Alberto Diaz - UID: 604967268

Evan He - UID: 705619333

Brandon Truong - UID: 705326387

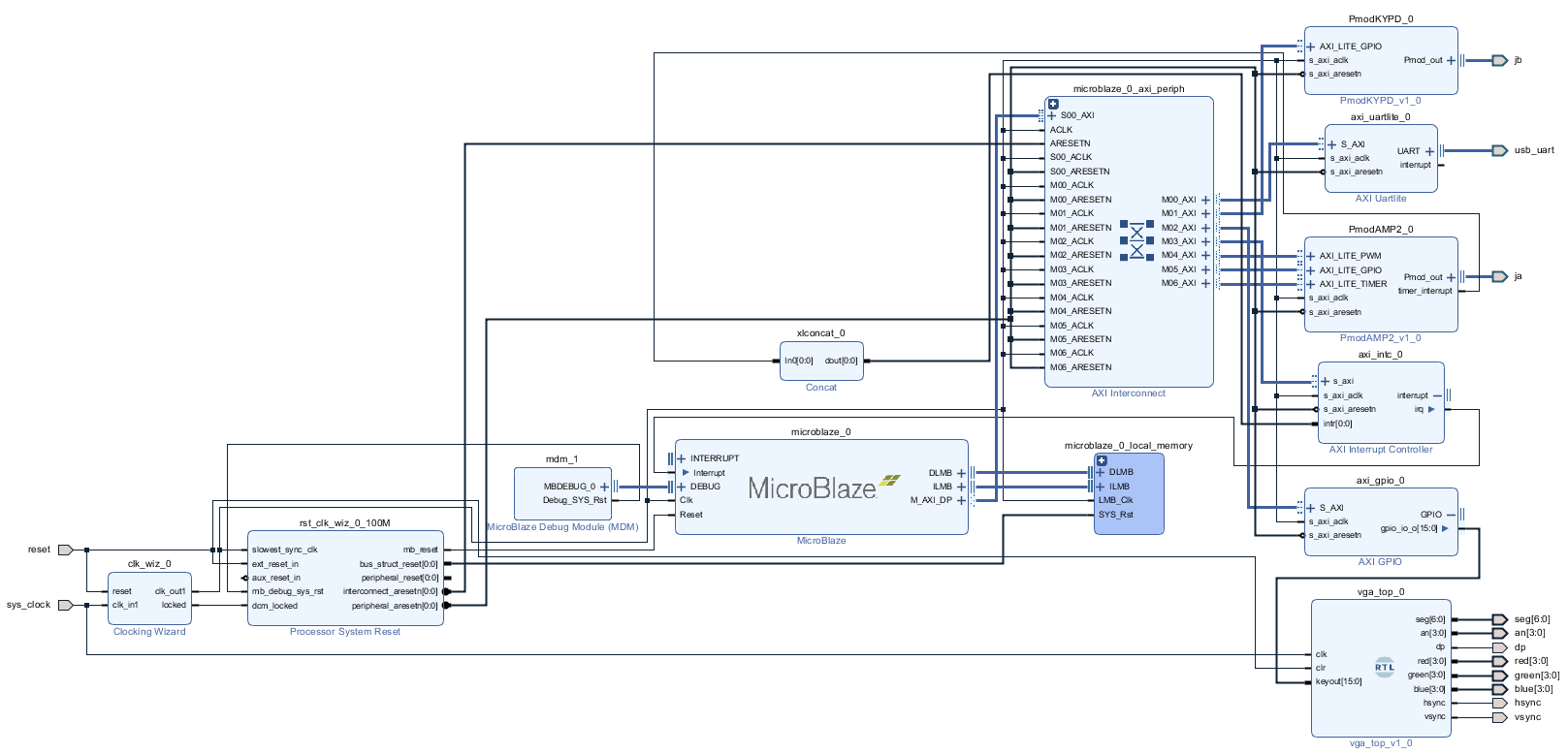
Final Project Report

**Introduction**

For our final project, we eventually decided to implement a piano simulation using the VGA display, PmodKYPD, and PmodAMP2. The final product works by receiving input from the user through the PmodKYPD. On the VGA display would be an image of a single octave on the piano keys, starting and ending in C on different octaves. The user can press any of the keys on the PmodKYPD to simulate a keypress on the piano. Each key is mapped to a single key on the piano display and a keypress would be represented by changing the color of that key to blue. Each key is also mapped to a certain frequency that corresponds to piano notes starting on middle C.

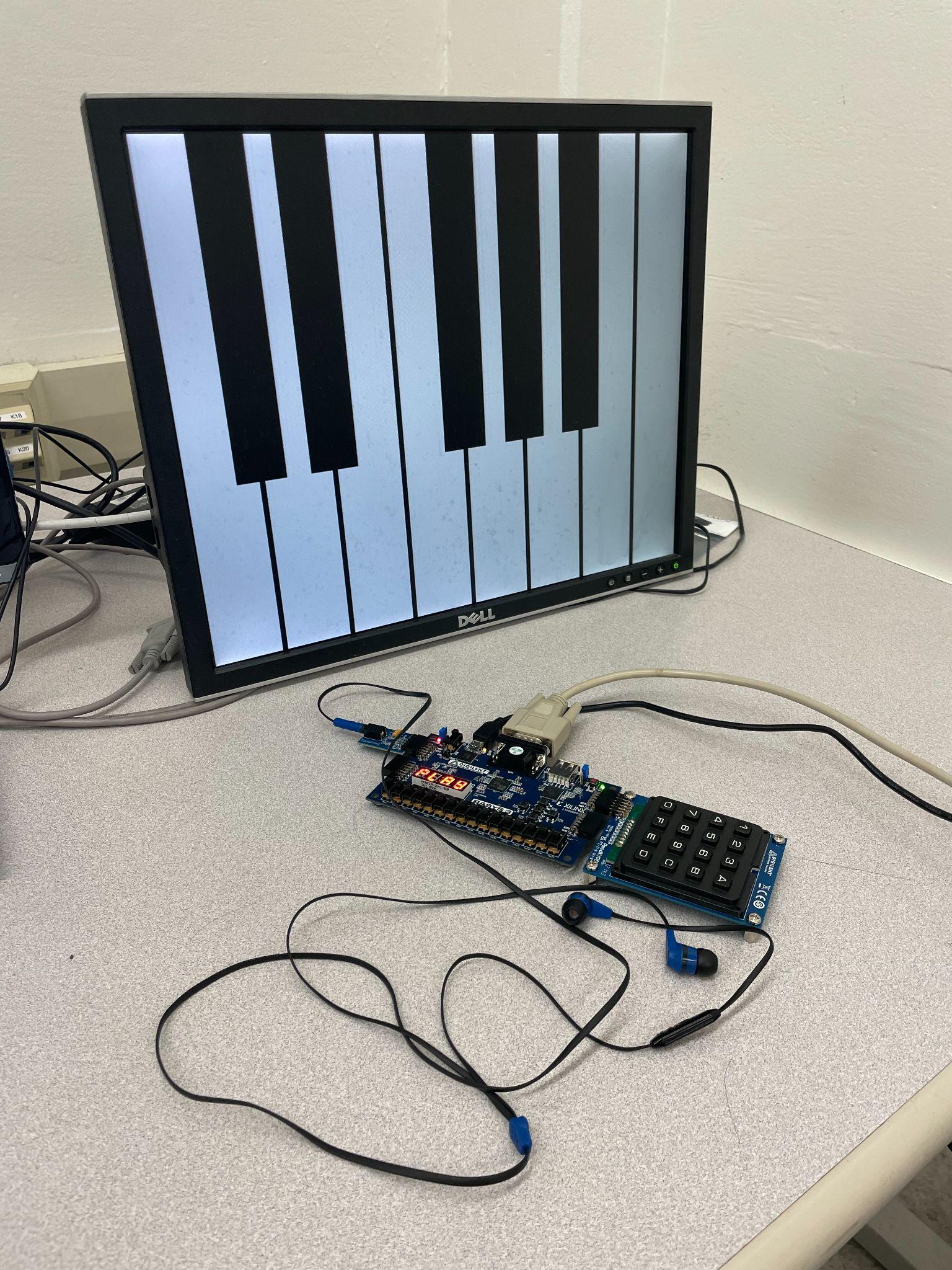
Although the PmodKYPD and PmodAMP2 have MicroBlaze-compatible IP cores that make implementations easier, there is no such IP core for the VGA display. Thus, we decided to implement the VGA display using Verilog and the other two peripherals using C code. One of the trickier parts of this project was finding a way to have our modules written in C to communicate with the VGA RTL so that the keypad can instruct which piano keys needed to be pressed on the display. Luckily, we were able to find a way to use the AXI\_GPIO module to facilitate communication between the PmodKYPD and the VGA display.

**Schematic**



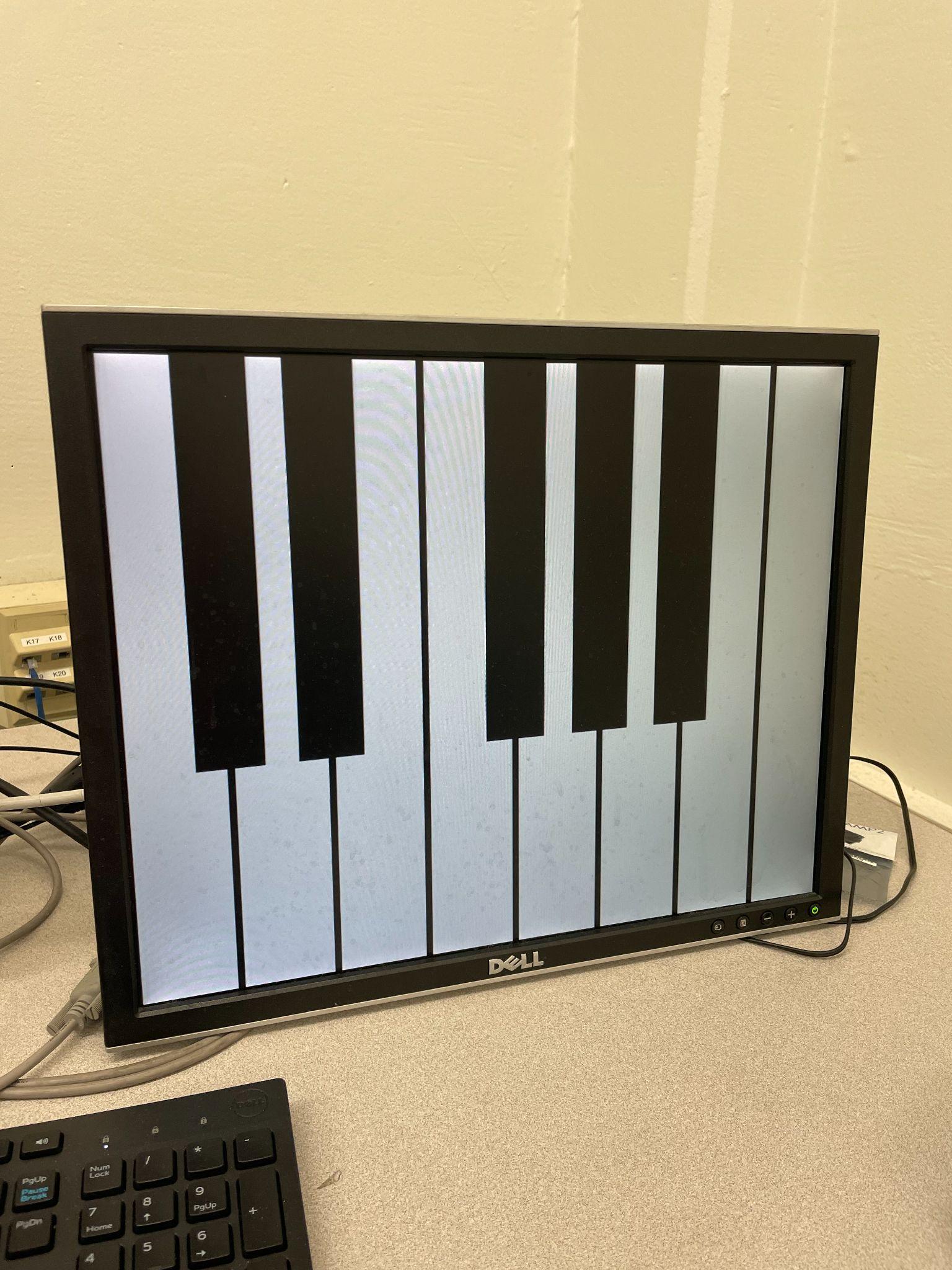
Specifics of the schematic will be detailed in the “Design Overview” section.

**Hardware Configuration**

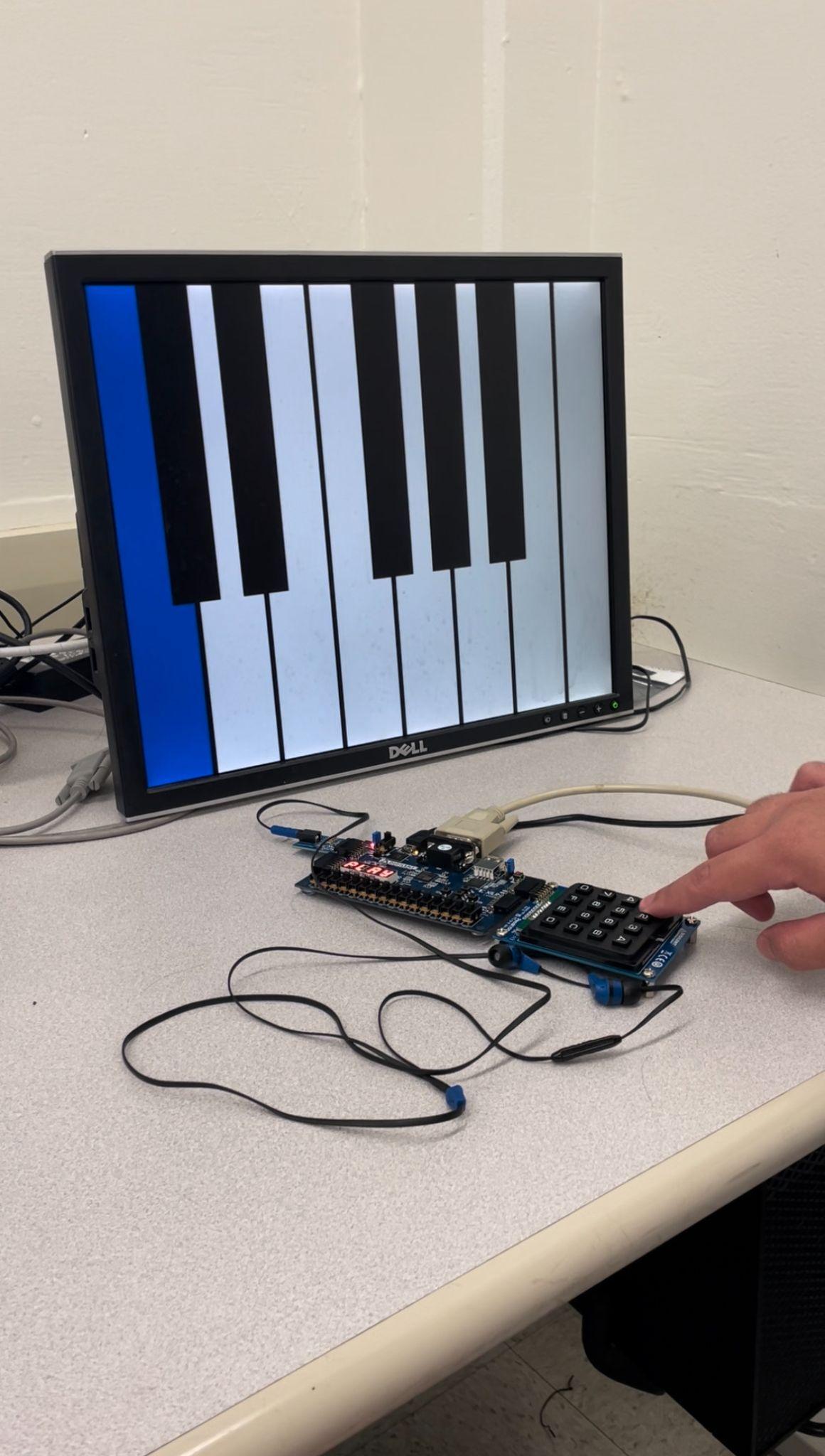
****

**Sample Output**

Default keyboard:

****

Key pressed:

****

**Code**

VGA (all Verilog code in this section is adapted from https://pumpingstationone.org/2013/04/nerp-fpgaok/)

vga\_top.v

This top-level module initializes all the submodules implemented on a hardware side: a clock divider to produce a pixel clock and seven-segment display clock (module U1), a seven-segment display controller for statically outputting the word “play” in the Basys3’s seven-segment display (module U2), and a VGA controller for outputting the dynamic piano display to a monitor.

clockdiv.v

This module produces the clock signals for refreshing the seven-segment display and VGA output. The module takes in as input a 1-bit wire clk (system clock) and 1-bit wire clr (asynchronous reset), and produces as output a 1-bit wire dclk (VGA clock) and 1-bit wire segclk (seven-segment display clock). Within the module, a 16-bit reg q serves as the counter register for both clocks. In an always block triggered on the positive edge of either clk or clr, q is incremented by 1 if the clr signal is not asserted. The reg resets naturally to 0 upon overflowing the 16 bits available to it, making setting an explicit upper threshold unnecessary. The most significant bit flips at a frequency of 381.47 Hz and is assigned as the value of the output segclk, while the second-to-least significant bit flips at a frequency of 25 mHz and is assigned as the value of the output dclk.

segdisplay.v

This module implements a finite state machine for outputting the word “play” from the Basys3’s seven-segment display. The module takes in as input a 1-bit wire segclk (clock for refreshing display) and 1-bit wire clr (asynchronous reset), and produces as output a 7-bit reg seg (encoding for current digit being refreshed) and 4-bit reg an (position of current digit being refreshed). In an always block triggered on the positive edge of either segclk or clr, the module will be in one of four states: left, midleft, midright, or right, representing the digit to refresh on the current clock cycle. A switch statement is used to select the current state and perform the necessary updates to the an and seg registers before transitioning to the next state. This fast refresh overcomes a hardware limitation of the Basys3, which is that by default, the four digits display the exact same output. Within the switch statement, while on each cycle the an register will logically illuminate only one of the four digits at a time, the refresh occurs so quickly that all four digits will appear to statically output distinct patterns, forming the word “play”.

vga640x480.v

This module produces a 640x480 VGA display of an octave of piano keys. The module takes in as input a 1-bit wire dclk (VGA refresh clock), 1-bit wire clr (asynchronous reset), and 16-bit wire data (button selected on keypad), and produces as output a 1-bit wire hsync (horizontal axis synchronization), a 1-bit wire vsync (vertical axis synchronization), a 4-bit reg red (r value for current pixel), 4-bit reg green (g value for current pixel), and 4-bit reg blue (b value for current pixel). Within the module, the 10-bit regs hc and vc represent the horizontal and vertical positions of the current pixel being refreshed, with the original located in the top-left corner and the positive x and y axes running rightward and downward respectively. The white keys are rendered first as 74x480 rectangles separated by 6-pixel gaps. The keys and gaps are rendered by simply checking the hc and vc values against a set of if conditions and setting the red, green, and blue outputs accordingly. Using similar logic, the black keys are then rendered atop the white keys as top-aligned 54x300 rectangles that evenly straddle two white keys at the locations that they would appear on a piano. In total, there are 8 white keys and 5 black keys. When a key is actively pressed, it will be rendered in blue instead of black or white. To accomplish this behavior, the lower 13 bits of the data input are each mapped to a key on the piano, and a value of 1 for any of the bits would represent a key currently being presse. Thus, the if block for rendering a key simply checks the value of the corresponding bit in data and renders the key blue if the bit contains a value of 1.

PmodKYPD and PmodAMP2

helloworld.c

After initializing the keypad, PmodAMP2, and GPIO channel to the VGA module, the program enters into an infinite loop to await the user’s keypad input. If only a single key is pressed (asserted using the KYPD\_getKeyPressed function), the user’s key input is converted from a char to an int and stored as the value k\_int. The value of the u16 state is then set to 1 << key\_int to conform to the format of the data input in the VGA module. This value is then written to the VGA module across the GPIO channel. Next, key\_int is used in a switch statement to determine the note frequency to output in the PmodAMP2, which is stored in the int freq. The predefined DemoSetFrequency function is then used to output a triangle wave of this frequency. A 0-initialized int counter is also incremented by 1. This value simulates the lingering of piano keys after short presses by ensuring that a frequency always persists for at least 11 loop iterations, even if the corresponding key is released immediately. If the key is released and the value of counter > 10 (indicating that the minimum linger time has been satisfied), the value of freq is set to 0, ending the audio output until a new key is pressed.

**Design Overview**

The components present in our block diagram are a MicroBlaze IP core, Microblaze debug module, Microblaze local memory, clocking wizard, processor system reset, AXI interconnect, PmodKYPD IP core, AXI Uartlite, PmodAMP2 IP core, AXI interrupt controller, concat, AXI GPIO, and a VGA module implemented at RTL. The MicroBlaze IP core represents the processor that will run our program. The MicroBlaze debug module is nominally for debugging purposes, although our group made no use of it. The Microblaze local memory provides the program with a small amount of RAM. The AXI interconnect is the interface that connects the set of slave modules to the FPGA. The PmodKYPD IP core takes input via GPIO and provides the key pressed as output to port jb, which determines which current note. The AXI Uartlite is the primary debug module, as it allows text to be output in the terminal via a serial connection. The PmodAMP2 IP core takes input corresponding to the current note via GPIO and outputs the sound wave for said note. This module also requires a timer interrupt, which is facilitated using connections to the concat module and AXI interrupt controller. The AXI GPIO module sets up a GPIO output channel to provide the VGA RTL module with the current note, allowing the module to update the display accordingly. The VGA RTL module renders an octave of piano keys and colors a key blue while its corresponding button is held down on the keypad.

**Problems and Solutions**

The first major problem faced was adapting a VGA demo made for the Nexys2 board for the Basys3. Even after following all the usual steps for adaptation, such as updating the constraint file to match the ports declared in the code, we were still unable to produce output on the VGA display. We ultimately solved this problem by thoroughly consulting the specifications for the Nexys2 and Basys3 and discovering that the boards had different system clock frequencies at 50 mHz and 100 mHz respectively. Adjusting the demo’s clock divider to account for the Basys3 clock’s higher frequency allowed the demo to function correctly.

The second major problem faced was making the entered key from the keypad available to both the PmodAMP2 and VGA modules. This proved challenging because the former was implemented in hardware, while the latter was implemented in software, making it unclear if we should implement the connection in hardware or software. In the end, software was the solution. For the VGA module, the XGPIO module was used to establish an explicitly output channel, which took in values across this channel as the 16-bit input keyout (name is a remnant of a previous design in which this value was an output). For the PmodAMP2 module, the software-side logic for sound generation was simply integrated into the same source file as the logic for reading keypad input, implicitly making the keypad values available for use in sound generation.

Our time for this project was limited as this was not the first idea we had explored. Our original idea was to create a wireless control module for the iRobot Create 2 using two Basys3 boards. Although we were able to get working demos of the Pmod Joystick and Pmod Bluetooth modules, we hit a roadblock when it came to creating a serial communication interface between the Basys3 and Create 2. At first we attempted to use the PMOD RS232 module that had been used in previous classes, but after some research we found that there were no working examples of this module on the Basys3. Eventually we decided to abandon this idea as we did not believe we had enough time to write new Verilog code to port the RS232 to the Basys3.

**Contributions**

Alberto consulted datasheets to validate the block diagram and wrote logic for producing audio output. Evan created the piano display on the VGA and wrote part of the logic for processing keypad input. Brandon set up the GPIO connection to the VGA and wrote part of the logic for processing keypad input. All three group members worked together to debug and test the project after it was completed.