appear on the right side of the screen at the same Y position that the player was before.
- Implementation of Non-Playable Characters (NPC) to interact with within the foyer
    - Not Implemented
- More complex check of the game being over (i.e. has the player picked up all the items AND visited all possible places)
    - The Basement must be unlocked to complete the game. In order to do this both the auditorium and the design room must be visited to get the keys needed to unlock the basement. Thus in order to complete the game the player must at least collect the 2 keys, visit the basement, and defeat the slime monster.
- An inventory menu that displays in real time what the player is holding as well as a map
    - There is an inventory indicator in the bottom left corner that updates in real time whenever a item is collected
    - Map was not implemented
- The implementation of a controller as an alternate input
    - Not implemented
- Create physics for platformer (jumping)
    - Platformer aspect not implemented
- Create collision detection for minigames
    - Created collisions between every interactable object, i.e. the walls stopping the player and bullets, and the bullets damaging the enemies and the player

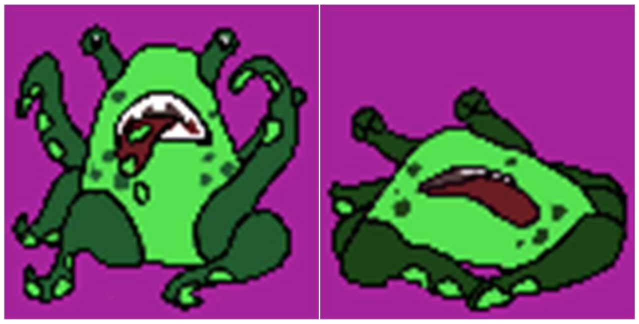
## Sprite Gallery

Above is our main character Gus

Above is our first antagonist "Business Frat Guy"

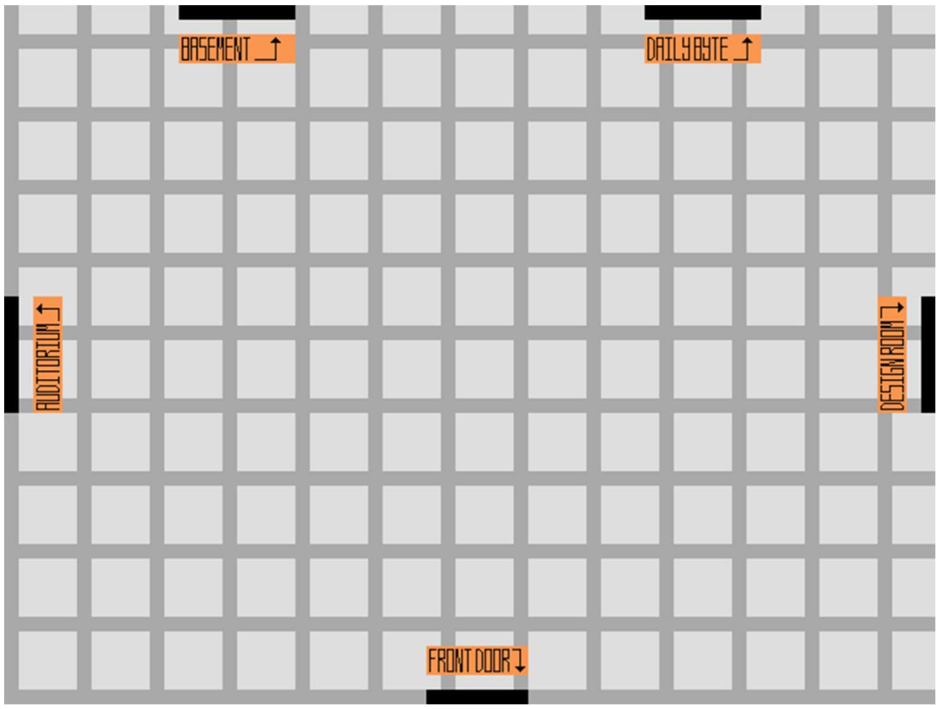


Above is our test enemy in various states of animation


Above is the final boss alive and dead

Above is the star and key for the inventory, as well as the player heart

SPRING JAM 2023
FEAT.
SEAN KINGSTON
Above are two examples of walls that we used, Sean Kingston as well as a design room table



This is the main lobby of the ECEB that we made

A more detailed version of the game and sprites can be seen in the video taken during our demo.

## Sprites Algorithm

i. Take the current X position of where we want to draw the sprite, and subtract it by the current X position of the VGA Electron Beam, giving the value DistX. Do the same for Y to get DistY.
ii. We then define the address to access to the sprite ROM with the formula: DistX + DistY * (Width of the Sprite)
iii. We then check if the current value of DistX is a value within the range of the width of the sprite and that DistY is a value within the range of the length of the sprite. If so, the indictator to print the sprite, Sprite_on.
iv. We then check if Sprite_on is on, if it is, the RGB values given by the current address of the Sprite ROM are output to the VGA Monitor