ECE 110/120 Honors Lab

Spring 2023

Final Lab Report

A Breadboard-based VGA Driver with Two User Inputs

P. Chen (pxchen2) - ECE 120, Z. Yan (zexuany2) - ECE 110, H. Zhang (hanwenz7) - ECE 120

1 Introduction

The goal of this project is to make a display card out of TTL chips with a keypad and a mouse as user input. A cursor will be displayed on the monitor showing the physical position of the mouse, and it will be able to draw colored lines on the screen when the mouse is dragged across the screen with its left button pressed. A different geometrical pattern (e.g. square, triangle, circuit, etc.) with different sizes will be generated on the monitor around the cursor when specific keys on the keypad are pressed. Also, other keys will be reserved to choose the color of the pattern and clear the screen. and the display card will communicate with the monitor through a VGA port. We split up the work as Phillip Chen mostly worked on the mouse and the program, Zexuan Yan mostly worked on the physical display circuit, and Hanwen Zhang worked mostly on the keypad and the program. We did finish the digital circuit mentioned in Figure 1, and it is able to output the input signals from the mouse and the keypad. The target outcome is a 200 pixels * 150 pixels resolution with 64-bit color and the ability to achieve double buffering. However, there are some discrepancies between the actual output and the desired output as shown in the implementation section.

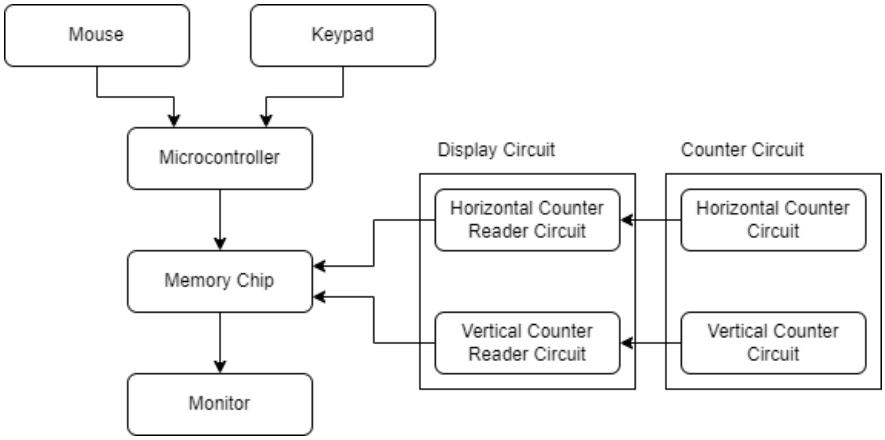

Figure 1: Flow Chart of the Project

2 Background Research

We did find quite some people making their own display circuits on breadboards. For example, we got our idea from Ben Eater, who made a series of videos on making a breadboard graphic card. Also, there is another YouTuber called jdh that also made a graphic card on a breadboard, but he used a much older display adaptor. Our project was planned to be different from what Ben Eater did because we want to add two user input, which means we need to read/write RAM chips and use a microcontroller to write the data into the RAM chip. In order to integrate the user input function into the existing project, there are three major functionalities: collecting data from the mouse and the keypad, loading the processed data into a memory chip, and displaying the corresponding data onto the screen. The data from the keypad will be gathered through a matrix scan to the Arduino which can be read directly. The mouse will communicate with the Arduino through a USB breakout port, and there are open-source libraries that can help us interpret the data collected from the mouse. After the Arduino read the data collected from the mouse and keypad, it will interpret and convert the data to binary and store the data in the memory chip. Finally, a similar approach to what Ben Eater did can be implemented to display the content of the memory chip to the monitor. The background information of each part mentioned above will be discussed in greater detail in the following sections, and the detailed implementation will be talked about in the Display section and Implementation of the Input section.

2.1 Keypad

The size of the keypad we are using for this project will be 4*4, which will provide enough buttons without overcomplicating this project. Like every keyboard we are using today, the keypad will read the

users' operation through a circuit matrix which is formed by both row pins and column pins as shown in Figure 1. Whenever the user presses the button, it connects one of the input pins and the output pins. The input pin and output pin can be the row pin or column pin depending on the practice. To actually detect which button the user is pressing, the input pin will practice a shift in voltage it input continuously one by one during the usage. For example, all output pins and all but one input pin are sending high voltage, leaving the first-row pin sending low voltage, whichever the user presses any button on the first row, the input pin that is connected due to the press of the button will detect low voltage which represents a "0". To make every row functional, the trail must change from time to time, after setting the first-row pin to send low voltage, the system then shifts the second-row pin to send low voltage in case the user is pressing the buttons on the second row. From this process, the user's operation will be transformed into an electric signal and sent to Arduino for further processing.

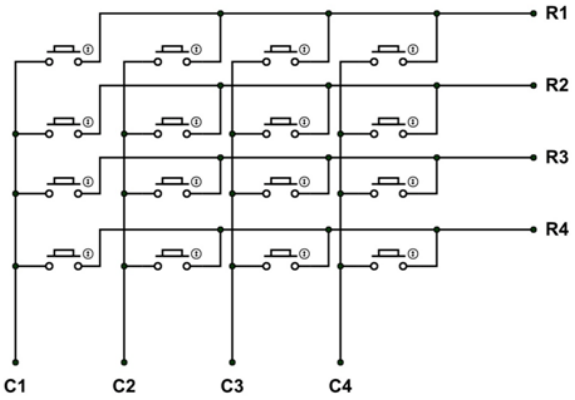

Figure 2: Keypad Schematics

2.2 Mouse

We will be using a standard USB mouse for this honors project, which consists of left and right-clicks buttons as well as a scroll wheel. We will only use the left mouse button (we might include the right-click button and the scroll wheel if time allows). When enough force is applied to the left button, its switch is triggered, closing the internal circuit and allowing current to flow. Upon receiving this current, the mouse's sensor chip will output a specific analog signal through its USB Port corresponding to the input done by the user on the mouse (can and will be treated as digital signals, please see the next paragraph for detailed information). We will plug the USB port of our mouse into a standard USB Connector shown in Figure 3.

This connector will have four pins that can be connected to other circuits. The outer pins are connected to Vcc (+5V) and Ground (0V). Our mouse has two sets of protocols: USB and PS/2. We'll use the PS/2 protocol since it's much simpler (and it's still implemented in most modern mice). For this protocol, the D- pin carries the data signal and the D+ pin carries the clock signal. During each clock cycle, the mouse sends movement and button information as a 3-byte package to the host. The second and third bytes carry the movement of the mouse in the X and Y directions. The first bit carries overflow and signs bits of the movements as well as bits that indicate which mouse buttons are pressed. This is visualized in Figure 3 below. We will treat the internal mouse circuit as a black box and will focus on analyzing the output current signals for further use.

2.3 Microcontroller

This project utilizes an Arduino Mega 2560 Rev3 as the microcontroller mostly due to its large number of I/O pins. The microcontroller will process the input from both the mouse and the keypad and then store the processed data in the display buffers. Because of the 200 pixels * 150 pixels resolution, double buffering on a 16-bit address and 6-bit data system would require $16 + 2 * 6 = 28$ output pins, the mouse would require two input pins, and the keypad would also require 8 pins in total. This gives a minimum number of 36 pins

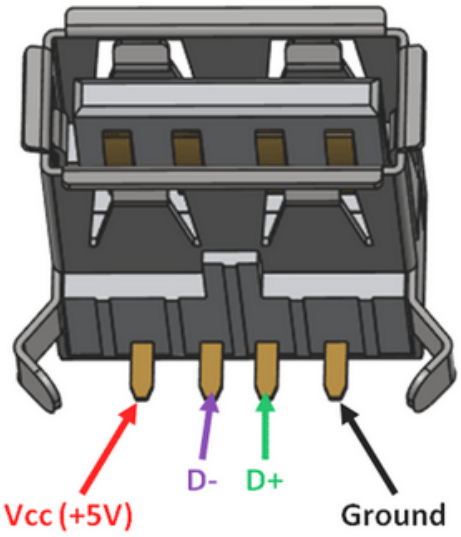

Figure 3: USB Port Pinout

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1	Y overflow	X overflow	Y sign bit	X sign bit	Always 1	Middle Btn	Right Btn	Left Btn
Byte 2	X Movement							
Byte 3	Y Movement							

Figure 4: USB Data Package

excluding many other control signals, so, a microcontroller with a large number of I/O pins is required in this project. The main function of the microcontroller is to interpret the input signal from the mouse and the keyboard and convert that into proper binary data to load into the memory chip. For example, when the clear button is pressed, the microcontroller will send a series of signals to the memory chip that loops through all memory locations and set the input data signal to be all zeros. When the mouse moves across the screen, the microcontroller will record every coordinate of the mouse and write data into the corresponding memory locations through its pins. The microcontroller will also have four outputs to control the mux switches and the read/write signal for the memory chip.

2.4 Display circuit

2.4.1 Plan for implementation

This project aimed to create a signal that the monitor will recognize as 800 pixels * 600 pixels with 60 Hz refresh rate mostly because this is one of the lowest resolutions that a VGA signal supports. In order to generate a proper VGA signal, both the horizontal and vertical signal timing has to match the specific timing of the visible area, front porch, sync pulse, back porch, and the whole line. For our desired resolution, the arrangement of all parts mentioned above is shown in Figure 6 below.

As shown in Figure 5, the monitor is only actively displaying pixels during the "Active Pixels" area while it does not display anything during the vertical and horizontal front/back porch area. Also, horizontal and vertical signals should also be sent to the monitor during horizontal and vertical sync areas to ensure the proper synchronization of the signals. The magnitude of each area depends on the resolution and refresh rate of the signal. In our case, we chose an 800*600 pixel 60 Hz refresh rate with a vertical refresh rate of 37.878788 kHz. The required timing for all signals is shown in Figure 6 below.

In order to achieve this signal timing, the base clock speed will be 40 MHz. We reduced the base clock to 10 MHz so that the circuit will be much easier to implement (i.e. we don't have to check a number as large as 1056 in binary because that would require 11 bits to represent). However, this also changes the resolution in the horizontal lines. In order to divide the clock by 4 times and keep the timing for all scanline parts the

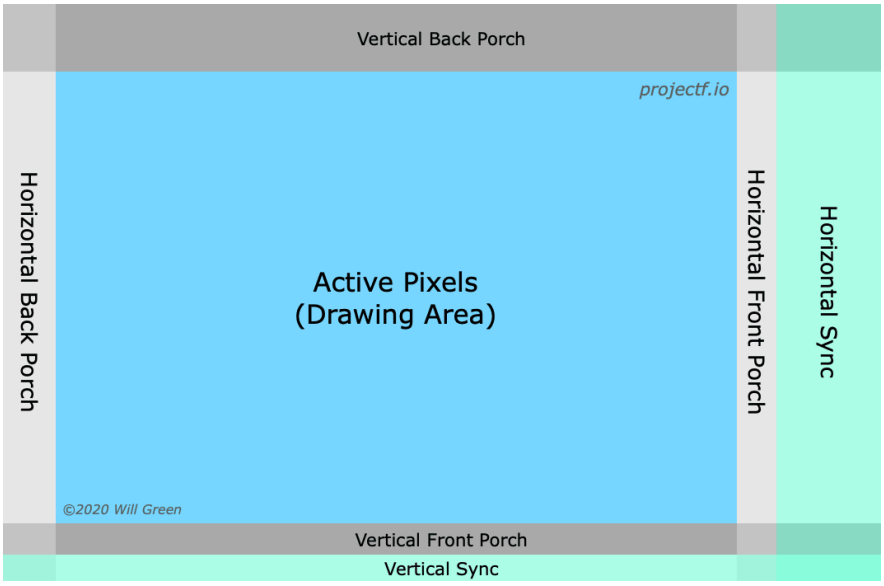

Figure 5: VGA Signal Area

Horizontal timing (line)

Polarity of horizontal sync pulse is positive.

Scanline part	Pixels	Time [µs]
Visible area	800	20
Front porch	40	1
Sync pulse	128	3.2
Back porch	88	2.2
Whole line	1056	26.4

Vertical timing (frame)

Polarity of vertical sync pulse is positive.

Frame part	Lines	Time [ms]
Visible area	600	15.84
Front porch	1	0.0264
Sync pulse	4	0.1056
Back porch	23	0.6072
Whole frame	628	16.5792

Figure 6: VGA Signal Timing

same, we need to divide the number of pixels by 4 too, and this would keep the vertical timing unchanged. This modification will give us a rather weird resolution of 200 pixels * 600 pixels. The vertical resolution of the signal will be further divided by 4, resulting in a final resolution of 200 pixels * 150 pixels. The modified horizontal timing is shown in Table 1 below.

Scanline Part	Pixels	Total Pixels	In Binary
Visible Area	200	200	011001000
Front Porch	10	210	011010010
Sync Pulse	32	242	011110010
Back Porch	22	264	100001000

Table 1: Modified VGA Horizontal Timing

2.5 Counter Circuit

2.5.1 Clock

We brought an ACH-10.000MHZ-EK as the clock for our circuit. We simply connected the input of this clock to a 5V source, and it can produce a 5V signal at 10MHz from its output pin. The max load of this oscillator is 10 TTL chips as specified in the datasheet. However, this oscillator does not seem to have a buffer. We did not realize the importance of a buffer when we first started, and we later learned that a buffer will make the signal much less influenced by the amount of load we put on the oscillator. Because we are only driving 3 TTL chips directly from this oscillator, our circuit ended up working fine. If anyone is planning to rebuild and improve upon this project, an oscillator with a buffer is strongly encouraged although our design worked.

2.5.2 Binary Counters

For the counter circuit, we used SN74LS161AN (4-bit output binary counter). The amount of bits we need to check depends on the number of pixels in binary representation in horizontal and vertical counters. We need to check 9 bits for the horizontal circuit because the whole line consists of 264 pixels, which needs 9 bits to represent. Analogously, the vertical counter goes up to 628 pixels, which needs 10 bits to represent as shown in Figure 8. Because the number of bits we need to check in both horizontal and vertical counters is between 8 and 12, we need 3 SN74LS161AN binary counters to count up to the number of bits needed. All inputs of the counters should be left empty because we simply want the counters to count up normally. The clock of the counters for the horizontal part should be connected to the oscillator directly, and the clock of the vertical counters should NOT be connected to the oscillator but rather to the reset signal to the horizontal counter. This is because we only want to count vertically after we finish counting a whole line horizontally.

Scanline Part	Pixels	Total Pixels	In Binary
Visible Area	600	600	1001011000
Front Porch	1	601	1001011001
Sync Pulse	4	605	1001011101
Back Porch	23	628	1001110100

Table 2: Modified VGA Vertical Timing

2.5.3 Reader Circuit for Counters

The horizontal/vertical counter reader circuit will read the output of the respected counter circuit and determine when to switch lines by sending a reset signal to the binary counters when a line is finished. We used SN74LS30 (8-input NAND gate) to check the stage of the scanline parts in the horizontal counter and frame parts in vertical counters. As we observed in the horizontal timing table (Figure 7), the binary pattern for the first three scanline parts (visible area, front porch, and sync pulse) starts at zero. That means we can ignore the most significant bit when checking for those three stages, which gives us eight bits to check. Because we will only count up to 264, we don't have to worry about the same bit pattern getting recognized again. For the last stage, we can ignore the least significant bit because we want to reset the horizontal counter right after 264 pixels, so it will never count to 265 if the circuit is done properly. When checking to see if we hit a specific part in the scanline, we connect the output of the counter bits to an inverter (we used SN74LS04) and the output of the inverter to the 8-input NAND gates so that, when the specific bit pattern is met, the NAND gate goes from high to low. The NAND gate that checks the last bit pattern (264 pixels) is also connected to the reset pin of the horizontal counters so that every time when we finish counting a line, the counters reset to zero. This output is also connected to the clock input of the vertical counters as every time a line is finished, we want to increment the vertical position and go to the next line.

Analogously, for the vertical counters, we can ignore bits 7 and 8 as indicated in Figure 8 because they are all zeros in all bit patterns that we are interested in. For the same reason, as it is in the horizontal circuit, we don't have to worry about the same bit pattern getting recognized more than once because we will reset the counter at 628 lines every time. The same technique is used to detect the specific bit pattern - using 8-input NAND Gates. When the bit pattern for 628 is detected, we know that the whole screen has

been scanned through. After the counter finished this line, we want to reset both the horizontal and vertical counter to start a new frame, so the output of the NAND Gate detecting 628 lines will be connected to the reset signal of the vertical counter.

2.5.4 Flip-flops

Now we have the circuit detect all parts of the VGA signal as shown in Figure 5, we need to switch the sync signal on and off at the right time. For the horizontal sync pulse, the sync pulse should be set to on after the front porch is finished (after pixel 210) and off after the sync pulse is finished (after pixel 242). Similarly, the vertical sync pulse should be set to on after line 601 and set to off after line 605. The easiest approach is to use flip-flops for this. We can just connect the set signal of the flip-flop with the NAND gate detecting pixel 210 in the horizontal reader circuit and the reset signal of the flip-flop with the NAND gate detecting pixel 242 to generate the horizontal sync pulse. Analogously, for the vertical sync pulse, we connected the NAND gate detecting line 601 with the set signal for another flip-flop and the NAND gate detecting line 605 with the reset signal of the same flip-flop. The flip-flop was implemented through two SN74LS00 Quad NAND Gates chips. We also used a similar design to generate the signal for the visible area, connecting the NAND gate detecting the whole line to the set signal and the NAND gate detecting the visible area to the reset signal. This was meant to be used along with the memory chip, but we did not finish that part. The schematics for all the parts mentioned above are shown in Figure 9 and Figure 10 below (the black cross on the wire in Figure 10 indicates that the wire should not be connected there - I probably shouldn't draw schematics with pens).

2.5.5 Wiring and Wire Management

Because of the number of TTL chips used and the 8 different bit patterns we are trying to detect, there will inevitably be a lot of wires, and this made debugging very inconvenient. To make the wiring process easier, here are a few rules that we followed:

- Color code your wire: we used purple wires for clock-related signals, red and orange for power and reset signals, yellow for non-inverted counter output, and grey for inverted counter output
- Cut your wires into proper length so that they make right angles and stay at the same height as the TTL chips. This will make the circuit look much neater.
- Reuse any signals that you can. For example, when detecting all bit patterns in the horizontal counter circuit, all the least significant bits are zero. In this case, one should not pull extra wires from the inverter to connect to the NAND gates but only pull one signal from the inverter and connect that same input to all other NAND gates using jumper wires.

2.6 Display Circuit with Memory Chip

Ben Eater used a parallel EEPROM to store data and read from it by inputting the X/Y coordinate of the pixel display on the monitor, which can be easily obtained from the counter circuits to the address pins of the EEPROM. We decided that we will be using a 200*150 pixel resolution because both the x and y coordinates of the pixel will be stored in 8-bit. When implementing, we can simply left-shift the counter on the y coordinate (i.e. the vertical counter which had ten bits) twice to obtain an 8-bit representation. The counter on the x coordinate (i.e. the horizontal counter which has nine bits) will only provide eight bits to the input because the visible area of the horizontal pixels only goes up to 200 pixels, which is enough to be represented with eight bits. We will be using 512K*8 memory chips because it has 18 address pins (it was hard to find memory chips with exact sizes, and more memory comes with little cost but more room for improvement in the future). Then, both inputs from the horizontal and vertical counter will be loaded into two 512K*8 memory chips for double buffering, which we will discuss in detail in the next paragraph.

The reason behind the double buffering (loading image data into buffer 1 when the data in the buffer is being displayed, and displaying the data in buffer 1 when loading image data into buffer 0) is that the Arduino is not fast enough to write data into the memory chip when displaying. We didn't find any dual-ported memory chips that are above 512k*8 which could be written and read at the same time, so two SRAMs with a capacity of 512K*8 were selected for this project. Doing double buffering will increase the

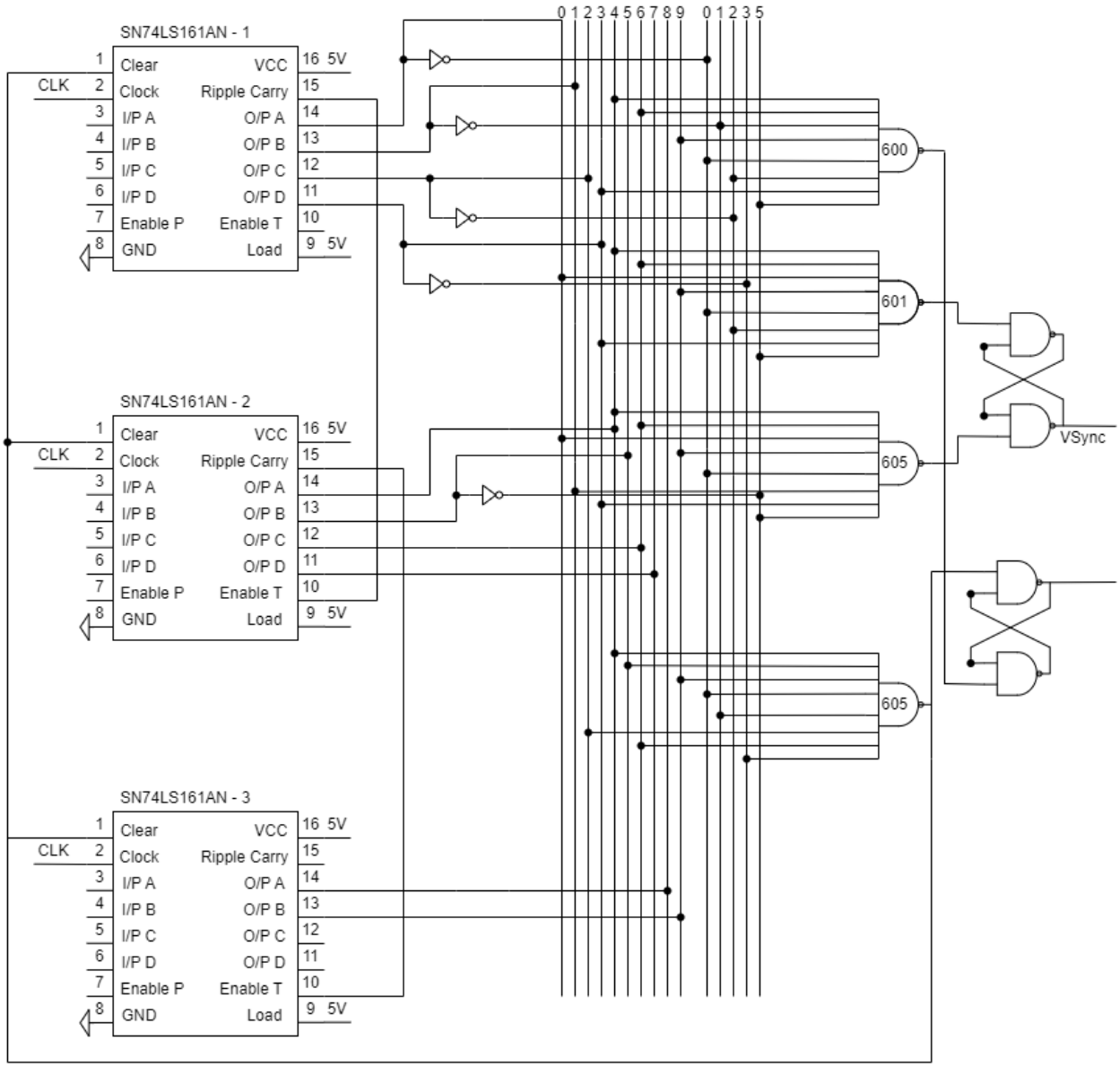

Figure 7: Vertical Counter Schematics

difficulty but would also give up a much smoother running display. However, due to time constraints, double buffering was not implemented and remains one of the future goals.

The memory chip can store 8-bit data for each memory location it has, and Ben Eater used a simple but very effective voltage dividing circuit to alter the input voltage into the VGA driver as shown in Figure 9.

Using the following configuration, we can get four different voltage outputs at where the arrow is by turning on/off the 5V voltage source at each end of the wire (i.e. Input 1 and Input 0). By setting the top and bottom input into different states (00, 01, 10, 11), we can obtain four different voltage outputs (0V, 0.23V 0.47V 0.7V), which corresponds to four different color hues on the monitor. The memory chip and voltage divider circuit will be two of the major changes that set this project apart from the previous one.

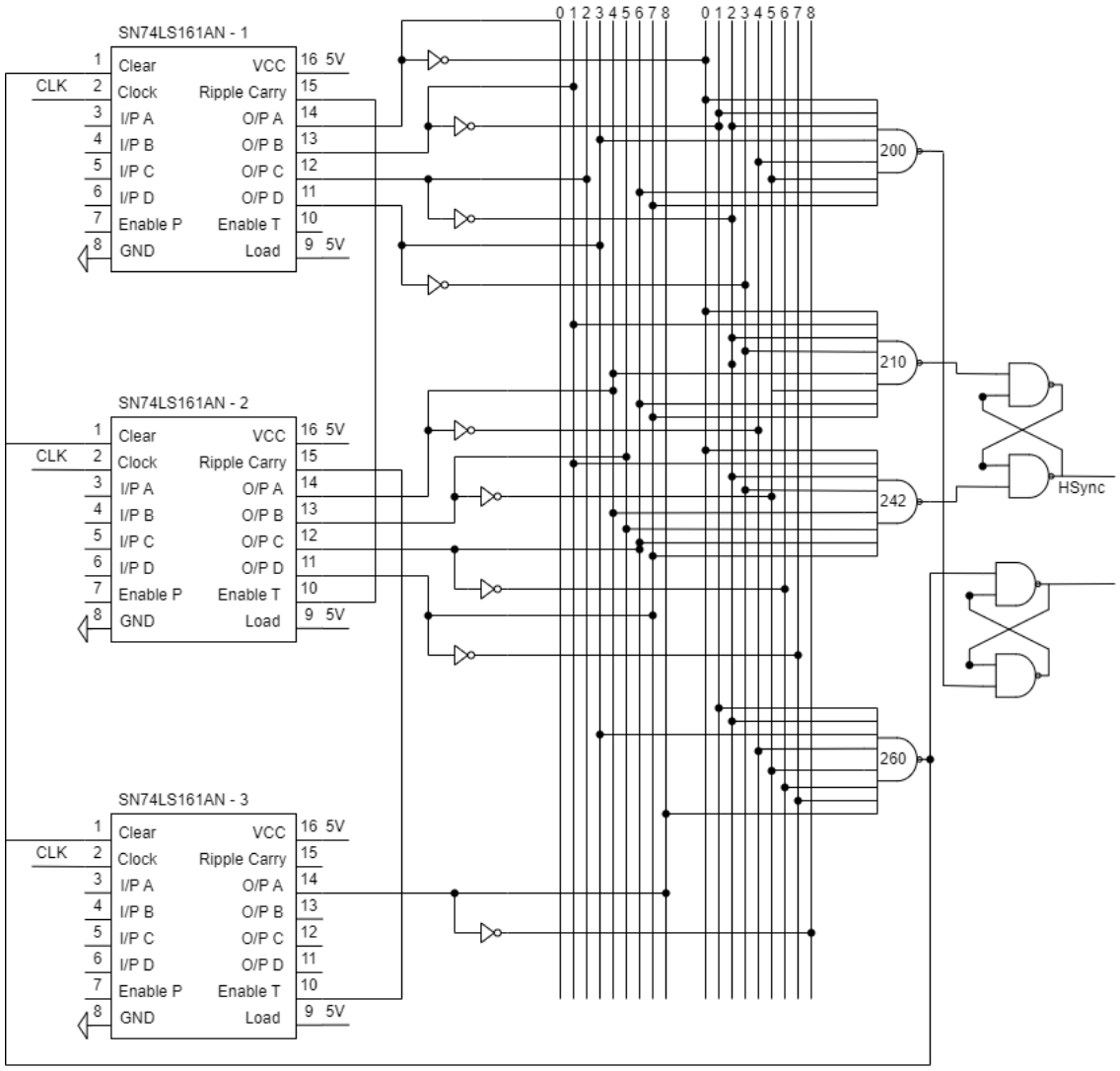

Figure 8: Horizontal Counter Schematics

3 Display

3.1 VGA Communication

We used a VGA port to communicate between our circuit and the monitor, and the output of the circuit should be connected to a VGA breakout port, and the breakout port itself will be connected to a VGA cable that is connected to the monitor. The color output is directly connected to the VGA breakout port. The specific pinout of the VGA cable and how the color stripes are displayed will be discussed in the next section.

3.2 Pinout and Single Color display

The specific pinout of the VGA cable is shown in Figure 10 below.

Out of 15 pins on a VGA cable, there are a few pins that we don't have to worry about. Pin 4 (RES), pin 9 (KEY or 5V), and pin 11 (ID0) can be left empty because pin 4 is reserved, and pin 9 and 11 are optional. Also, pin 12 (SDA) and pin 15 (SCL) can be also left empty because we are not using I2C communication here.

Also, there are 5 ground pins (pin 5 (SGND), pin 6 (RGND), pin 7 (GGND), pin 8 (BGND), and pin 10 (GND)) that can simply be connected to ground on the breadboard. There are only 5 data pins left that we need to worry about. First, pin 13 (HSYNC) should be connected to the inverted output of the flip-flop

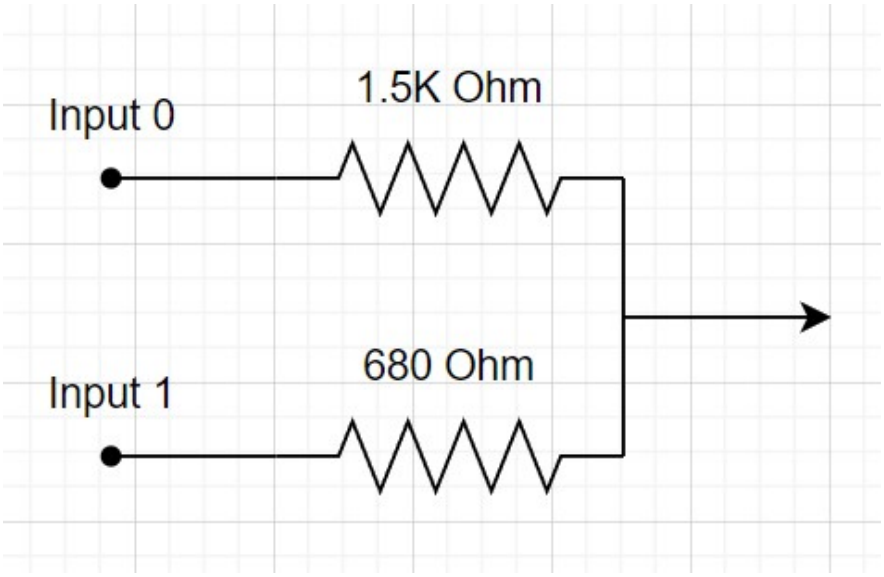

Figure 9: Voltage Divider Circuit for Display Circuit

Pin	Name	Dir	Description
1	RED	→	Red Video (75 ohm, 0.7 V p-p)
2	GREEN	→	Green Video (75 ohm, 0.7 V p-p)
3	BLUE	→	Blue Video (75 ohm, 0.7 V p-p)
4	RES		RESERVED
5	SGND	—	Ground
6	RGND	—	Red Ground
7	GGND	—	Green Ground
8	BGND	—	Blue Ground
9	KEY	-	Key (No pin) / Optional +5V output from graphics card
10	GND	—	Sync Ground
11	ID0	←	Monitor ID Bit 0 (optional)
12	SDA	↔	I2C bidirectional data line
13	HSYNC or CSYNC	→	Horizontal Sync (or Composite Sync)
14	VSYNC	→	Vertical Sync which works also as data clock
15	SCL	↔	I2C data clock in DDC2, Monitor ID3 in DDC1. A slave can pull SCL low to make the master wait.

Figure 10: VGA Pinout

(we want the signal to be low in the sync pulse) that stores the horizontal sync signal, and pin 14 (VSYNC) should be connected to the inverted output of the flip-flop (we want the signal also to be low in the sync pulse) that stores the vertical sync signal. If both sync pulses are generated properly, after connecting the circuit with the monitor, the monitor should indicate that the input signal from the VGA port is active. And

if you check the mode of the monitor now, you should see the current input mode being 800*600 60Hz.

Unlike the sync pulses that can be directly connected to the VGA cable, the RGB color signal (pins 1,2,3) works a little differently. As shown in Figure 12, the three color signals take an input voltage of up to 0.7 V and have an internal resistance of 75 ohms. The color displayed will depend on the amount of voltage input into red, green, and blue pins. The higher the voltage is in a certain color pin, the darker that color is. We could set up a voltage divider using two to reduce the voltage to below 0.7 V. Note that the voltage should never be higher than 0.7 V with the voltage divider present or it may damage the monitor. For example, the output voltage of the TTL chips is around 3.3V in our circuit, and we used a 680-ohm resistor to connect to the blue pin of the VGA cable, which created a 0.328 V input into the blue pin, resulting in the color shown in Figure 13.

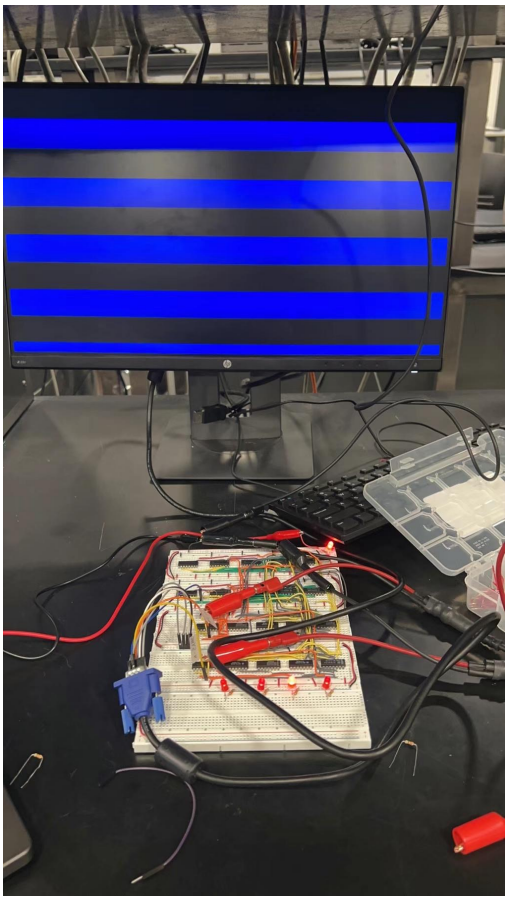

Figure 11: VGA Display with Blue Stripes

Although displaying color stripes was not the original intention of this project, due to time constraints, we were not able to finish the input circuit, and displaying color stripes is a simple but efficient way to show the function of the VGA driver. How exactly we make the monitor display color strips will be discussed in the next section.

3.3 Color Stripe and Multi-color Display

The simplest way to display color stripes is to connect one of the RGB inputs to a specific output bit in the vertical counter. For example, if the blue pin is connected to output bit 0 (indicated in Figure 8), the monitor will display a blue stripe for every other line. This is because the first place bit flips on and off every other line, so the output of blue light will be set to high on every other line. Similarly, if we want the color to flip on and off every sixteen lines, we can connect the blue input to output bit 4 in the vertical counter. Analogously, if we want to display vertical stripes instead of horizontal stripes, we just need to connect the color input to the output bits of the horizontal counter instead of the vertical counter. Note that the darkness

of the color will depend on the resistance of the resistor that you used to set up the voltage divider circuits. Also, this approach will make the monitor display color pixels during the whole line. This did not become a problem in our implementation. However, if we were working on a CRT monitor instead of a VGA monitor, there is a very high chance that this approach will damage the CRT monitor.

For displaying multiple colors and mixing red, green, and blue to get other colors, we just need to set two colors or more to be on in the same line. For example, if we connect the input for red to output bit 0 and the input for blue to output bit 1 in the vertical counter, the red light will be on for every other horizontal line, and the blue light will be on and off for every two lines. By doing this, we can get a mix of red and blue colors, which gives us purple. Again, the degree of purple can be adjusted based on the resistance of the resistors. Mixing more than two colors follows the same logic. For example, we made the monitor display different color stripes with a combination of RGB inputs in Figure 14.

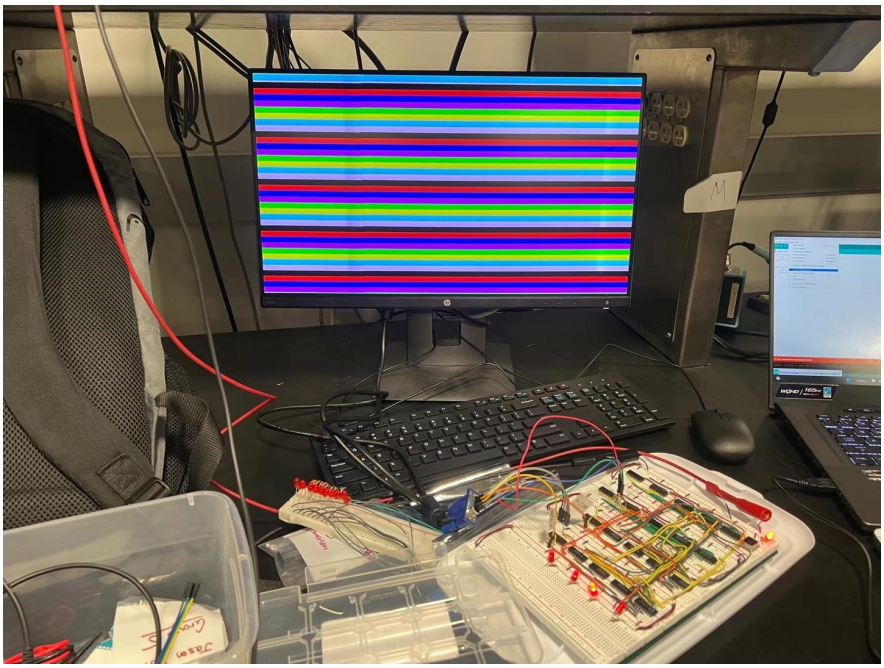

Figure 12: VGA Display with Multi-color Stripes

At the end of the Fall 2022 semester, the display circuit was hardwired and did not have a user input function. The goal for Spring 2023 semester was to finish the user input and enable the monitor to display what the memory chip stores. All implementations with input and memory chips will be discussed in the following section.

4 Implementations of the Input

4.1 Timeline for the Second Semester

- Week 1: Analyze the input signals produced by the mouse and keypad, and install the mux switch and memory.
- Week 2: Spring Break
- Week 3: Determine how to interact with the memory chips to display input data on the monitor, and install the VGA port.
- Week 4: Program the Arduino such that it can convert these signals into a binary representation that could be loaded into the memory chip. Start connecting the output of the mux to the Memory chip.
- Week 5: Start working on writing the input data of the mouse and keypad to the memory chip. Finish connecting the output of the mux to the Memory chip.

- Week 6: Finish the program that writes the input data into the memory chip. Finish connecting the other input of the mux to the Arduino.
- Week 7: Finish testing the program and the analog mux for the circuit. Then start combining the two parts of the project.
- Week 8: Finish combining the project. Troubleshoot everything then start drawing the schematics and working on the report.
- Week 9: Finalize the project and do the demo.

4.2 Mouse to Arduino

Our initial plan to communicate with the mouse was to use oscilloscopes to analyze the outputted analog signals of the USB Port by setting up evaluation parameters and criteria such as frequency and amplitude to differentiate between the outputted signals. Then, by connecting the pins of the USB Collector to the input ports of the Arduino, it receives these signals and will be ready to process them. Next, We will convert the analog signals to corresponding discrete digital signals by programming these methods on Arduino IDE. However, we later discovered that we can skip these steps because we have done additional research on how the PS/2 protocol works and found a library that already established the communication methods with the mouse. The Arduino will feed these digital signals into our breadboard VGA circuit for further execution.

We have searched for tutorials on how to process the data signals sent by the mouse and what clock signal we should provide to the mouse using an Arduino. The mouse's communication with the Arduino is fairly simple. The Arduino provides 5V to the mouse, and the D+ and D- pins of the mouse are connected to two digital pins as shown in Figure 6 below.

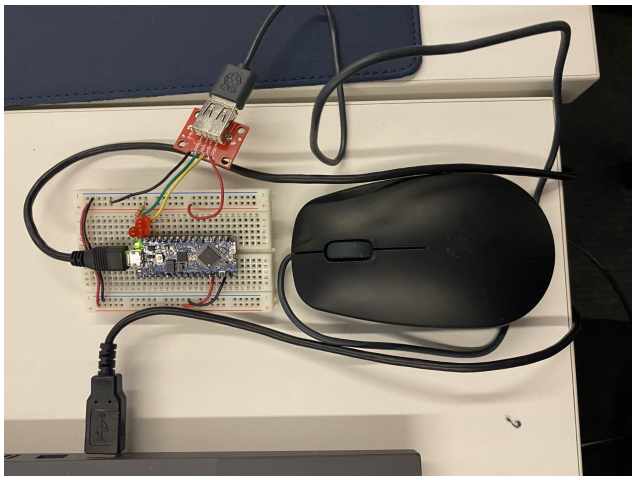

Figure 13: Mouse Connection with Arduino

We found an Arduino Library created by Github user "getis" that contains code for reading data and providing clock signals. In the code shown in Figure 7, we called the functions and methods provided in the library.

To check whether the code is working, we wrote print statements to print the mouse's activity every 100ms. The printed values include movements in the X and Y directions, the scroll wheel, and whether any button is pressed. The mouse is outputting correct movement data.

We then coded the communication between the mouse and breadboard circuit through the Arduino. The following Figure is a code snippet from our software that establishes the communication:

Everything is coded under an if statement that is triggered every 100ms and is located at the beginning of the loop() method. Thus, the program receives the mouse's input every 100ms. This program accomplishes a few tasks: it first erases the mouse's trace by setting the color data of previous coordinates to zeros, which is black. Then, it fetches the change in coordinates of the mouse and then calculates the new coordinates, constraining them within the monitor's size if necessary. Next, the program displays a five-pixel crosshair centered at the coordinates on the monitor.

```
26  void loop()
27  {
28    if (millis() - last_run > 100) {
29      last_run = millis();
30      mouse.get_data();
31      Serial.print(mouse.device_id()); // device id
32      Serial.print(":");
33      Serial.print(mouse.status()); // Status Byte
34      Serial.print(":");
35      Serial.print(mouse.x_movement()); // X Movement Data
36      Serial.print(",");
37      Serial.print(mouse.y_movement()); // Y Movement Data
38      Serial.print(",");
39      Serial.print(mouse.z_movement()); // Z Movement Data - scroll wheel
40      Serial.print(",");
41      Serial.print(mouse.button(0));
42      if (mouse.button(0)) {digitalWrite(9, HIGH);}
43      else {digitalWrite(9, LOW);}
44      Serial.print(",");
45      Serial.print(mouse.button(1));
46      Serial.print(",");
47      Serial.print(mouse.button(2));
48      if (mouse.button(2)) {digitalWrite(10, HIGH);}
49      else {digitalWrite(10, LOW);}
50      Serial.print(",");
51      Serial.print(mouse.clicked(0));
52      Serial.print(",");
53      Serial.print(mouse.clicked(1));
54      Serial.print(",");
55      Serial.print(mouse.clicked(2));
56      Serial.println();
57    }
58  }
```

Figure 14: Mouse Arduino Test Code

During the debugging phase, it has come to our attention that the basic mouse movements cannot be displayed correctly on the monitor. Thus, we abandoned the functionality of drawing colored lines on the monitor.

4.3 Keypad to Arduino

After the hardware arrives, we will test the jitter frequencies of the buttons for the further debouncing process. For every pin that is connected to Arduino, we will series connect a pull-up resistor to justify the input and output voltage. It will take 8 pins to deal with all operations on the keypad. Among these 8 pins, four-row pins will be used as input pins while the others used as output pins. Through programming, four output pins will continuously shift voltage from column to column so that we can detect the button pressed. We may introduce some kinds of hardware debouncing techniques to make sure Arduino receives a clear signal. With the purified signal input, through programming in Arduino IDE, the digital signal input will be sent to the VGA circuit for further processing.

We then connected the keypad with the Arduino and made it fully functional to input chars to the screen. Though there are still problems like the loose connection between the keypad and breadboard, we can correct it with physical upholders, with will be fixed at the very end of the process.

Next, we added more functions by writing serval functions for different buttons. By now, we complete the function of writing the color on the screen, clearing the screen with a single color, and some basic interaction between the mouse and keypad.

Through programming, we are now able to collect input data from both the mouse and keypad, including the button signal and the mouse's movement information. By setting color data to a 6 bits array, we can compile the color we want to input through the combination of two bits of red, two bits of green, and two bits of blue. With several number buttons (1,3,4,6,7,9), users can change the color they want to write on the

```
void loop()
{
  if (millis() - last_run > 100) { //update mouse movement based on sampling period
    last_run = millis();
    mouse.get_data();
    changeColor(prevX,prevY,colorZeros); // set color of previous coordinates to black
    if (prevX > 0) { changeColor(prevX - 1,prevY,colorZeros); }
    if (totMovementCol < 256) { changeColor(prevX + 1,prevY,colorZeros); }
    if (totMovementRow > 0) { changeColor(prevX,prevY - 1,colorZeros); }
    if (totMovementRow < 157) { changeColor(prevX,prevY + 1,colorZeros); }
    deltaX = mouse.x_movement() / scalingFactor; // slow down mouse movement speed
    deltaY = mouse.y_movement() / scalingFactor;
    newX = totMovementCol + deltaX;
    newY = totMovementRow + deltaY;
    if (newX <= 0) { newX = 0; }
    if (newY <= 0) { newY = 0; }
    if (newX >= 255) { newX = 255; }
    if (newY >= 156) { newY = 156; }
    totMovementCol = newX;
    totMovementRow = newY;
    //display cursor as a 5-pixel crosshair
    changeColor(totMovementCol,totMovementRow,colorInput);
    if (totMovementCol > 0) { changeColor(totMovementCol - 1,totMovementRow,colorInput); }
    if (totMovementCol < 256) { changeColor(totMovementCol + 1,totMovementRow,colorInput); }
    if (totMovementRow > 0) { changeColor(totMovementCol,totMovementRow - 1,colorInput); }
    if (totMovementRow < 157) { changeColor(totMovementCol,totMovementRow + 1,colorInput); }
    prevX = totMovementCol; //set current coordinates to previous coordinates after modifying the screen
    prevY = totMovementRow;
  }
```

Figure 15: Mouse Arduino Code

screen. We also write some color blocks on the screen through some easy for-loops. Through a little function of changeColor (in Figure 16), we use three inputs, column coordinate, row coordinate, and color data to write any single pixel on the screen. To better match the inputs from the mouse's movement coordinates, the column and row input data are designed to be int data type. This changeColor function is designed to transform the decimal row and column data into binary int arrays. This design proved to be functional, leading to an extremely long calculating time, especially when clearing the full screen.

We also write three functions to print the square, triangle, and circle on the screens. By scanning all the points of the screen, the programs are designed to print all these patterns with less than three points input. These functions were designed to have an interaction with the mouse inputs. However, since the insight into VGA signals is yet not perfect, we still can't make those functions fully work.

4.4 Display Circuit with Memory Chip

As previously mentioned, the counters from the display circuit will supply the coordinates of the pixels into the memory chip. However, when writing data into the memory chip, the microcontroller should supply an address to the memory chip while the memory chip should ignore the coordinates supplied by the counters. In order to achieve this, we are using four quad-mux TTL chips (SN74LS157N) to select the right signal depending on if we are in the reading state or writing state. When in reading state, the mux will supply the coordinates from the counters as the address of the pixels. When in writing state, the mux will supply the coordinate supplied by the microcontroller to write the data into the desired location.

After the data is loaded into the SRAM chip, the display circuit will supply the x coordinate of the pixel from the horizontal counter and the y coordinate of the pixel from the vertical counter. When implementing, we can just connect the bit-shifted coordinates into the first 16 address pins of the two memory chips directly. Then, one of the memory chips will have the write-enable pin set to 0, meaning that we are reading from this chip, and the other will have the write-enable set to 1, meaning that we are writing into this chip. We will swap the function of those two memory chips when there is any pixel change to achieve a stable buffer swap. The output of the memory chips will just be connected to the monitor through a color voltage divider circuit as previously mentioned.

Without changing the data in the SRAM, we successfully displayed all the random data.

```
// method to store color
void changeColor(int col,int row,int color[]) {
  for (k_color = 0; k_color < 8; k_color++) //decimal to binary converter
  {
      x_coor[7-k_color] = col % 2;
      col /= 2;
      y_coor[7-k_color] = row % 2;
      row /= 2;
  }
  for (i_color = 22; i_color < 30; i_color++) //iterate through columns
  {
    if (x_coor[i_color-22] == 1)
    {
      digitalWrite(i_color,HIGH);
    }
    else if (x_coor[i_color-22] == 0)
    {
      digitalWrite(i_color,LOW);
    }
  }
  for (j_color = 30; j_color < 38; j_color++) //iterate through rows
  {
    if (y_coor[j_color-30] == 1)
    {
      digitalWrite(j_color,HIGH);
    }
    else if(y_coor[j_color-30] == 0)
    {
      digitalWrite(j_color,LOW);
    }
  }for (l = 38; l < 44; l++) //iterate through color bits
  {
    if (color[l-38] == 1)
    {
      digitalWrite(l,HIGH);
    }
    else if (color[l-38] == 0)
    {
      digitalWrite(l,LOW);
    }
  }
}
```

Figure 16: changeColor() Function

Clearing the screen or displaying a solid color on the whole screen was also successful when controlled by the keypad. However, when data with specific addresses were written into the memory chip, the circuit did not behave according to the desired functionality. For example, when a solid blue block of pixels was written into the memory chip, the bottom few lines of the pixels were not displayed in the right place. Also, when the mouse input was read by the microcontroller, it did create lines on the monitor but in rather random positions. A few debugging methods were applied to the circuit, but the true causes of the problem remain unknown. Also, because most time was spent debugging the circuit, the double buffering was not implemented in the software although it was implemented in the hardware.

5 Conclusion

5.1 Summary

From a hardware perspective, we did finish the display circuit although it took a little longer than we expected. We were able to read/write data from and into the memory chip and were very close to having a working circuit. One potential problem was the way that we supplied the coordinate although we checked that several times. There could be a deeper problem that we did not find due to some potential problem with the VGA signal. We generally followed our timeline, but the debugging phase took much longer than we expected. We could have made the timeline more compact and left more time for debugging.

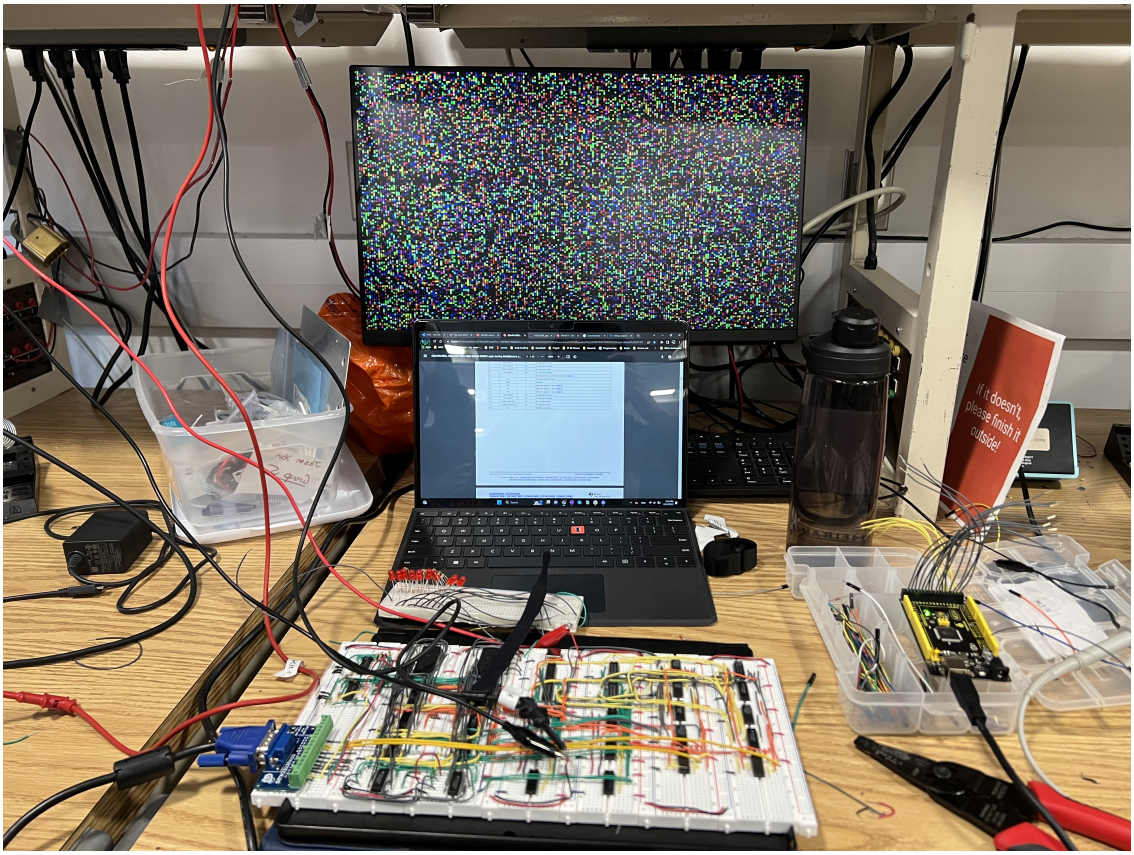

Figure 17: Random Data Displayed in the SRAM

Figure 18: Monitor Failed to Display a Solid Block

5.2 Challenges

- As mentioned previously, we did not realize the importance of the buffered oscillator at the beginning of this project. Although this project worked using an unbuffered oscillator, an oscillator with a proper buffer is highly recommended.
- As mentioned in section 4, the wiring of the circuit slowly became very messy. I suggest being very careful when you are wiring, or it could take hours to find one single wiring error.

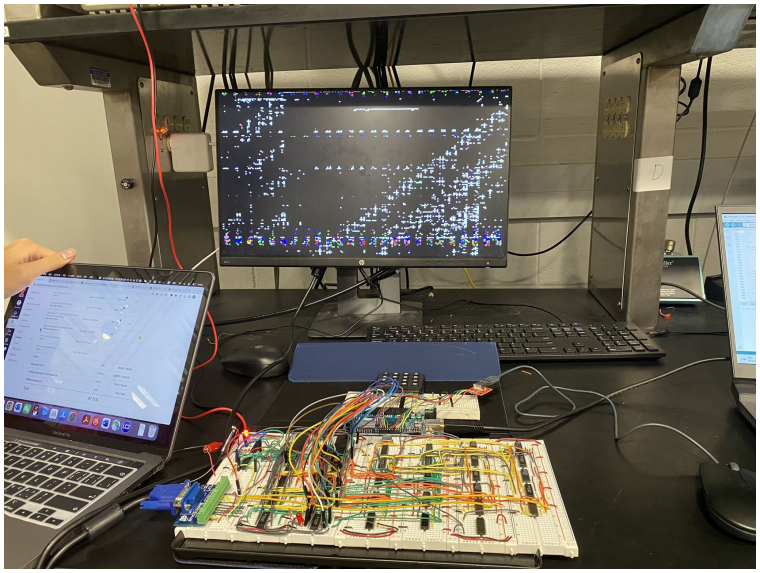

Figure 19: Affect of the Mouse Input

- When we are building our circuit on the breadboard, we first brought some cheap breadboards from Digikey. This was proven to be a bad idea after we found out that the holes in those breadboards could become loose quickly and the TTL chips wouldn't fit very well in them. This was a big problem, and we had to disassemble two sets of breadboards from ECE 120 Lab Kit and put them together to work as the breadboard for this project. If anyone was to do any breadboard-related projects, using high-quality breadboards is highly suggested although they could be expensive.
- Although we could properly set the entire monitor to a desired color, we've been struggling to modify a specific region of the monitor. Regions that aren't supposed to be modified are modified, and the display is somewhat corrupted.
- The display of the mouse is corrupted despite that we didn't spot any errors in our code. We didn't manage to find a fix.

5.3 Extensions of project

In the future, we should continue to debug our circuit and program to ensure that we can modify the monitor correctly. Once the mouse movements can be correctly displayed, we can add the draw-line function that we initially planned to implement. Additionally, there could be many more fun extensions of this project. For example, with this display circuit, we could potentially program the Arduino to play Tetris on a monitor purely from our circuit. Because of the nature of the display circuit, it could be integrated with a lot of other tasks that need to use a display. Tetris could be a good choice of idea although it is very programming-heavy. Overall, we believe there are many ways that we can extend this project, and it would be very likely that the project we do in the future will be related to this current project.

5.4 Significance

This project helped us understand the basic logic behind a VGA monitor and how to generate VGA signals. It also taught us how to interact with microcontrollers and memory chips. This could be useful for future classes like ECE 385 which we would interact a lot with VGA signals. Also, it gives us a chance to interact a lot with TTL chips. This project gives us a good insight into what will be happening in 385, although it will be on a higher level in 385. It has been a fun project with a considerable amount of time input on the display circuit side. We learned a lot both in a technical sense and about setting up timelines and project management.

6 Parts List

1* ACH-10.000MHZ-EK - 10 MHz Clock for the base clock of the digital circuit	digikey.com/en/products/detail /abracon-llc/ACH-10-000MHZ-EK/675362	2.61 dollars
1* VGA D-sub DE15 3-Row 15-Pin Female Terminal Breakout Board Connector (DISCONTINUED)	fasttech.com/product/9669151-vga-d-sub-de15-3-row-15-pin-female-terminal	3.82 dollars
1* Arduino Nano - the microcontroller we are going to use	Can be found in ECE supply center	11.77 dollars
3* 74LS00 - used in digital circuits to process outputs	can be found in electronics services shop	N/A
4* 74LS04 - used in digital circuits as inverters	can be found in electronics services shop	N/A
8* 75LS30 - used in digital circuits to check pixel count	can be found in electronics services shop	N/A
6* 74LS161 - used in digital circuits as counters	can be found in electronics services shop	N/A
1*AT28C64B-15PU - A EEPROM that we are going to test our circuit with	www.futureelectronics.com/p/semiconductors–memory–RAM–eeprom–parallel/at28c64b-15pu-microchip-2045833	5.54 dollars
1* 4*4 Matrix Keypad - the keypad we are going to use	https://www.digikey.com/short/28qn0r04	5.95 dollars
1* RPI-MOUSE BLK - the mouse we are going to use	https://www.digikey.com/short/1pd5jdfj	8 dollars
1* BOB-12700 - USB Breakout port to communicate with the mouse	https://www.digikey.com/short/p3phqrh0	4.95 dollars
2* AS6C4008-55PCN - the memory chip (buffer) that we will use	https://www.digikey.com/short/nzcmct9j	6.40 dollars * 2 = 12.8 dollars
1*ARDUINO MEGA BOARD 2560 - The arduino we will be using	Can be found at the supply center	56.52 dollars
1*DB15F-VGA-TERM - The VGA breakout ports we will use - the one from last semester is broken	https://mou.sr/3JJeORi	15.54 dollars
2* breadboard set from ECE 120 Lab Kit	from ECE 120 Lab Kit	N/A

Note that We also used a bunch of LEDs and resistors. Those things can be easily found in the Honors Lab room, ECEB OpenLab, and Electronic Service Shop.

7 References

1 B. eater, "Let's build a video card!", Eater.net, 2022. [Online]. Available: https://eater.net/vga.[Accessed: 25- Sep- 2022].

2 B. Eater, The world's worst video card?. 2019.

3 B. Eater, World's worst video card? The exciting conclusion. 2019.

4 jdh, I built my own graphics card. 2021.

5 W. Green, Project F. projectf.io, 2022 [Online]. Available: https://projectf.io/posts/video-timings-vga-720p-1080p/

6 ABRACON, LLC "HALF SIZE DIP LOW VOLTAGE 5.0V CRYSTAL CLOCK OSCILLATOR," ACH-10.000MHZ-EK datasheet [revised Nov. 2016]

7 Texas Instruments, "Synchronous 4-Bit Counters," SN74LS161AN datasheet, Oct. 1976 [revised Mar. 1988]

8 Texas Instruments, "SNx400, SNx4LS00, and SNx4S00 Quadruple 2-Input Positive-NAND Gates," SN74LS00 datasheet, Dec. 1983 [revised May 2017]

9 Texas Instruments, "8-Input Positive-NAND Gates," SN74LS30, Dec. 1983 [revised Mar. 1988]

10 Texas Instruments, "Hex Inverters," SN74LS04, Dec. 1983 [revised Jan. 2004]

11 Desai, S. (2022, June 6). VGA pinout. Hardware connector pinouts and cables circuits wirings. Retrieved December 6, 2022, from "https://pinoutguide.com/Video/VGA15-pinout.shtml "

12 SECONS Ltd. (2008). SVGA signal 800 x 600 @ 60 hz timing. Retrieved December 6, 2022, from http://tinyvga.com/vga-timing/800x600@60Hz

13 Beneater (2018) Beneater/EEPROM-programmer: Arduino EEPROM programmer, GitHub. MIT. Available at: https://github.com/beneater/eeprom-programmer (Accessed: December 6, 2022).

14 Components101, "4x4 keypad module," Components101, 2021. [Online]. Available: https://components101.com/misc/4x4-keypad-module-pinout-configuration-features-datasheet. [Accessed: 03-Feb-2023].

15 Components101, "USB-A-Jack-Pinout" Components101, 2021. [Online]. Available: https://components101.com/sites/default/files/component-pin/USB-A-Jack-Pinout.png [Accessed: 04-Feb-2023].

16 Getis, "Getis/arduino-PS2-mouse-handler: PS2 mouse handler library for arduinos," GitHub, 25-Oct-2021. [Online]. Available: https://github.com/getis/Arduino-PS2-Mouse-Handler. [Accessed: 01-Apr-2023].

17 PixArt Imaging Inc., "PAW3515DB DS Simple 1.0 - epsglobal," PAW3515DB SERIES USB OPTICAL MOUSE SINGLE CHIP, Apr-2013. [Online]. Available: https://www.epsglobal.com/Media-Library/EPSGlobal/Products/files/pixart/PAW3515DB.pdf?ext=.pdf. [Accessed: 04-Feb-2023].

8 Appendix - Code Used for Arduino

```
#include <PS2MouseHandler.h>
#define MOUSE_DATA 11
#define MOUSE_CLOCK 12
PS2MouseHandler mouse(MOUSE_CLOCK, MOUSE_DATA, PS2_MOUSE_REMOTE);
long lastDebounce = 0;
long debounceDelay = 300;
int counterofPast = 0;
char testData[4][4] = {{'1','2','3','A'},{'4','5','6','B'},
                       {'7','8','9','C'},{'*','0','#','D'}};
char p; //pressed button
//mouse coordinates (USE THESE FOR CURRENT COORDINATES)
int totMovementCol = 100;
int totMovementRow = 75;
//previous coordinates
int prevX = 100;
int prevY = 75;
//new coordinates
int newX;
int newY;
//change in coordinates
int deltaX;
int deltaY;
int scalingFactor = 25;
int x_coordinate[] = {0,1,1,0,0,1,0,0}; //100 in binary
int y_coordinate[] = {0,1,0,0,1,0,1,1}; //75 in binary
int colorOnes[] = {1,1,1,1,1,1};
int colorZeros[] = {0,0,0,0,0,0};
int colorRed[] = {1,1,0,0,0,0};
int colorGreen[] = {0,0,1,1,0,0};
int colorBlue[] = {0,0,0,0,1,1};
int colorYellow[] = {1,1,1,1,0,0};
int colorInput[] = {1,1,1,1,1,1}; // default white
int colorZero[] = {0,0,0,0,0,0};
int x_coor[] = {0,0,0,0,0,0,0,0};
int y_coor[] = {0,0,0,0,0,0,0,0};

int mouseClickCounter = 0;
int keyPadCounter = 0;
int inputSquareFirstPoint[] = {0,0};
int inputSquareSecondPoint[] = {0,0};
int inputTriFirstPoint[] = {0,0};
int inputTriSecondPoint[] = {0,0};
int inputTriThirdPoint[] = {0,0};
int inputCircleFirstPoint[] = {0,0};
int inputCircleSecondPoint[] = {0,0};
int radius = 0;

int invertLeftMux;
int invertRightMux;
int invertRightWE;

//loop counters
int i = 0;
int j = 0;
```

```
int k = 0;
int l = 0;
int i_color;
int j_color;
int k_color;

// method to store color
void changeColor(int col,int row,int color[]) {
  for (k_color = 0; k_color < 8; k_color++) //decimal to binary converter
  {
      x_coor[7-k_color] = col % 2;
      col /= 2;
      y_coor[7-k_color] = row % 2;
      row /= 2;
  }
  for (i_color = 22; i_color < 30; i_color++) //iterate through columns
  {
    if (x_coor[i_color-22] == 1)
    {
      digitalWrite(i_color,HIGH);
    }
    else if (x_coor[i_color-22] == 0)
    {
      digitalWrite(i_color,LOW);
    }
  }
  for (j_color = 30; j_color < 38; j_color++) //iterate through rows
  {
    if (y_coor[j_color-30] == 1)
    {
      digitalWrite(j_color,HIGH);
    }
    else if(y_coor[j_color-30] == 0)
    {
      digitalWrite(j_color,LOW);
    }
  }for (l = 38; l < 44; l++) //iterate through color bits
  {
    if (color[l-38] == 1)
    {
      digitalWrite(l,HIGH);
    }
    else if (color[l-38] == 0)
    {
      digitalWrite(l,LOW);
    }
  }
}

void setup() {
  //delay(200);
  // put your setup code here, to run once:
  Serial.begin(9600);
  //pins 2~9: Keypad
  pinMode(2,OUTPUT);
  pinMode(3,OUTPUT);
```

```
pinMode(4,OUTPUT);
pinMode(5,OUTPUT);
pinMode(6,INPUT);
pinMode(7,INPUT);
pinMode(8,INPUT);
pinMode(9,INPUT);

//pins 22~29: column bits
pinMode(22,OUTPUT);
pinMode(23,OUTPUT);
pinMode(24,OUTPUT);
pinMode(25,OUTPUT);
pinMode(26,OUTPUT);
pinMode(27,OUTPUT);
pinMode(28,OUTPUT);
pinMode(29,OUTPUT);

//pins 30~37: row bits
pinMode(30,OUTPUT);
pinMode(31,OUTPUT);
pinMode(32,OUTPUT);
pinMode(33,OUTPUT);
pinMode(34,OUTPUT);
pinMode(35,OUTPUT);
pinMode(36,OUTPUT);
pinMode(37,OUTPUT);

//red bits
pinMode(38,OUTPUT);
pinMode(39,OUTPUT);
//green bits
pinMode(40,OUTPUT);
pinMode(41,OUTPUT);
//blue bits
pinMode(42,OUTPUT);
pinMode(43,OUTPUT);

//MUX & Memory Signals
pinMode(49,OUTPUT); //
pinMode(50,OUTPUT); //Left Memory MUX
pinMode(51,OUTPUT); //Right Memory MUX
pinMode(52,OUTPUT); //Left Memory WE
pinMode(53,OUTPUT); //Right Memory WE

//keypad bits
digitalWrite(2,LOW);
digitalWrite(3,LOW);
digitalWrite(4,LOW);
digitalWrite(5,LOW);
digitalWrite(6,LOW);
digitalWrite(7,LOW);
digitalWrite(8,LOW);
digitalWrite(9,LOW);

digitalWrite(50,HIGH);
digitalWrite(51,HIGH);
```

```
  invertLeftMux = 1;
  invertRightMux = 1;
  digitalWrite(52,LOW);
  digitalWrite(53,LOW);
  invertRightWE = 0;

  digitalWrite(38,HIGH);
  digitalWrite(39,HIGH);
  digitalWrite(40,HIGH);
  digitalWrite(41,HIGH);
  digitalWrite(42,HIGH);
  digitalWrite(43,HIGH);

  if(mouse.initialise() != 0){
    // mouse error
    Serial.println("mouse error");
  };
  Serial.println(mouse.device_id());
}

unsigned long last_run = millis(); //mouse movement sample start time

void loop()
{
  if (millis() - last_run > 100) { //update mouse movement based on sampling period
    last_run = millis();
    mouse.get_data();
    changeColor(prevX,prevY,colorZeros); // set color of previous coordinates to black
    if (prevX > 0) { changeColor(prevX - 1,prevY,colorZeros); }
    if (totMovementCol < 256) { changeColor(prevX + 1,prevY,colorZeros); }
    if (totMovementRow > 0) { changeColor(prevX,prevY - 1,colorZeros); }
    if (totMovementRow < 157) { changeColor(prevX,prevY + 1,colorZeros); }
    deltaX = mouse.x_movement() / scalingFactor; // slow down mouse movement speed
    deltaY = mouse.y_movement() / scalingFactor;
    newX = totMovementCol + deltaX;
    newY = totMovementRow + deltaY;
    if (newX <= 0) { newX = 0; }
    if (newY <= 0) { newY = 0; }
    if (newX >= 255) { newX = 255; }
    if (newY >= 156) { newY = 156; }
    totMovementCol = newX;
    totMovementRow = newY;
    //display cursor as a 5-pixel crosshair
    changeColor(totMovementCol,totMovementRow,colorInput);
    if (totMovementCol > 0) { changeColor(totMovementCol - 1,totMovementRow,colorInput); }
    if (totMovementCol < 256) { changeColor(totMovementCol + 1,totMovementRow,colorInput);
    if (totMovementRow > 0) { changeColor(totMovementCol,totMovementRow - 1,colorInput); }
    if (totMovementRow < 157) { changeColor(totMovementCol,totMovementRow + 1,colorInput);
    prevX = totMovementCol; //set current coordinates to previous
                            coordinates after modifying the screen
    prevY = totMovementRow;
  }

  for(i = 2;i < 6;i++) //keypad code
  {
    digitalWrite(i,HIGH);
```

```
  for(j = 6;j < 10;j++)
  {
    if((millis() - lastDebounce) > debounceDelay)
    {
      if(digitalRead(j) == HIGH)
      {
        // something to define the following stuff
        p = testData[j-6][i-2];
        Serial.println(p);
        // all sorts of function
        lastDebounce = millis();
      }
    }
  }
  digitalWrite(i,LOW);
}

if(p == '1')
{
  if (colorInput[0] == 1)
  {
    colorInput[0] = 0;
  }
  else
  {
    colorInput[0] = 1;
  }
}
else if (p == '2')
{
  if (colorInput[1] == 1)
  {
    colorInput[1] = 0;
  }
  else
  {
    colorInput[1] = 1;
  }
}
else if (p == '3')
{
  if (colorInput[2] == 1)
  {
    colorInput[2] = 0;
  }
  else
  {
    colorInput[2] = 1;
  }
}
else if (p == '4')
{
  if (colorInput[3] == 1)
  {
    colorInput[3] = 0;
  }
```

```
  else
  {
    colorInput[3] = 1;
  }
}
else if (p == '5')
{
  if (colorInput[4] == 1)
  {
    colorInput[4] = 0;
  }
  else
  {
    colorInput[4] = 1;
  }
}
else if (p == '6')
{
  if (colorInput[5] == 1)
  {
    colorInput[5] = 0;
  }
  else
  {
    colorInput[5] = 1;
  }
}

if (mouse.clicked(0)==1)// Store the mouse input
{
  delay(300);
  inputSquareSecondPoint[0] = inputSquareFirstPoint[0];
  inputSquareSecondPoint[1] = inputSquareFirstPoint[1];
  inputSquareFirstPoint[0] = totMovementCol;
  inputSquareFirstPoint[1] = totMovementRow;

  inputCircleSecondPoint[0] = inputCircleFirstPoint[0];
  inputCircleSecondPoint[1] = inputCircleFirstPoint[1];
  inputCircleFirstPoint[0] = totMovementCol;
  inputCircleFirstPoint[0] = totMovementRow;

  inputTriThirdPoint[0] = inputTriSecondPoint[0];
  inputTriThirdPoint[1] = inputTriSecondPoint[1];
  inputTriSecondPoint[0] = inputTriFirstPoint[0];
  inputTriSecondPoint[1] = inputTriFirstPoint[1];
  inputTriFirstPoint[0] = totMovementCol;
  inputTriFirstPoint[1] = totMovementRow;

  changeColor(totMovementCol,totMovementRow,colorInput);
  Serial.println();
  Serial.print(totMovementCol);
  Serial.print(",␣");
  Serial.print(totMovementRow);
}

if (p == 'A') //function to draw rectangular
```

```
{
keyPadCounter++;
for (i = inputSquareFirstPoint[0];i <= inputSquareSecondPoint[0];i++);
{
  for (j = inputSquareFirstPoint[1];j <= inputSquareSecondPoint[1];j++);
  {
    changeColor(i,j,colorInput);
  }
}
keyPadCounter = 0;
inputSquareFirstPoint[0] = 0;
inputSquareFirstPoint[1] = 0;
inputSquareSecondPoint[0] = 0;
inputSquareSecondPoint[0] = 0;
mouseClickCounter = 0;
Serial.print("rectangular complete");
}

if(p == 'B')
{
  keyPadCounter++;
  if ((inputTriFirstPoint[0]==inputTriSecondPoint[0] &&
       inputTriFirstPoint[0]==inputTriThirdPoint[0])==0)
  {
    if ((inputTriFirstPoint[1]==inputTriSecondPoint[1] &&
         inputTriFirstPoint[1]==inputTriThirdPoint[1])==0)
    {
      int PointTest[] = {0,0};
      if (inputTriFirstPoint[0] > inputTriSecondPoint[0])
      {
        PointTest[0] = inputTriFirstPoint[0];
        PointTest[1] = inputTriFirstPoint[1];
        inputTriFirstPoint[0] = inputTriSecondPoint[0];
        inputTriFirstPoint[1] = inputTriSecondPoint[1];
        inputTriSecondPoint[0] = PointTest[0];
        inputTriSecondPoint[1] = PointTest[1];
      }
      int kTwo = (inputTriFirstPoint[0]-inputTriThirdPoint[0])/
                 (inputTriFirstPoint[1]-inputTriThirdPoint[1]);
      int kThree = (inputTriThirdPoint[0]-inputTriSecondPoint[0])/
                   (inputTriThirdPoint[1]-inputTriFirstPoint[1]);
      int BTwo = inputTriThirdPoint[1] - kTwo * inputTriThirdPoint[0];
      int BThree = inputTriThirdPoint[1] - kThree * inputTriThirdPoint[0];
      if (inputTriThirdPoint[1]>inputTriFirstPoint[1])
      {
        if (kTwo > 0 && kThree > 0)
          {
            for (i = 0;i < 200;i++)
            {
              for (j = 0;j < 150;j++)
              {
                if (j <= ((i * kTwo) + BTwo) && j >= ((i * kThree) +
                    BThree) && j >= inputTriFirstPoint[1])
                  {
                    changeColor(i,j,colorInput);
                  }
```

```
        }
      }
    }
  else if (kTwo > 0 && kThree < 0)
  {
    for (i = 0;i < 200;i++)
    {
      for (j = 0;j < 150;j++)
      {
        if (j <= ((i * kTwo) + BTwo) && j <= ((i * kThree) + BThree)
            && j >= inputTriFirstPoint[1])
        {
          changeColor(i,j,colorInput);
        }
      }
    }
  }
  else if (kTwo < 0 && kThree < 0)
  {
    for (i = 0;i < 200;i++);
    {
      for (j = 0;j < 150;j++);
      {
        if (j >= ((i * kTwo) + BTwo) && j <= ((i * kThree) + BThree)
        && j >= inputTriFirstPoint[1])
        {
          changeColor(i,j,colorInput);
        }
      }
    }
  }
}
else if (inputTriThirdPoint[1] < inputTriFirstPoint[1])
{
  if (kTwo > 0 && kThree > 0)
  {
    for (i = 0;i < 200;i++);
    {
      for (j = 0;j < 150;j++);
      {
        if (j <= ((i * kTwo) + BTwo) && j >= ((i * kThree) + BThree)
         && j <= inputTriFirstPoint[1])
        {
          changeColor(i,j,colorInput);
        }
      }
    }
  }
  else if (kTwo > 0 && kThree < 0)
  {
    for (i = 0;i < 200;i++);
    {
      for (j = 0;j < 150;j++);
      {
        if (j >= ((i * kTwo) + BTwo) && j >= ((i * kThree) + BThree)
            && j <= inputTriFirstPoint[1])
```

```
                {
                  changeColor(i,j,colorInput);
                }
              }
            }
          }
          else if (kTwo < 0 && kThree < 0)
          {
            for (i = 0;i < 200;i++);
            {
              for (j = 0;j < 150;j++);
              {
                if (j >= ((i * kTwo) + BTwo) && j <= ((i * kThree) + BThree)
                   && j <= inputTriFirstPoint[1])
                {
                  changeColor(i,j,colorInput);
                }
              }
            }
          }
        }
      }
    }
    Serial.print("Triangle complete");
}

if (p == 'C')
{
    keyPadCounter ++;
    radius = pow((inputCircleFirstPoint[0]-inputCircleSecondPoint[0]),2) +
             pow((inputCircleFirstPoint[1]-inputCircleSecondPoint[1]),2);
    for (i = 0;i < 200;i++);
    {
      for (j = 0;j < 150;j++);
      {
        if ((pow((i - inputCircleFirstPoint[0]),2) + pow((j -
             inputCircleFirstPoint[1]),2)) <= radius * 1.1)
        {
          changeColor(i,j,colorInput);
        }
      }
    }
    mouseClickCounter = 0;
    keyPadCounter = 0;
    inputCircleFirstPoint[0] = 0;
    inputCircleFirstPoint[1] = 0;
    inputCircleSecondPoint[0] = 0;
    inputCircleSecondPoint[1] = 0;
    radius = 0;
    Serial.print("Circle Complete");
}

 if (p == '*') //invert Right MUX & Right Memory WE signals
{
    if (invertRightMux == 0)
    {
```

```
    digitalWrite(51,HIGH);
    invertRightMux = 1;
  }
  else
  {
    digitalWrite(51,LOW);
    invertRightMux = 0;
  }
  if (invertRightWE == 0)
  {
    digitalWrite(53,HIGH);
    invertRightWE = 1;
    //set to floating mode
      //red bits
      delay(100);
    pinMode(38,INPUT);
    pinMode(39,INPUT);
    //green bits
    pinMode(40,INPUT);
    pinMode(41,INPUT);
    //blue bits
    pinMode(42,INPUT);
    pinMode(43,INPUT);

  }
  else
  {
    digitalWrite(53,LOW);
    invertRightWE = 0;

    //set to output mode and set all to high
    //red bits
    pinMode(38,OUTPUT);
    pinMode(39,OUTPUT);
    //green bits
    pinMode(40,OUTPUT);
    pinMode(41,OUTPUT);
    //blue bits
    pinMode(42,OUTPUT);
    pinMode(43,OUTPUT);
    /*
    digitalWrite(38,HIGH);
    digitalWrite(39,HIGH);
    digitalWrite(40,HIGH);
    digitalWrite(41,HIGH);
    digitalWrite(42,HIGH);
    digitalWrite(43,HIGH);
    */
  }
}

if (p == '0')
{
 // digitalWrite(53,LOW);
  unsigned long prev = millis();
  for (i = 0;i < 256;i++)
```

```
    {
      for (j = 0;j < 157;j++) //155 covers all y's except for one row
      {
        changeColor(i,j,colorYellow);
      }
    }
    unsigned long now = millis();
    totMovementCol = 100;
    totMovementRow = 75;
    Serial.println(now - prev);
    Serial.println("clear complete!");
    //digitalWrite(53,HIGH);
  }
  p = 'P';
}
```