2021/2022 Electronics A-level Project
Pixel Art

Jude Waide
Manchester Grammar School
Candidate Number: 8403
Centre Number: 32371

1 System Planning

1.1 Problem Analysis

The goal of this project is to allow a user to draw pixel art on an LED display using a computer mouse. A cursor will be controlled through the movement of the mouse. Each pixel of the display can be in one of two states, on or off. The pixel the cursor is currently on will switch states when the left mouse button is pressed.

The display should be smooth with no visible flicker, meaning it must have a refresh rate above the flicker fusion threshold[1] - 50Hz. Flickering lights above this this threshold appear continuous. This project will be using an IBM MO098k mechanical ball mouse, which uses the PS/2 protocol. This is a bidirectional serial synchronous protocol and uses two lines for communication, data and clock. The device, in this case the mouse, always drives the clock line at a frequency between 10 – 16.7kHz. The exact frequency varies from mouse to mouse depending on its internal oscillator. During packet transmission, bits are read from the data line in a serial manner at the frequency of the clock. This means any system interfacing with the mouse needs to be able to operate at these speeds. The system needs to feel responsive, meaning latency needs to be lower than 0.1s for the system to feel instantaneous[2]. The display needs to be clearly visible from less than 5m away so the user can see what is being drawn.

This problem will be solved with the following subsystems:

- A packet sender to deliver packets to the mouse, following the PS/2 protocol
- A microcontroller to do the processing of moving a cursor and to read packets from the mouse
- An LED display on which to display the art
- A demultiplexer to select a column on the display from a 3-bit signal
- A RAM chip to store what is on each column
- An address counter to synchronise the RAM chip and demultiplexer, as well as drive the scanning of the LED display

1.2 SPECIFICATION

Quantitative

- The refresh rate of the display is at least 50Hz
- Packets can be sent to the mouse at any frequency between 10 – 16.7kHz
- The latency between input and output must be lower than 0.1s
- The display is visible from at least 5m away

Qualitative

- The cursor will move in the direction the mouse is moved
- Left clicking will change the state of the pixel underneath the cursor
- Holding down left click will only change the state of a pixel once
- Pixel art can be drawn

2 SYSTEM DEVELOPMENT

2.1 SUBSYSTEM #1 – LED DISPLAY

2.1.1 Overview

This subsystem has an 8x8 LED matrix with protective resistors and acts as the output for the entire system. Every LED should have the same power so the brightness across the screen is consistent. From the LED data sheet[3], the max peak current through an LED is 100mA and the max continuous current is 20mA. However, the demultiplexer can only source a maximum of 10mA meaning any one column may draw a maximum of 10mA.

Specification:

- Every LED must have the same power (±5% from average power)
- No more than 100mA can flow through an LED at any one time
- The max current draw of any one column is 10mA

2.1.2 System Design

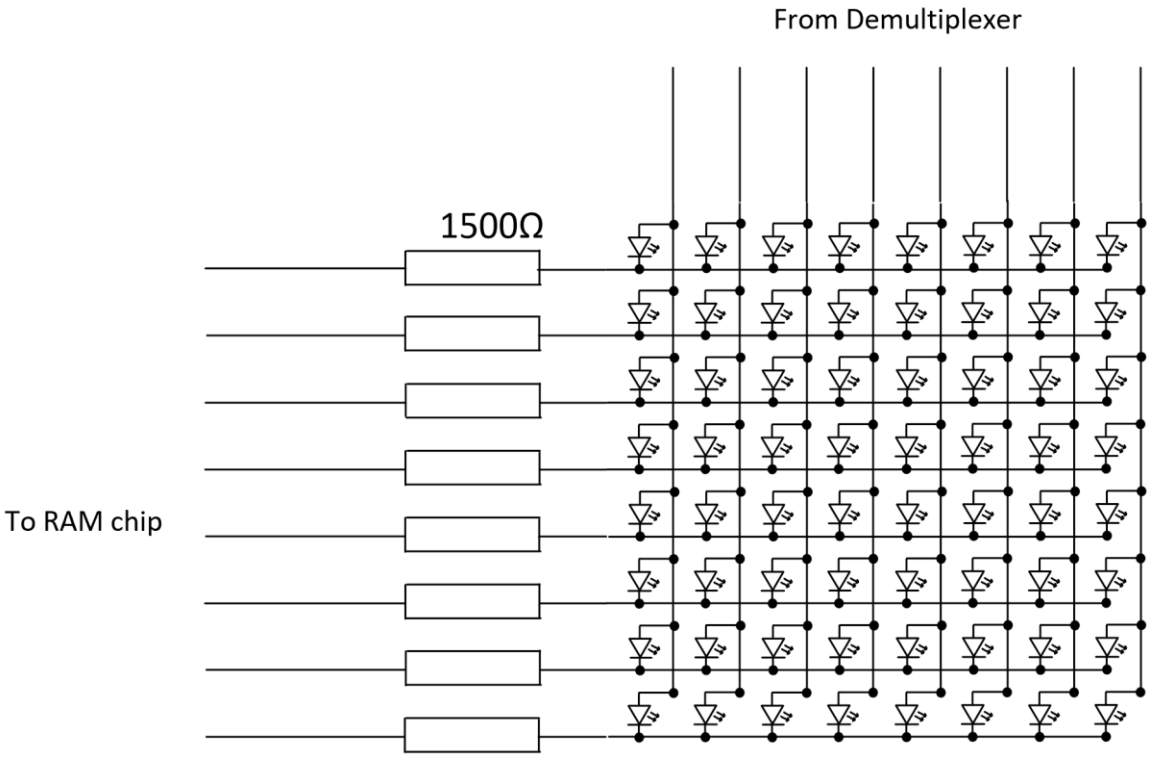

Figure 1 - Circuit diagram for display

Only one column will be turned on at once by the demultiplexer, supplying 5V to eight different LEDS. Any combination of LEDs in the column can then be turned on or off by the eight data lines. If 0V is supplied, there will then be a PD of 5V across the LED in that row, turning it on. If 5V is supplied, there will be a PD of 0V across the LED instead. As each row has its own resistor, the power across each LED remains constant regardless of how many are on at once. Increasing the number of activated LEDs instead just draws more current from the demultiplexer which is then sunk by the RAM chip.

According to the datasheet[3] for the LED matrix, the typical voltage across a white LED is 3.3V. As there is 5V across each LED and resistor pair, this leaves $5 - 3.3 = 1.7V$ across the resistor. As the max current through a column is 10mA, each LED in a column can have a maximum current of 1.25mA.

$$I = \frac{V}{R}$$

$$R = \frac{1.7}{1.25\text{x}10^{-3}}$$

$$R = 1360\Omega$$

Selecting the next highest value in the E24 series:

$$R = 1500\Omega$$

This should therefore limit the peak current through any one LED to well below the 100mA threshold and ensure the display will never draw more than 10mA from the multiplexer.

2.1.3 Testing

To test this subsystem, the current through the LEDs and voltage across them was measured. This was performed by connecting a single column to the 5V rail through an ammeter and taking the reading to find the total current through the column, repeating eight times for different columns. Next the voltage was measured by again connecting a single column to the 5V rail, this time connecting a voltmeter across the first LED in the column, repeating eight times for each column. Whist this does not get the voltage across every LED in the display, it is sufficient to obtain an average voltage.

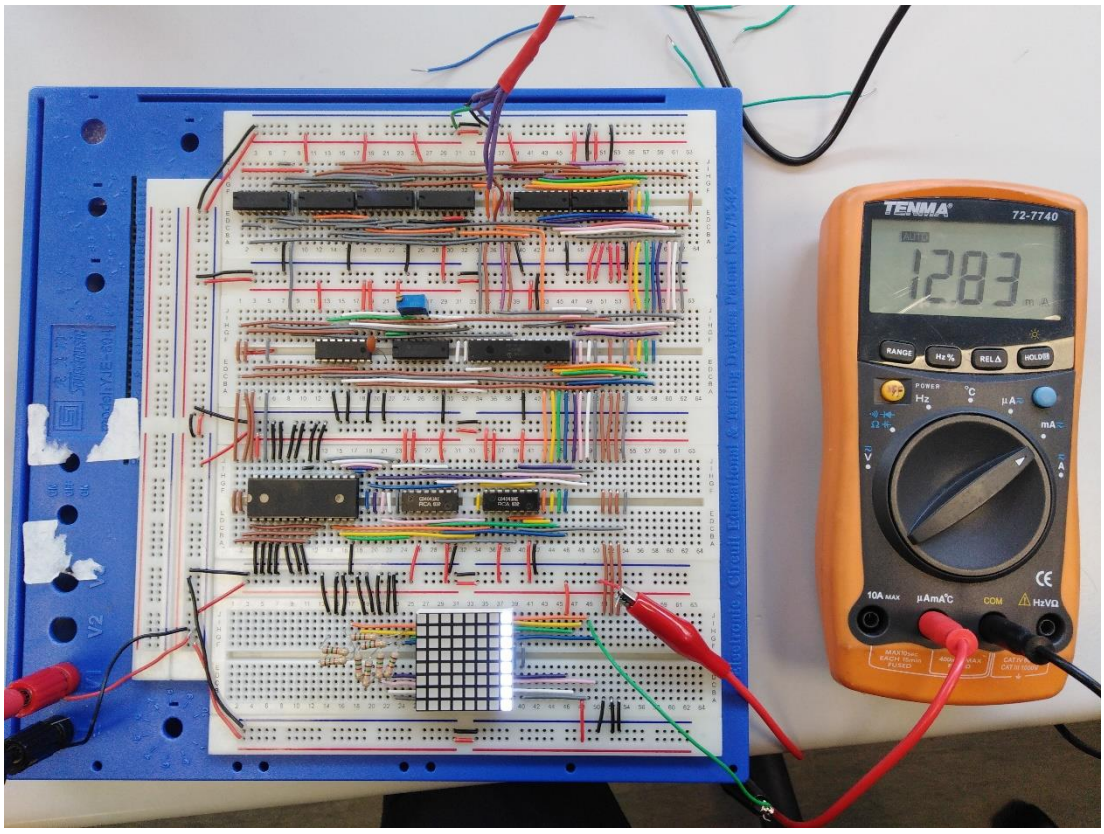

Figure 2 - Current of LEDs being measured

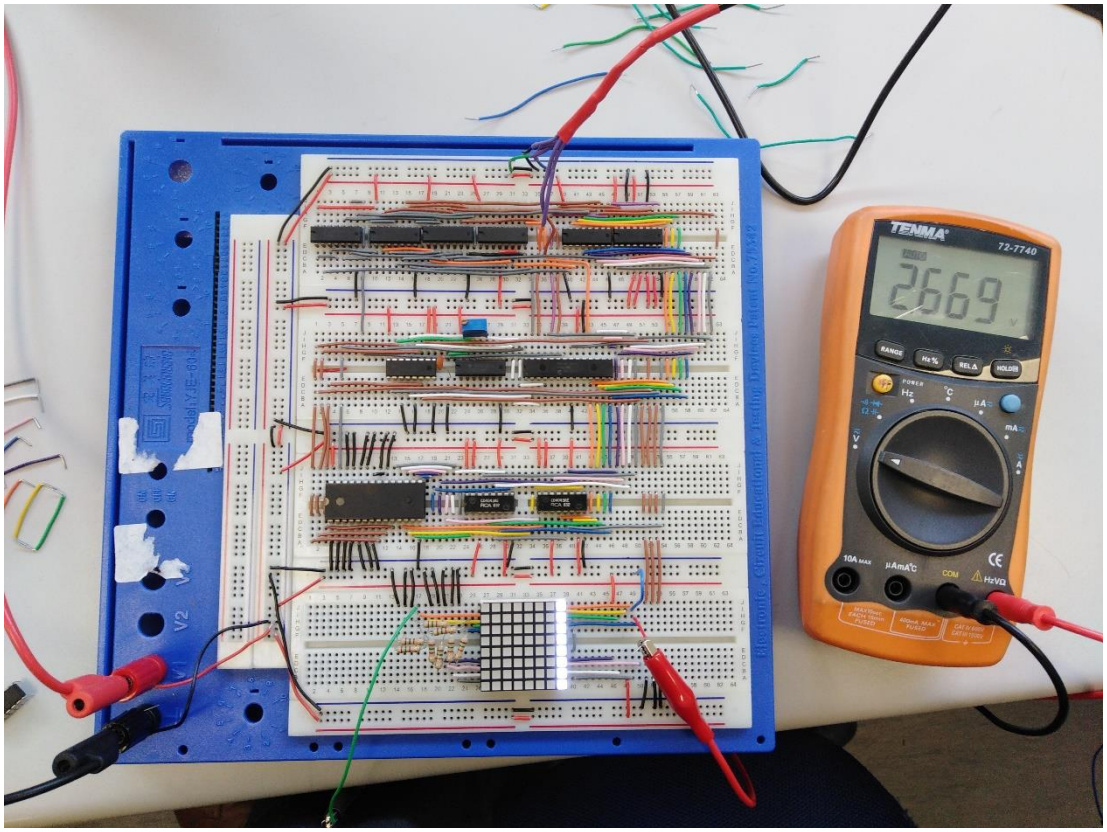

Figure 3 - Voltage of LEDs being measured

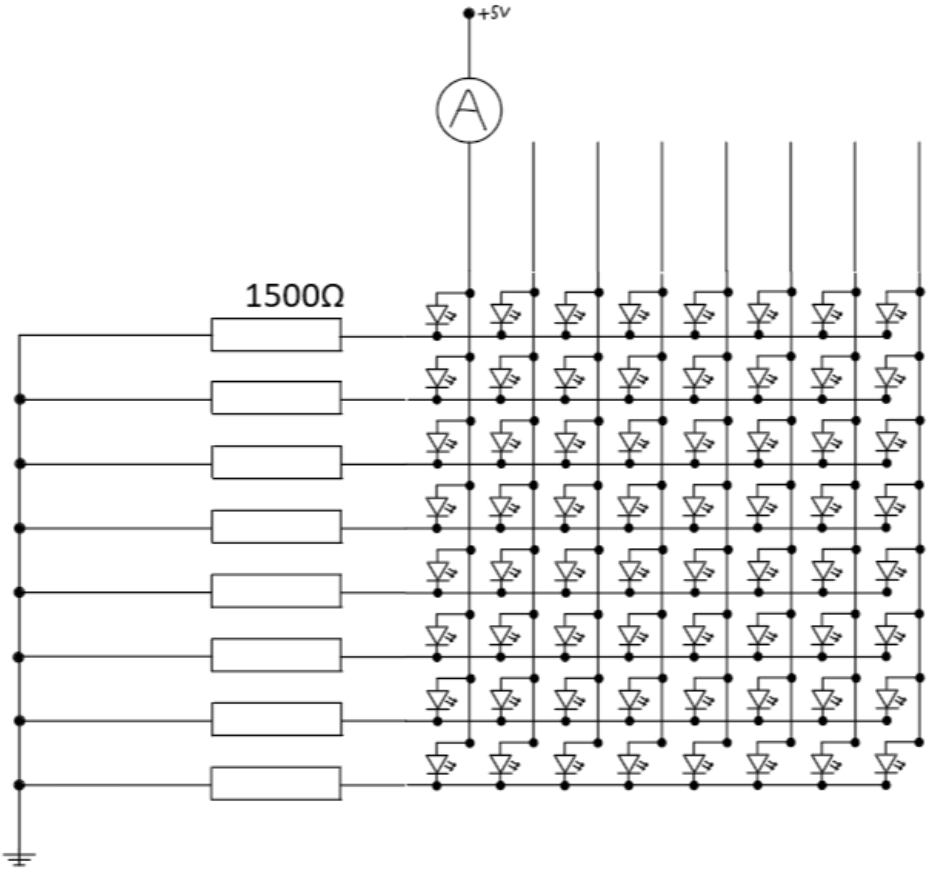

Figure 3 - Current diagram of current test

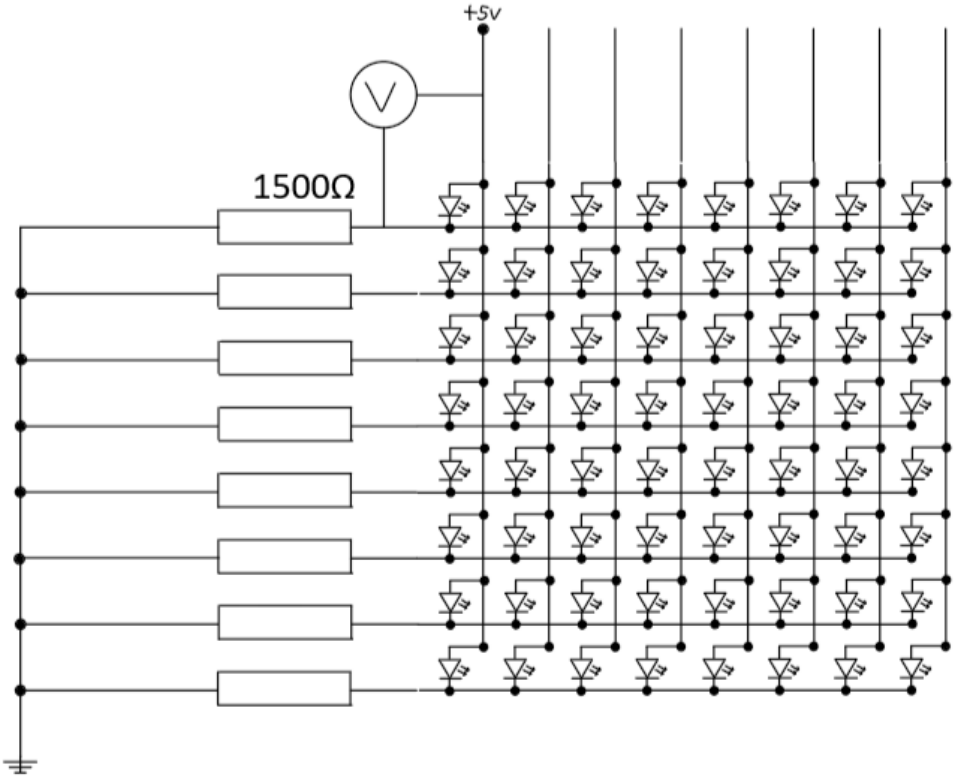

Figure 4 - Circuit diagram of voltage test

The results from the test were as follows:

Column	Current (mA)	Voltage (V)
1	12.84	2.67
2	12.75	2.68
3	12.77	2.69
4	12.82	2.67
5	12.62	2.71
6	12.68	2.77
7	12.63	2.74
8	12.62	2.71

The first thing to note about these results is that the voltage was not the expected 3.3V, with an average voltage instead of 2.705V. Calculating the new expected current with this voltage gives:

$$I = \frac{V}{R}$$

$$I = \frac{5 - 2.705}{1500} \times 8$$

$$I = 12.3mA\ (3sf)$$

This is very close to the recorded average column current of 12.7mA (3s.f) and is higher than expected. This is higher than the maximum current draw of 10mA for a single column, so this fails to meet the spec and could result in the LEDs not getting enough current when connected to the demultiplexer. Using this new voltage of 2.705V to ensure no more than 10mA is drawn would require resistor values of:

$$R = \frac{V}{I}$$

$$R = \frac{5 - 2.705}{1.25 \times 10^{-3}}$$

$$R = 1836\Omega$$

Next resistor up in the E24 series

$$R = 2000\Omega$$

This issue could therefore be easily fixed by using 2000Ω resistors instead of 1500Ω. The average current through a single LED is 1.59mA, an eighth of the average column current, which meets the specification of a maximum peak current of 100mA through any 1 LED. The average power of a column was 34.4mW (3s.f) and calculating the average power dissipation of each column gives:

Column	Power (mW)	Deviation (%)
1	34.3	0.328
2	34.2	0.656
3	34.4	0.128
4	34.2	0.483
5	34.2	0.568
6	35.1	2.117
7	34.6	0.613
8	34.2	0.568

No column had a power that deviated further than 5% from the average column power, the furthest deviation being 2.117%. Every LED in a column should have roughly the same power, so this indicates that every LED on the display should have roughly the same power and brightness. The circuit therefore meets the first specification point.

2.2 SUBSYSTEM #2 – DEMULTIPLEXER

2.2.1 Overview

The purpose of this subsystem is to turn on one of the eight outputs to the LED display, depending on a 3-bit binary input.

Specification:

- Must be able to supply 10mA at 5V
- Must turn on one of the eight outputs for a 3-bit bit input from the address counter

2.2.2 System Design

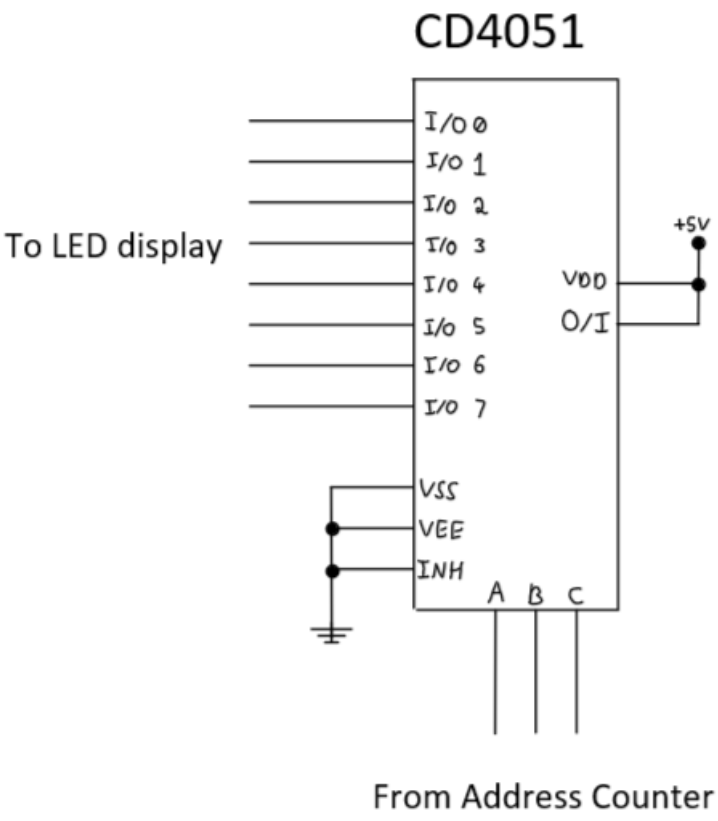

Figure 5 - Demultiplexer circuit diagram

A CD4051 analogue multiplexer/demultiplexer was used for this subsystem. There are three address bits, which will link the O/I pin with the corresponding I/O pin. For example, an address of '010' will connect the O/I pin to the I/O 2 pin. The O/I pin has been connected to +5V so that when an output is connected, +5V will be supplied. According to the datasheet[5], the CD4051 can provide a maximum DC output of 10mA, which is enough to satisfy the specification. Bringing the inhibit pin (INH) high will disable all outputs, so it has been connected to 0V as one output should always be on.

2.2.3 Testing

This subsystem was tested by measuring the voltage and current of each output at the maximum expected load to test if there was a difference in output between channels as well as if the demultiplexer could supply enough current. The demultiplexer needs to supply 10mA at 5V, meaning the maximum expected load is:

$$R = \frac{V}{I}$$

$$R = \frac{5}{10 \times 10^{-3}}$$

$$R = 500\Omega$$

A 1kΩ potentiometer was measured with a multimeter and adjusted until its resistance equalled 500Ω. The address bits were then set to ‘000’ to connect the I/O 0 pin to 5V. This was then passed through the potentiometer and an ammeter, with a voltmeter measuring the voltage across the potentiometer. This process was repeated 8 times, for the 8 different possible values of the address bits.

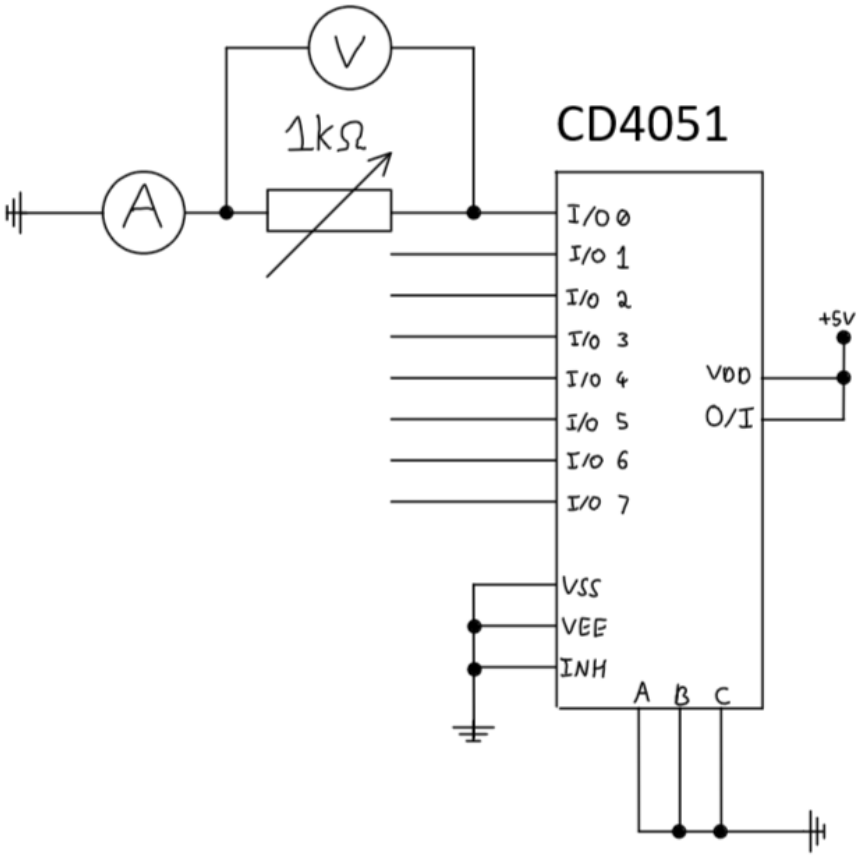

Figure 6 - Circuit diagram of first test

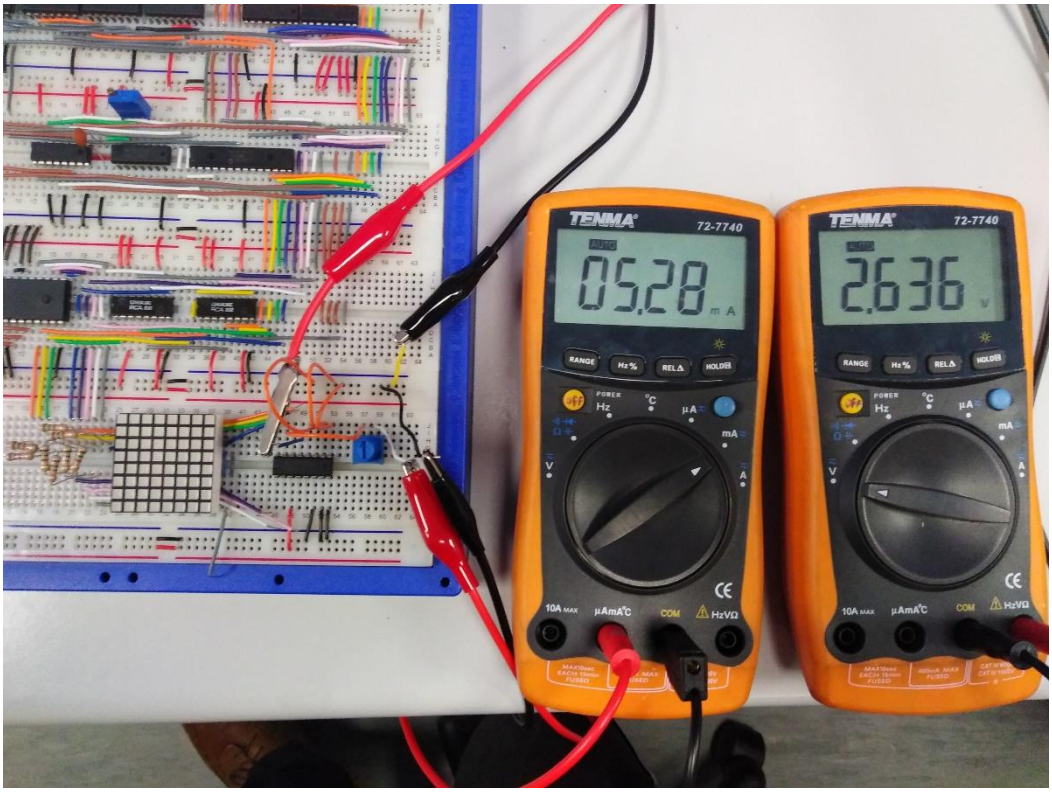

Figure 7 - Picture of first test

Output	Current (mA)	Voltage (V)
0	5.28	2.63
1	5.25	2.62
2	5.25	2.62
3	5.25	2.62
4	5.25	2.63
5	5.28	2.64
6	5.27	2.64
7	5.33	2.67

Figure 8 - Results of first test

The second test recorded how the voltage output by the first channel responded to different load resistances. This was done by setting the three address bits to '000' to connect the 'I/O 0' pin to 5V. The resistance of a 10kΩ ohm potentiometer was then adjusted and measured using a multimeter, before being connected to the first output and ground. A voltmeter was then used to record the voltage across the resistor.

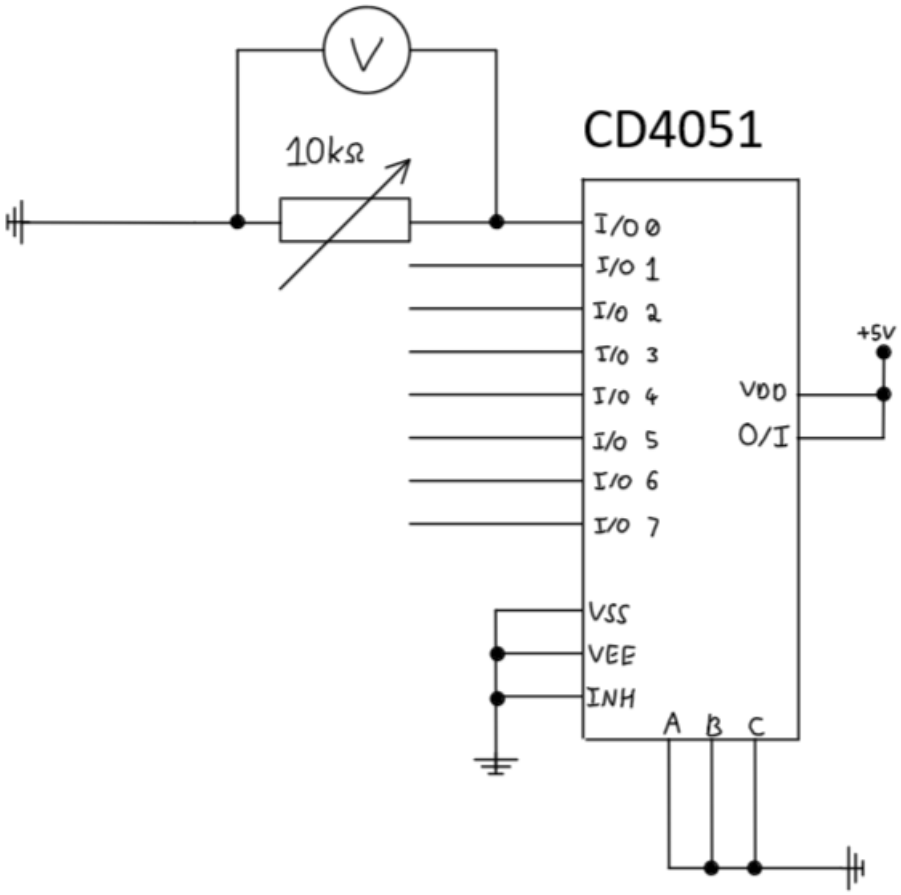

Figure 9 - Circuit diagram of second test

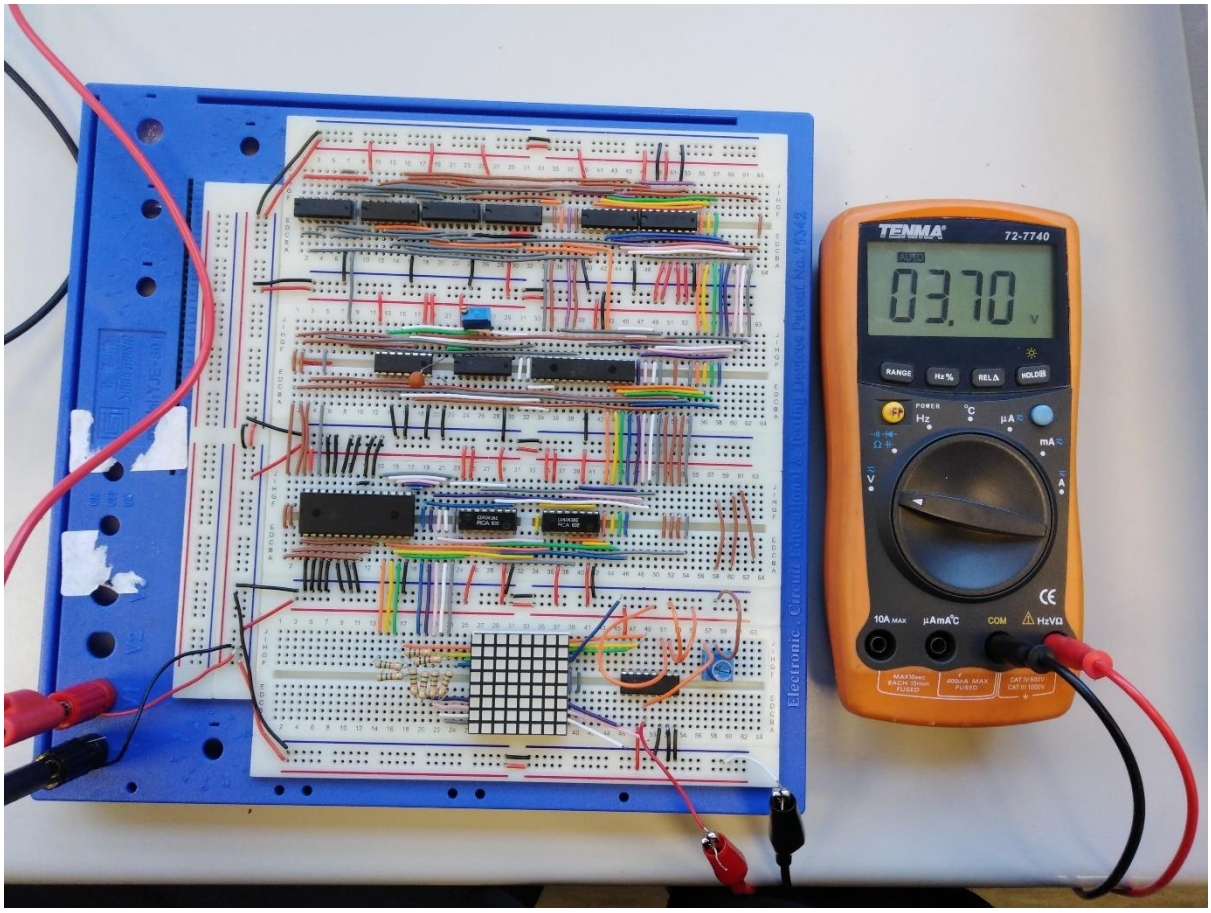

Figure 10 - Picture of second test

Resistance (kΩ)	Voltage (V)
8.98	4.81
7.94	4.78
6.96	4.76
5.91	4.72
4.97	4.68
4.08	4.61
3.03	4.49
1.99	4.25
1.04	3.66
0.5	2.63

Figure 11 - Results of second test

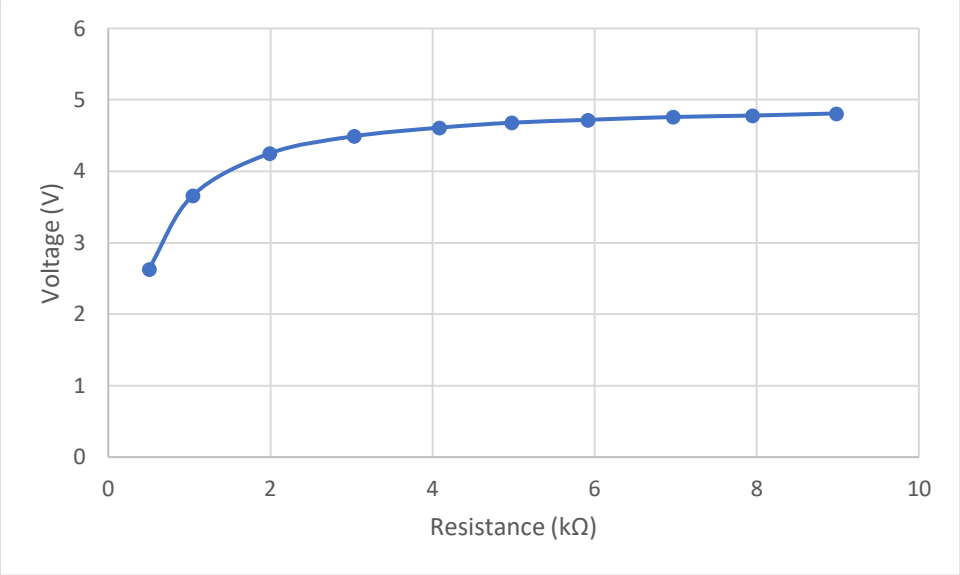

Figure 12- Results as a graph

From the results of the first test, two things are made apparent. The first is that there is no substantial difference in the output voltage and current between different outputs. This is important as if different outputs performed differently, the brightness of LEDs in different columns would not be consistent. The second thing to note is that the demultiplexer is only providing around half the expected voltage and current meaning it does not meet the second point of the specification. The graph of results from the second test demonstrates how the output voltage decreases as the resistance of the potentiometer decreases and more current is drawn. This fall off occurs as there is an increasing voltage drop across the internal resistance of the demultiplexer as higher currents are drawn, leading to a smaller voltage across the potentiometer and subsequently a smaller than expected current. The voltage across the demultiplexer can be found by subtracting the voltage of the potentiometer away from the supply voltage of 5V. The current through the multiplexer can also be found by finding the current flowing through the potentiometer using Ohm's law. Doing so produces the follow table:

Voltage (V)	Current (mA)
2.37	5.26
1.34	3.52
0.75	2.14
0.51	1.48
0.39	1.13
0.32	0.94
0.28	0.80
0.24	0.68
0.22	0.60
0.19	0.54

Plotting a graph of voltage against current will produce a graph with a gradient equal to the internal resistance of the demultiplexer:

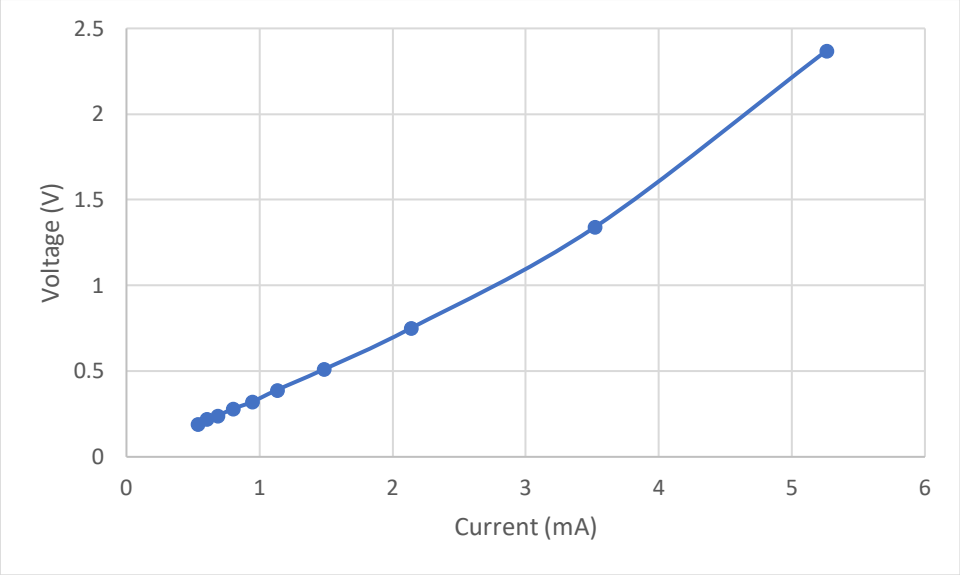

Figure 13 - A graph showing the internal resistance of the demultiplexer

This graph shows a straight line which then begins to curve up at higher currents. This increasing of resistance at higher currents is most likely due to more power being dissipated in the chip. This will mean the chip heats up more, increasing the internal resistance. By finding the line which passes through the final two points, this will give a straight line with a gradient close to the internal resistance of the demultiplexer at the desired current.

$$R = \frac{\Delta V}{\Delta I}$$

$$R = \frac{2.37 - 1.34}{5.26 - 3.52}$$

$$R = 0.592\text{k}\Omega \text{ (3s. f.)}$$

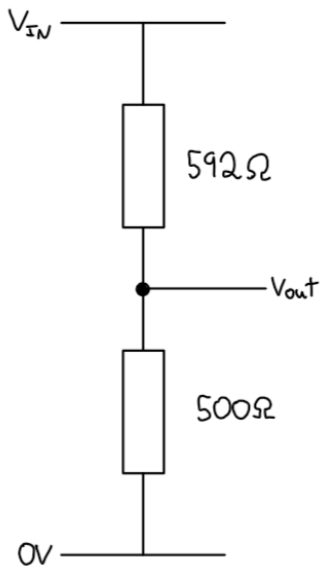

Figure 14 - Potential divider between the internal resistance and load

This explains why the output voltage is only half of what is expected, as the internal resistance of the demultiplexer and potentiometer effectively form a voltage divider. When the resistance of the potentiometer is equal to 500Ω it is close to the resistance of the potentiometer, causing the voltage at the output to be halved. This issue could be solved by increasing the voltage at the 'O/I' pin, effectively increasing the voltage across the potential divider in order to increase the output voltage. The voltage required can be calculated using the potential divider equation:

$$V_{out} = V_{in}\left(\frac{V_2}{V_1 + V_2}\right)$$

$$V_{in} = V_{out}\left(\frac{V_1 + V_2}{V_2}\right)$$

$$V_{in} = 5\left(\frac{592 + 500}{500}\right)$$

$$V_{in} = 10.92V$$

Supplying this voltage should allow the demultiplexer to supply the current required to meet the specification. In practise, the required voltage may be higher as the internal resistance further increases beyond 592Ω at higher currents.

2.2.4 Alternate Design

Another wat to build this subsystem would have been to build the logic of a demultiplexer rather than use a 4051.

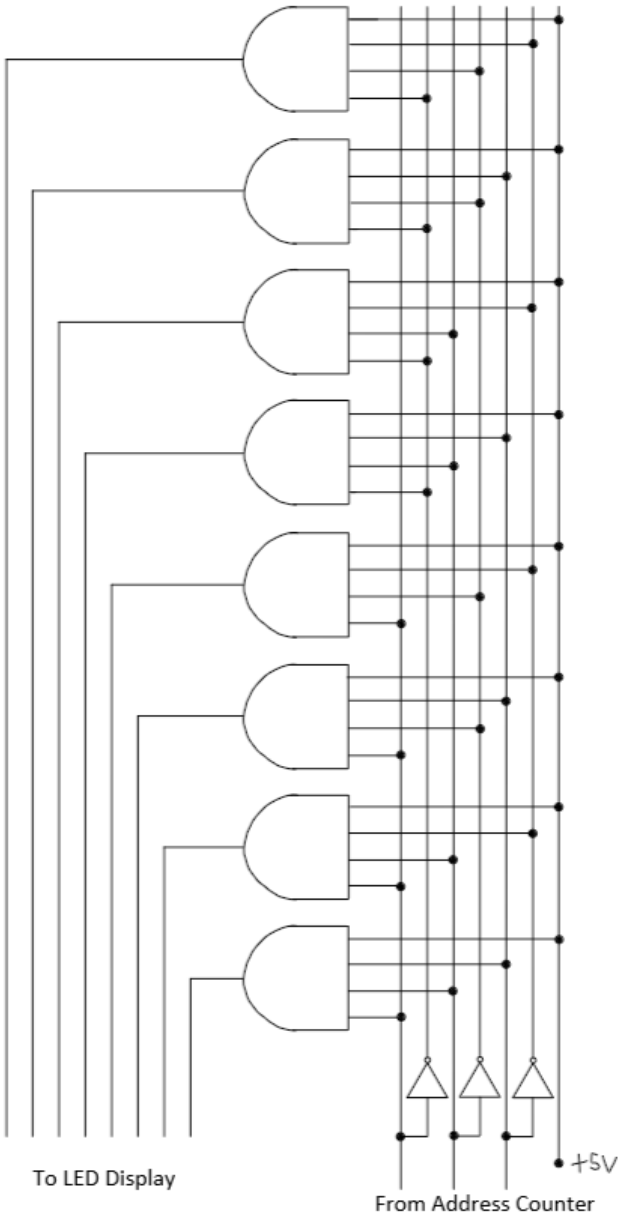

Figure 15 - Circuit diagram of alternate design

This design would have achieved the same thing as using the 4051, however would require more board space and be time consuming to wire. In the diagram, the furthest most address line is the least significant address line, with the left-most output being the first output. When all three address lines are low, only the top AND gate will be activated. Incrementing the address lines will move down the AND gates, activating one at a time. In summary, the original design was chosen as it did the same thing in fewer parts.

2.3 SUBSYSTEM #3 – RAM CHIP

2.3.1 Overview

The point of this subsystem is to store what is to be displayed on the LED display and output the contents of a column on the display when supplied with an address from the address counter.

Specification:

- Memory must be able to be written to through eight data lines
- Contents stored in memory must be able to be displayed on the eight data lines
- At least eight 8-bit memory locations must be available
- The address which is displayed/written to is controlled by three address bits from the address counter
- At least 10mA must be able to be sunk from the data lines

2.3.2 System Design

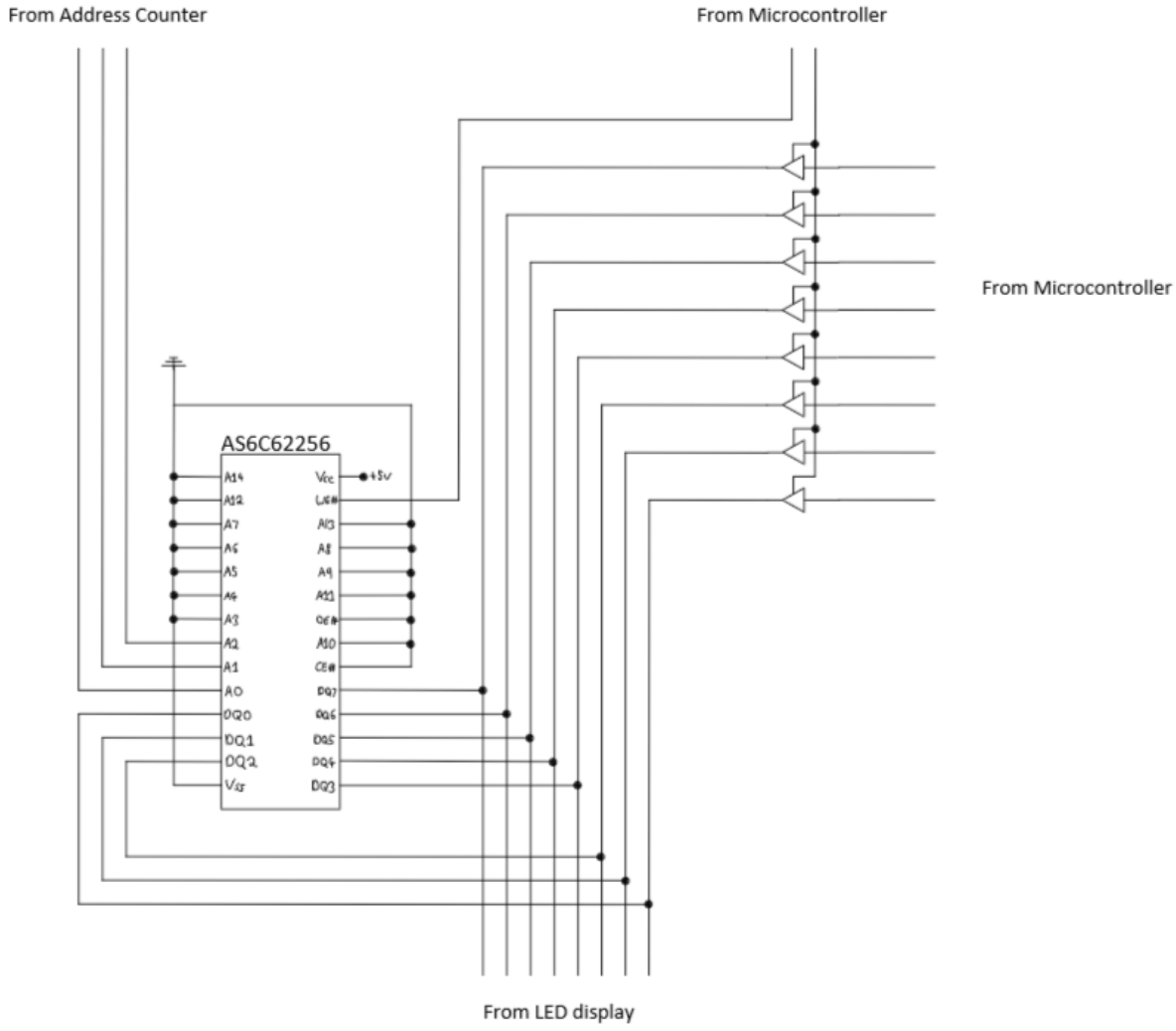

Figure 16 - RAM chip circuit diagram

The AS6C62256[4] SRAM chip has 32,786 8-bit memory locations and can source or sink 50mA, so should be able to meet all points on the specification. The chip has fifteen addressable bits ($A_0 – A_{14}$), however only three of these are used since only eight addresses are required (2^3). When displaying to the screen, the write enable-pin (WE#) is held high and A_0 to A_2 will be cycled through by the three address bits from the address counter. This will access the corresponding memory location and output it to the data pins, DQ0 to DQ1.

When writing to the AS6C62256 the tri-states will be activated by a signal from the microcontroller, allowing the eight data lines from the microcontroller to pass through. The write-enable pin is then brought low, causing the RAM chip to store the value provided to the eight data pins by the microcontroller to the provided address.

2.3.3 Testing

The reading and writing functionality of this subsystem was tested by manually recreating the input from the microcontroller whilst using a logic analyser to observe the RAM chip's eight data I/O pins. For this test the three address lines were brought to a logical low so that address 0 would be written to and read from. Initially, the WE# pin was held high to keep the chip in write mode and the tri-states disabled. First, the RAM chip was power cycled by disconnecting and reconnecting the power supply, causing random values to be stored in the memory. The value b'11111111' was then supplied to the inputs of the tri-states before enabling the tri-states to allow this value through. The WE# pin was then momentarily brought low so the RAM chip would store the provided value before entering reading mode again. Finally, the tri-states were disabled so the provided input no longer had any effect at the RAM chip's data I/O pins.

The expected result of this test is to first see a random value output by the RAM chip until the WE# pin is brought low. If after the tri-states have been disabled again the provided value remains, that means the RAM chip was successfully written to as otherwise the output would still be the original random value.

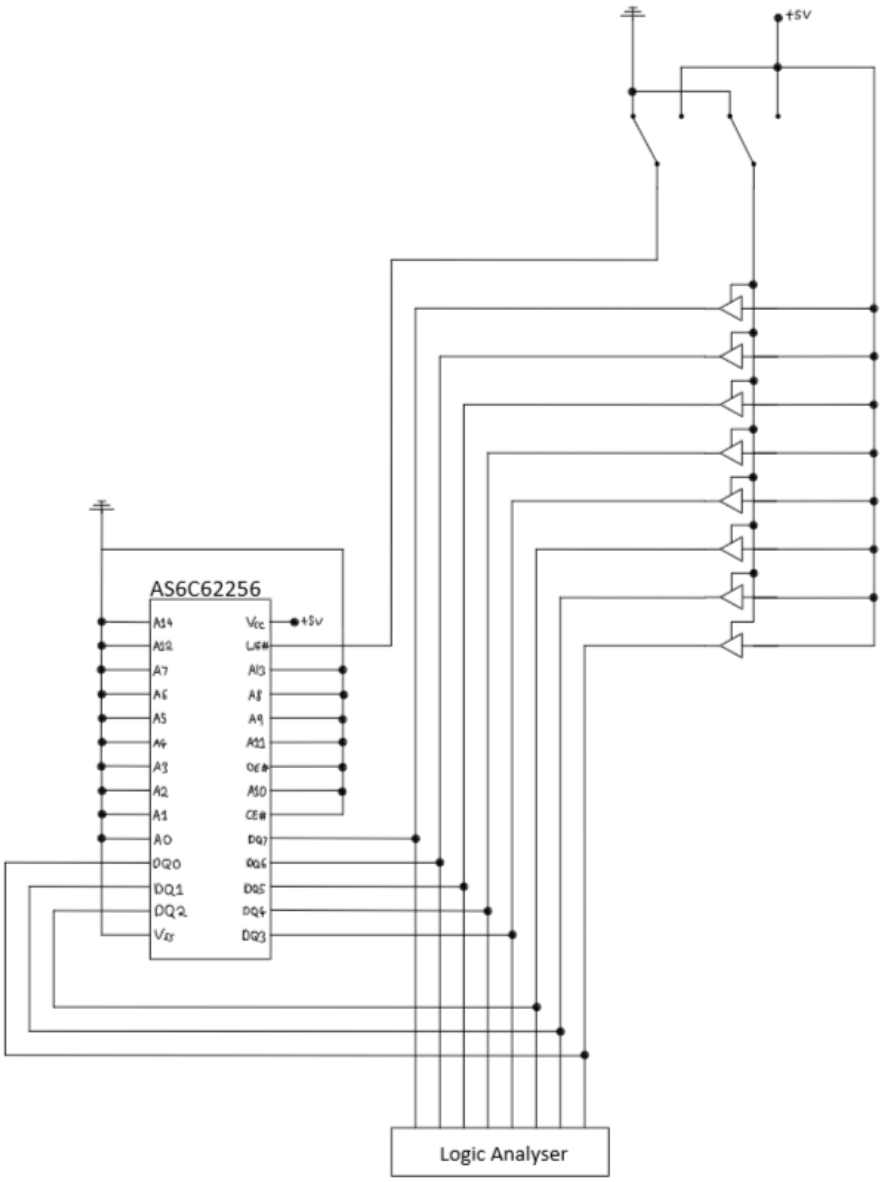

Figure 17 - Circuit diagram of test

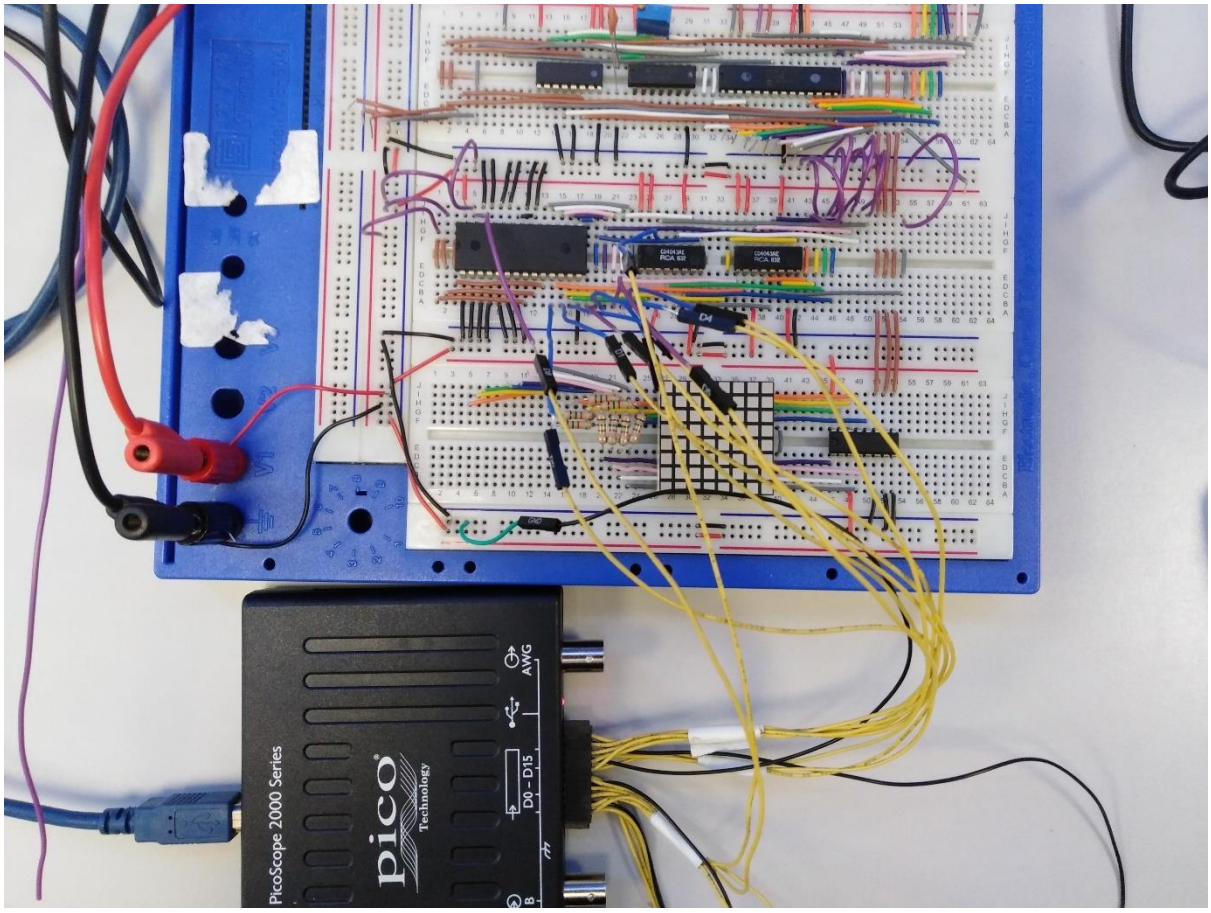

Figure 18 – Photo of test

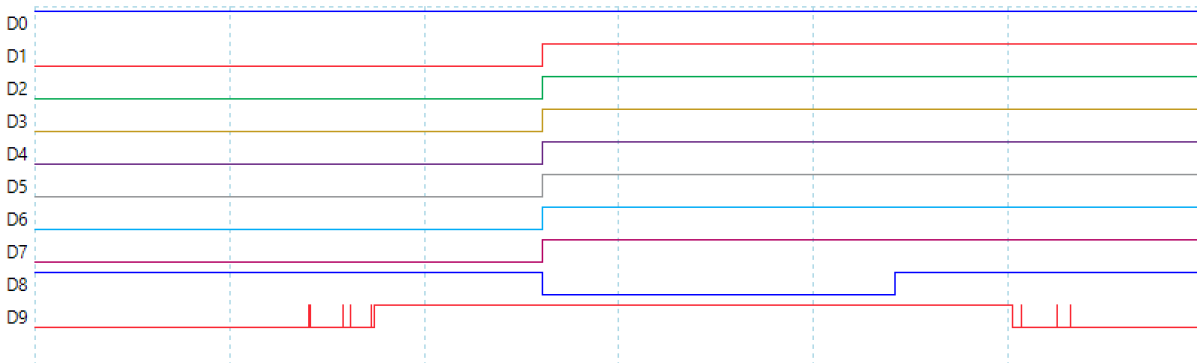

Figure 19 - Results of test. D0-D7 are the data inputs to the RAM chip, D8 is the WE# pin and D9 is the tri-state control

The results display the expected result. At first a random number is on the chip, b'10000000', which is output until the WE# pin is brought low, at which point the input from the tri-states is displayed. The value remains even after the tri-states are disabled, showing that the RAM chip was successfully written to and read from. Note that the spiking in the tri-state control line is due to switch bounce from the flying lead used. This test therefore demonstrates this subsystem meets the first two points of the spec.

One thing to note about this subsystem is that the RAM chip used had far more addresses than were needed, so a thinner chip with less available addresses would have been a better choice since the thickness of the chip made wiring harder.

2.4 Subsystem #4 – Address Counter

2.4.1 Overview

The purpose of this subsystem is to count from 0 to 7 repeatedly to cycle through addresses on the RAM chip and demultiplexer. Any value from 0 to 7 must also be able to be loaded in by the microcontroller. As the refresh rate of the display needs to be a minimum of 50Hz, the clock needs to count from 0 to 7 at least fifty times a second. This means the clock driving the counter must be at least eight times that, so a minimum of 400Hz.

The AS6C62256 (RAM chip) has an access time of 55ns and the 4051 (demultiplexer) has a max address-to-signal propagation delay of 720ns at 5V. This means that the next address is accessed and displayed for $720 - 55 = 665\text{ns}$ before the column of the demultiplexer changes. This blurring will only become noticeable if it makes up a significant fraction of the time period of the clock. Assuming the time period of the clock needs to be at least a hundred times as large for the blurring to be negligible, the minimum time period of the clock will be 66.5µs (A frequency lower than 15000Hz).

Specification:

- The subsystem must count from 0 to 7 fifty times every second
- The clock frequency should be between 400 and 15000Hz
- Any value from 0 to 7 must be able to be loaded into the counter by the microcontroller

2.4.2 System design

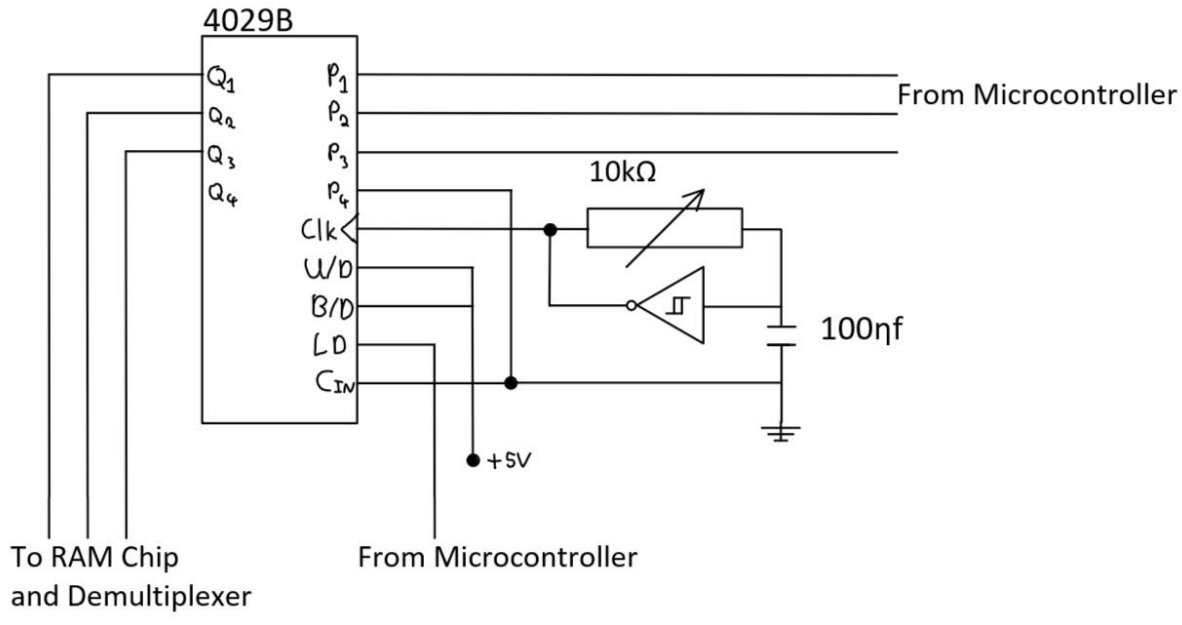

Figure 20- Circuit diagram for the Address Counter

The relaxation oscillator produces a 50% duty cycle astable pulse which feeds into the clock of the 4029B counter. The frequency of the relaxation oscillator needs to be at least 400Hz, therefore assuming the capacitor has a value of 100ηf:

$$f \approx \frac{1}{RC}$$

$$R \approx \frac{1}{FC}$$

$$R \approx \frac{1}{400 \times 100 \times 10^{-9}}$$

$$R \approx 25000\Omega$$

The maximum value R can take is roughly 25kΩ

The maximum frequency is 15kHz, meaning:

$$R \approx \frac{1}{15000 \times 100 \times 10^{-9}}$$

$$R \approx \frac{1}{15000 \times 100 \times 10^{-9}}$$

$$R \approx 665\Omega$$

The minimum value R can take is roughly 665Ω

Hence, a 10kΩ potentiometer is a suitable choice to provide well above the required refresh rate without blurring the display. The counter triggers on a rising edge to clock, counting from 0 to 15 with Q_1 being the least significant bit. Note that even though this is a 4-bit counter when a 3-bit counter is needed, the 4th bit can be ignored to have the same effect.

The Binary/Decade pin has been pulled high so the counter acts as a binary counter rather than a decade counter. It doesn't matter whether the up/down pin is pulled high or low, as the only thing it changes is in which direction the display is scanned. In this case, it has been pulled high so the counter will be counting up rather than down. C_{in} is used when cascading multiple counters together, so has been pulled low since there is only one counter.

When the load pin (LD) is brought high, the values in p_1 to p_4 will be loaded into the counter. Any rising edge at the clock pin will be ignored when LD is high, meaning the address won't be changed by the clock when the microcontroller is still trying to write to the RAM chip. The microcontroller can load in any value from 0 to 7 through pins p_1 to p_3. P_4 is insignificant as Q_4 is ignored regardless, so it can be pulled either low or high – in this case low.

2.4.3 Testing

To test this subsystem, a logic analyser was connected to the counter's output pins, Q_1 to Q_3,to measure the voltage levels. LD and Q_1 to Q_3 were pulled low so the output of the counter was not jammed. The expected result is the 3 outputs incrementing in binary from 0 to 7 continuously, Q_1 being the least significant bit. The frequency of the clock was then also measured by finding the time period between two changes in output, which should be in the range of 665-15000Hz.

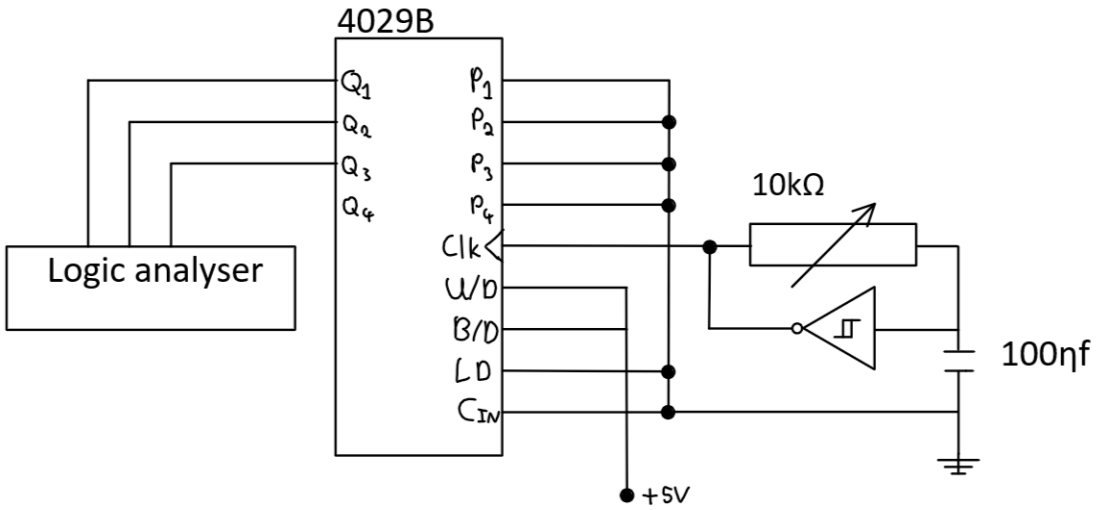

Figure 21 - Circuit diagram of test

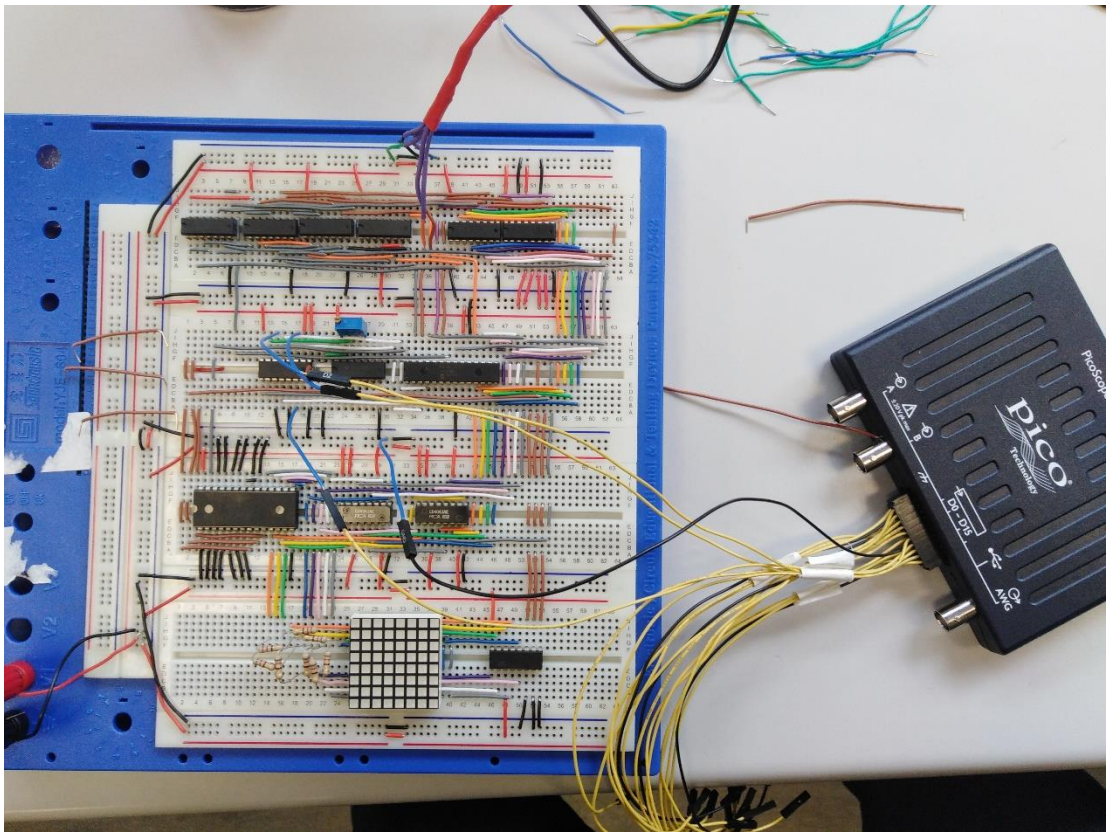

Figure 22 - Picture of test

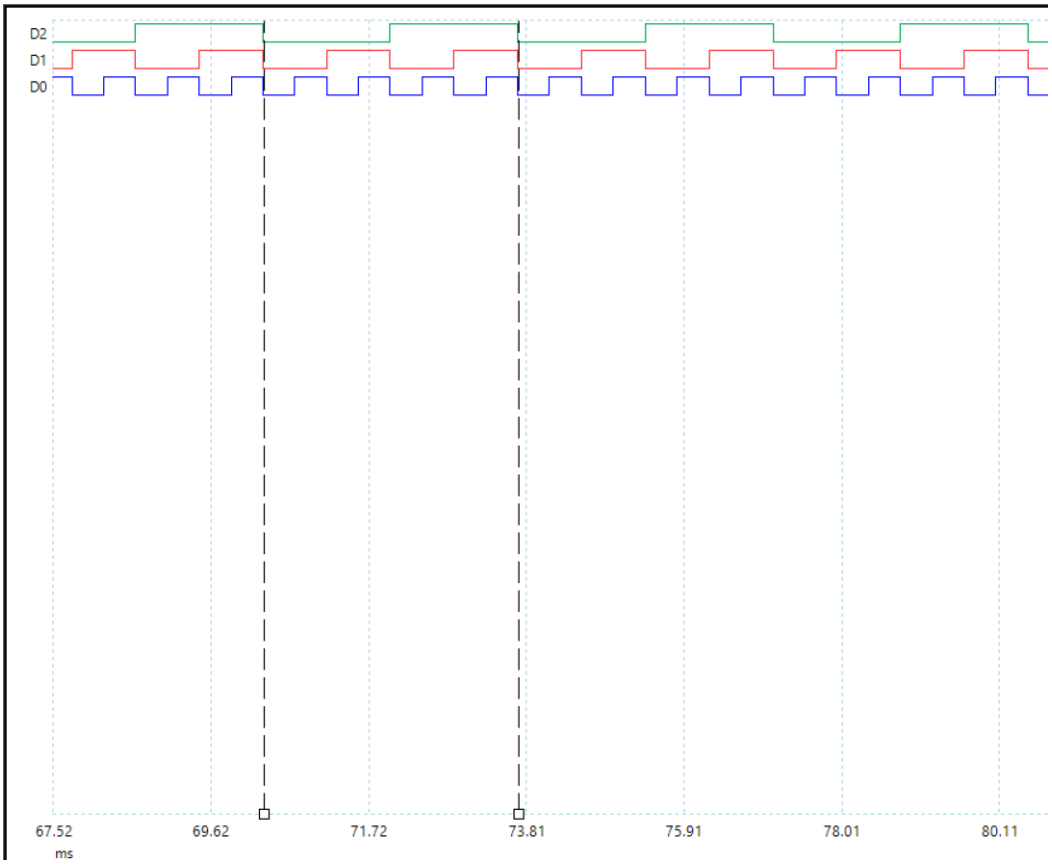

Figure 23 - Results of test. D0 is Q

Q_3	Q_2	Q_1	Denary
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Figure 24 - Truth table of repeating signal

As can be seen from the graph, there is a repeating signal with the output shown in the truth table above. This is the expected output of counting from 0 to 7 with Q_1 as the least significant bit. The time between the two points on the graph is the time it takes for the subsystem to count from 0 to 7, equating to 3.387ms.

$$f = \frac{1}{T}$$

$$f = \frac{1}{3.387 \times 10^{-3}}$$

$$f = 295\text{Hz (3s.f)}$$

This is well above 50Hz so the design meets the specification. The frequency of the clock will be eight times this value, equalling 2360Hz, which is within the 400-15000Hz range so again meets the specification.

2.4.4 Alternate design

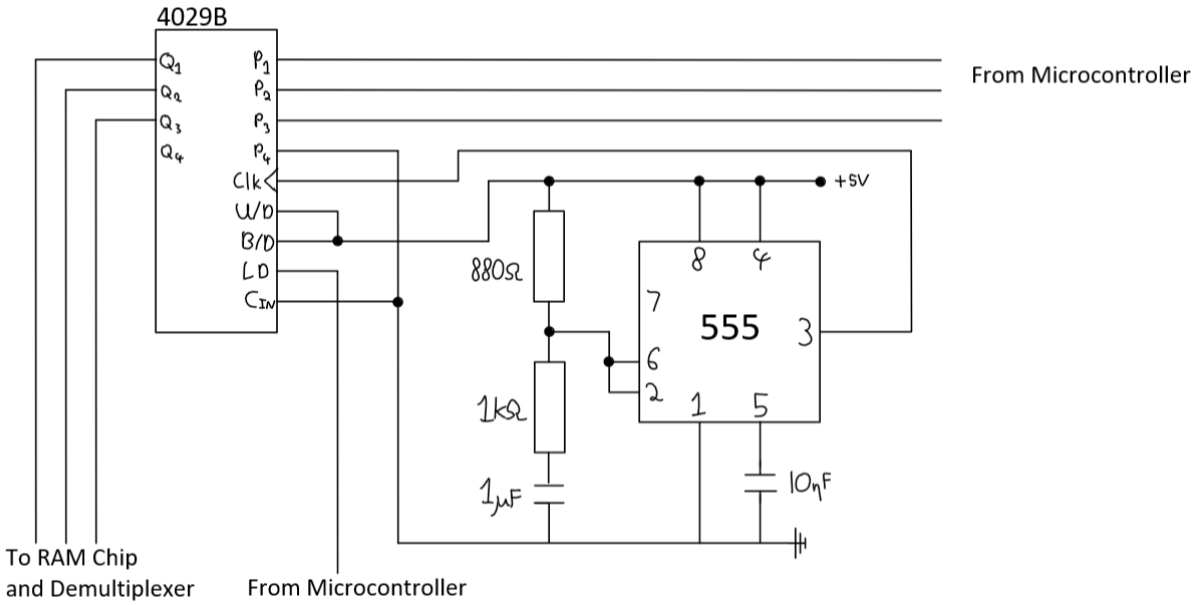

Figure 25 - Circuit diagram of alternate subsystem

An alternative way to have built this subsystem would be to use a 555 astable as a clock instead of an inverting Schmitt trigger. To achieve a frequency of 500Hz, letting R_2 = 1kΩ and C_1 = 1μF:

$$f = \frac{1.44}{C_1(R_1 + 2R_2)}$$

$$R_1 = \frac{1.44}{C_1 f} - 2R_2$$

$$R_1 = \frac{1.44}{1 \times 10^{-6} \times 500} - 2000$$

$$R_1 = 880\Omega$$

Using a 555 allows the duty cycle of the output to be controlled unlike the original design, however the duty cycle is unimportant here as only the frequency of the rising edges is important. Another advantage the 555 has is that it is a smaller chip compared to the 40106B, although this smaller chip size is mitigated by the increased component count, requiring an extra resistor and comparator, as well as more complex wiring. In conclusion the inverting Schmitt trigger was chosen over the 555 as the same effect can be achieved in fewer parts.

2.5 SUBSYSTEM #5 – MICROCONTROLLER

2.5.1 Overview

This subsystem has three main functions – to write a value to the RAM chip, to read packets sent by the mouse and to interface with the packet sender. It also does the processing of moving a cursor and handling mouse clicks.

Specification:

- The microcontroller must be able to send packets to the packet sender and execute the logic sequence required to get the packet sender to send a packet
- The microcontroller must be able to read packets from the mouse
- The microcontroller must be able to write to the RAM chip
- Must have at least nineteen I/O pins

2.5.2 System Design

To send a packet to the packet sender, the following sequence must occur:

1. The packet to be sent is presented to the packet sender, consisting of eight data bits and a parity bit
2. The packet sender is forced to listen to the clock line by bringing a control line low
3. The packet sender is brought into a ready to load state by bringing a second control line high
4. The clock line is brought low, causing the packet sender to load in the presented value
5. The first control line is released so the packet sender is no longer forced to listen to the clock
6. Bring the packet sender out of the ready-to-load state by bringing the second control line low
7. Remain in this state for at least 100ms
8. Give the packet sender control of the data line by bringing a third control line high
9. Release the clock line
10. Wait for 2ms for the packet to be sent
11. Remove the packet sender's control of the data line by bringing the third control line high

In total this requires thirteen pins - eight data bits, one parity bit, three control lines, the clock line.

To read a packet sent by the mouse, the microcontroller must be able to read the PS2 protocol. The mouse will send packets in 11-bit frames – a start bit, eight data bits, a parity bit (odd parity) and a stop bit. By default, both the clock line and data line are pulled high by the mouse. When the mouse is sends a packet it will produce an astable pulse on the clock line at a frequency between 10-16kHz until 11 falling edges have been produced. A falling edge signals to read the next bit from the data line and the mouse will change the logic level on the data line during the logical high of the generated clock pulse.

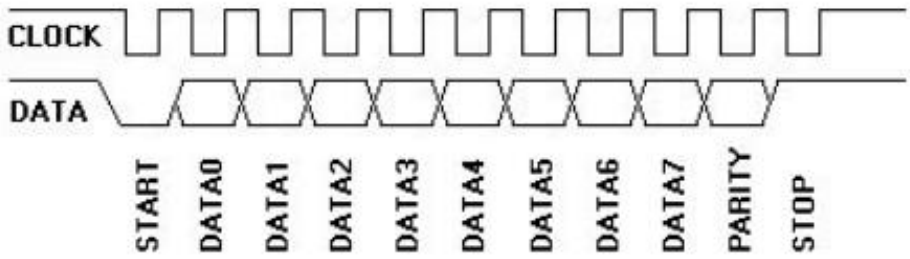

Figure 26 - device to host communication

The microcontroller will need to be able to understand this protocol and have high enough processing speeds to be able to read it.

To write a value to the RAM chip, the following sequence must occur:

1. The address on the RAM chip to be written to is output on three address lines to the address counter
2. The value to be written is output to the RAM chip on eight data lines
3. A control line is brought high to load the address onto the address counter and put the RAM chip into a ready-to-write state
4. A second control line is momentarily brought low to write the data to the RAM chip
5. The first control line is brought low again

This process requires thirteen pins, however as the RAM chip will never be written to at the same time as a packet is being sent the eight data pins can be shared between the two processes, bringing the total number of required pins down to nineteen.

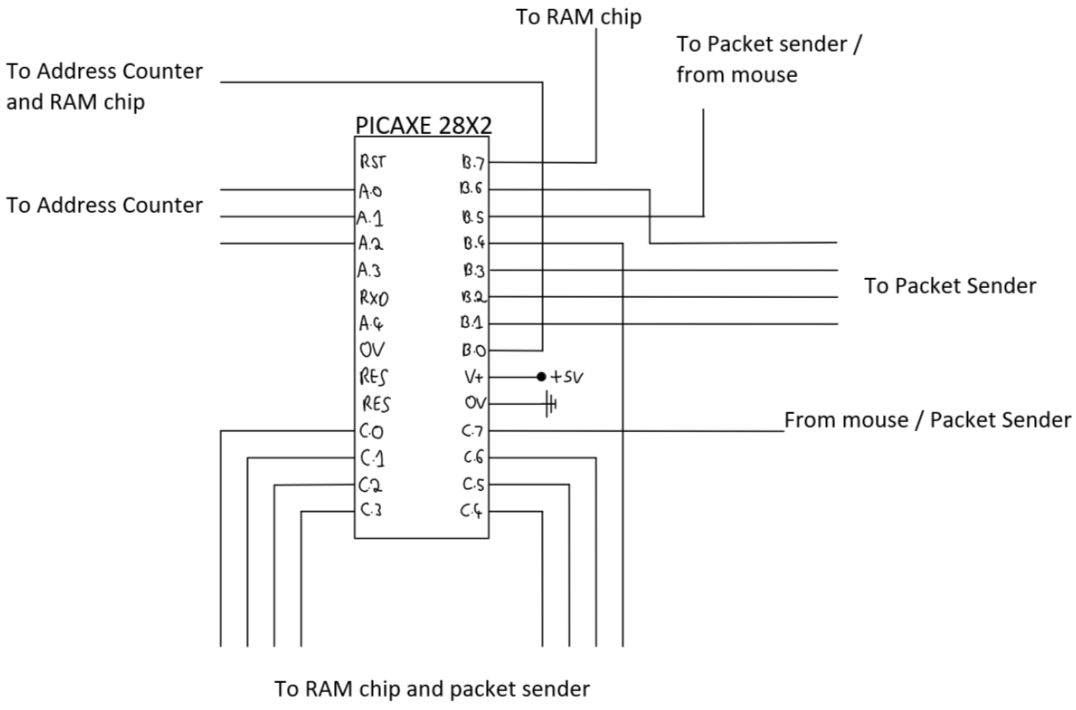

Figure 27 - Circuit diagram of microcontroller subsystem

Pin details:

A_0 – A_2: Address pins used to load values into the address counter

C_0 – C_6: 7/8 data out pins used to specify value to write to RAM chip/packet sender, C_0 is LSB

B_0: Enables loading of address counter and enables access to RAM chip

B_1: Forces clock pulse to be seen by packet sender

B_2: Parity bit of the packet loaded into the packet sender

B_3: Gets the packet loader into a ready to load state

B_4: Eighth bit of data out pins

B_5: Clock line which connects to the clock line of mouse and to the packet sender

B_6: Gives packet sender control of the data line

B_7: Bringing low puts RAM chip into writing mode

To meet these requirements a PICAXE 28X2 was used, on which the following code was executed:

```
setfreq M16       ;set frequency to 16MHz

hsersetup 319, %00001001      ;sets up hserin port to read packets from
mouse

;sets which pins are outputs and inputs for the three ports
dirsA = 255
dirsB = %11011111
dirsC = %01111111


high B.7    ;puts RAM chip into read mode
high B.1    ;stops packet sender from being forced to read clock



symbol column = b16     ;address to be written to
symbol value = b17      ;value to be written
symbol temp = b18       ;multipurpose cache
symbol packet = b19     ;first recieved byte of mouse packet
symbol xmov = b20       ;second recieve byte of mouse packet
symbol ymov = b21       ;third recieved byte of mouse packet
symbol x = b22          ;x location of cursor
symbol y = b23          ;y location of cursor
symbol col = b24        ;which column the cursor is on
symbol row = b25        ;which row the cursor is on
symbol counter = b26
symbol limit = b27
symbol clickFlag = b28  ;stores flags about status of mouse clicks

;setting eight addresses to increasing powers of 2 - becomes useful later
temp = 1
for bptr = 8 to 17
      @bptr = temp
      temp = temp * 2
next bptr
```

```
call clearDisplay       ;clears the LED display

goto start  ;go to start tag so procedures are not ran


display:
      let temp = INV value    ;inverts as a 0 will turn an LED on

      pinsA = column          ;ouputting the address to be written to
      let pinsC = temp AND %01111111      ;outputting the value to be
written

      if temp bit 7 set then  ;outputting eight bit to be written
            high B.4
      else
            low B.4
      endif

      high B.0    ;loads address and gets RAM chip ready to be written to
      low B.7     ;toggles WE# of RAM chip, writing the value
      high B.7
      low B.0     ;stops loading address and disables access to data lines
of RAM chip

      return

clearDisplay:
      ;calls display procedure once for each column and clears it
      value = 0
      for column = 0 to 7
            bptr = column
            let @bptr = value ;stores value in virtual display
            call display
      next column
      return

sendPacket:
      counter = 0

      ;calculates parity bit
      for bptr = 8 to 17
            temp = @bptr AND value  ;looks at each bit of address
            if temp > 0 then
                  counter = counter + 1   ;incraments counter if bit = 1
            endif
      next bptr
      let counter = counter % 2

      if counter = 0 then
            high B.2          ;brings high if even number to make odd
number
      else
            low b.2           ;else brings low
      endif

      let pinsC = temp AND %01111111      ;outputs eight data bits
      if temp bit 7 set then
            high B.4
      else
            low B.4
      endif
```

```
      low B.1                 ;force clock
      high B.3                ;bring load high
      low B.5                 ;take clock low
      high B.1                ;unforce clock
      low B.3                 ;take load low
      pause 1                 ;wait
      high B.6                ;bring data control high
      let dirsB = %11011111   ;release clock

      pause 2                 ;wait for packet to send

      low B.6                 ;take data control low
      return


start:


      pause 2000  ;wait for mouse initialisation

      value = 0xFF      ;resets mouse
      call sendPacket

      value = 0xF4      ;enables streaming mode of mouse
      call sendPacket

      hserinflag = 0    ;incraeents by three every time a
      hserptr = 0       ;next location where packet will be written

      column = 0        ;creates initial pixel - pixel is not stored in
virtual display
      value = 1
      call display

readloop:

      if hserinflag = 1 then
            limit = hserptr - 1
            ;if packets have been recieved, will loop until all packets
have been addressed
            for counter = 0 to limit step 3

                  ;aquired packet from scratchpad memory and loads into
vars
                  temp = counter
                  get temp, packet
                  temp = counter + 1
                  get temp, xmov
                  temp = counter + 2
                  get temp, ymov

                  ;clears pixel from display, displaying what's on virtual
display
                  column = col
                  bptr = column
                  value = @bptr
                  call display

                  ;calculates new x and y position
                  x = x - xmov
```

```
                    y = y - ymov

                    col = x/32  ;calculates new column and row
                    row = y/32
                    bptr = col
                    temp = @bptr

                    bptr = 8 + row

                    ;displays new pixel position
                    column = col
                    let value = @bptr or temp
                    call display

                    if packet bit 0 set then      ;if the left mouse button
is down
                          if clickFlag bit 0 clear then       ;if this is the
first packet the button has been pressed for
                                let clickFlag = clickFlag or 1      ;flag
that click has been addressed
                                bptr = col
                                value = @bptr     ;gets what is displayed on
the column the cursor is in
                                bptr = 8 + row

                                ;switches the value of the pixel the cursor
is on in the virtual display
                                let temp = value and @bptr
                                if temp > 0 then
                                      let temp = inv @bptr
                                      let value = value and temp
                                else
                                      let value = value or @bptr
                                endif
                                bptr = col
                                let @bptr = value
                          endif
                    else
                          let clickFlag = clickFlag and %11111110   ;resets
flag if mouse not pressed
                    endif


              next counter

      ;resets pointer and flag after all packets addressed
      hserinflag = 0
      hserptr = 0
      endif
```

2.5.3 Testing

To test this subsystem the packet-sending sequence was observed using a logic analyser. The thirteen pins involved in the sequence were connected to a probe and the PICAXE was then instructed to call the 'sendPacket' procedure with the value 'F4'. A 10kΩ pull-up resistor was connected to 'Clk' to simulate the functionality of the mouse.

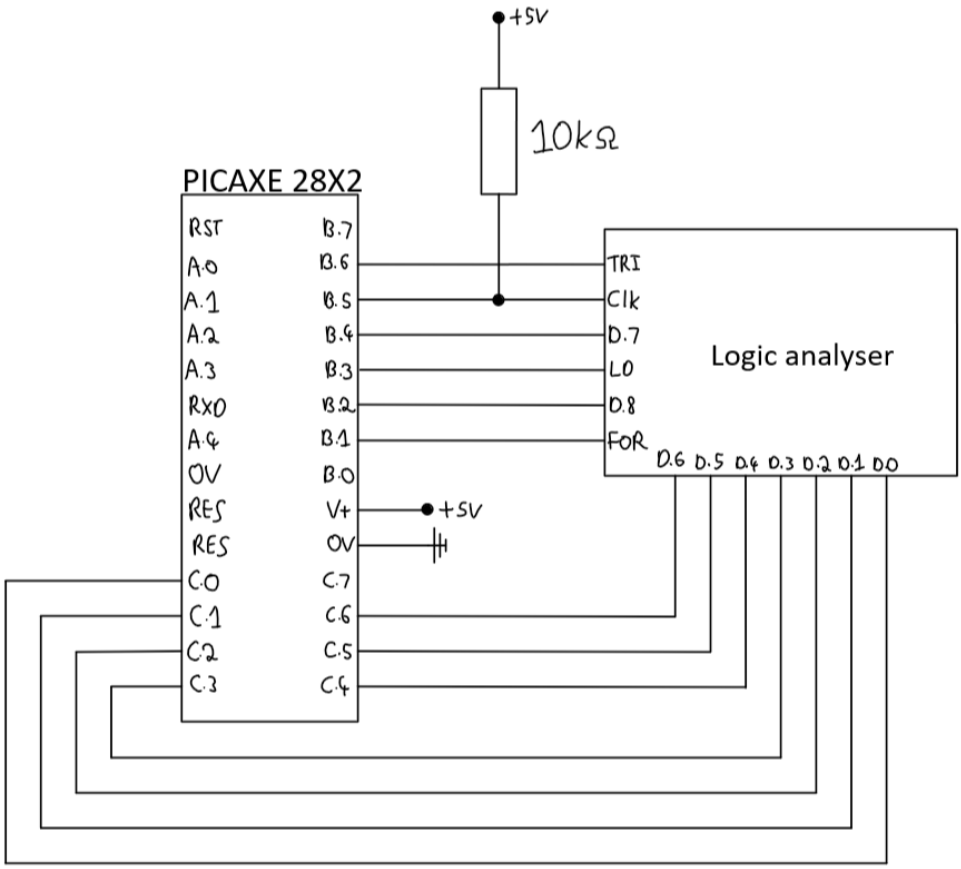

Figure 28 - Circuit diagram of test

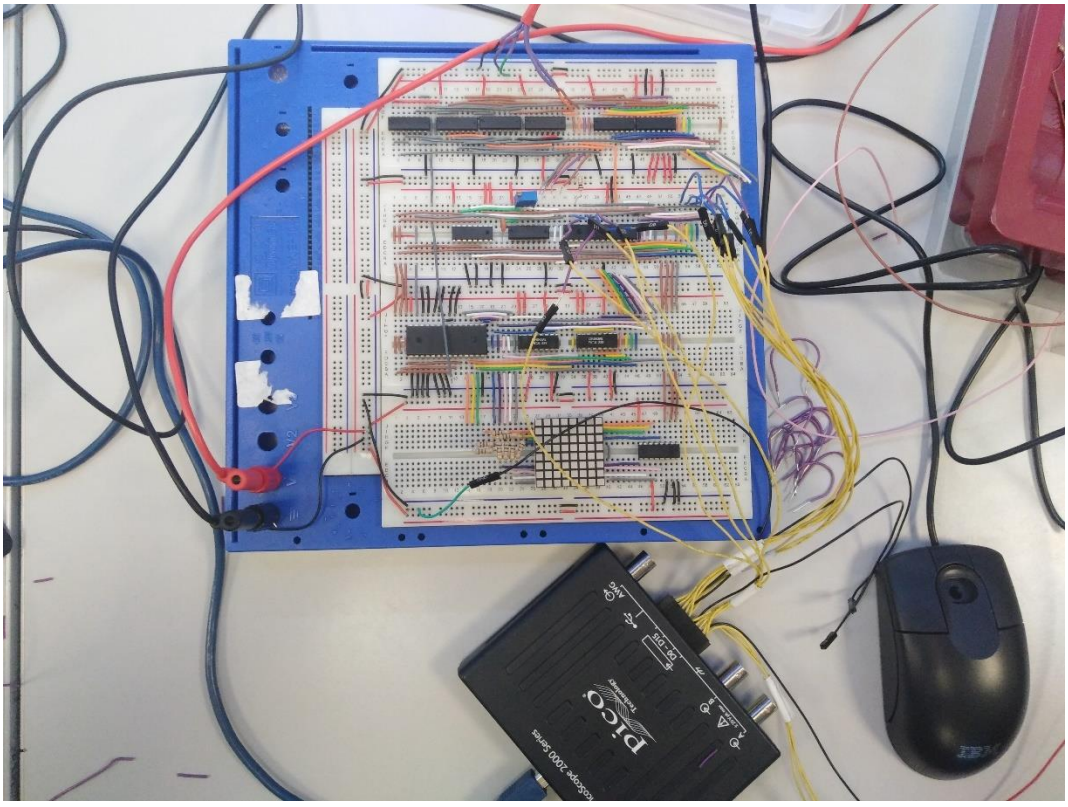

Figure 29 - Photo of test

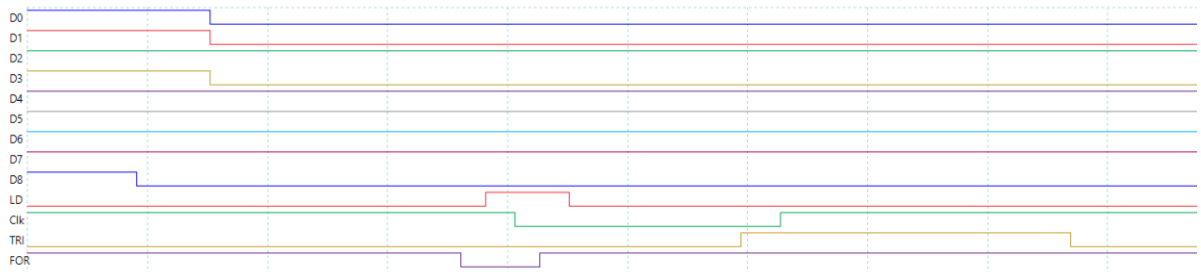

Figure 3 - Results of test. 'FOR' is the first control line mentioned in the sequence earlier, 'LD' is the second and 'TRI' is the third. 'Clk' is the clock line, $D_0 – D_7$ being the eight data bits and D_8 the parity bit.

The trace from the logic analyser shows the expected result. There is a short delay between the parity bit being set and the rest of the data outputs being set due to the time it takes for the PICAXE chip to execute the instruction in-between the two events. This is not an issue as what is important is that the correct value is in place by the time the clock line is brought low, which this delay does not effect. F4 in binary is b'11110100' so should have a parity bit of '0' since there are an odd number of 1s. From the trace it can be seen that the value loaded in is '011110100', so has functioned as intentioned. The rest of the sequence is then followed according to the steps outlined earlier.

A similar test was also performed to test the RAM writing sequence. A logic analyser was connected to the thirteen pins involved and the PICAXE was then instructed to execute the 'clearDisplay' procedure. The expected result is to see the writing cycle repeat for the address values of 0 to 7.

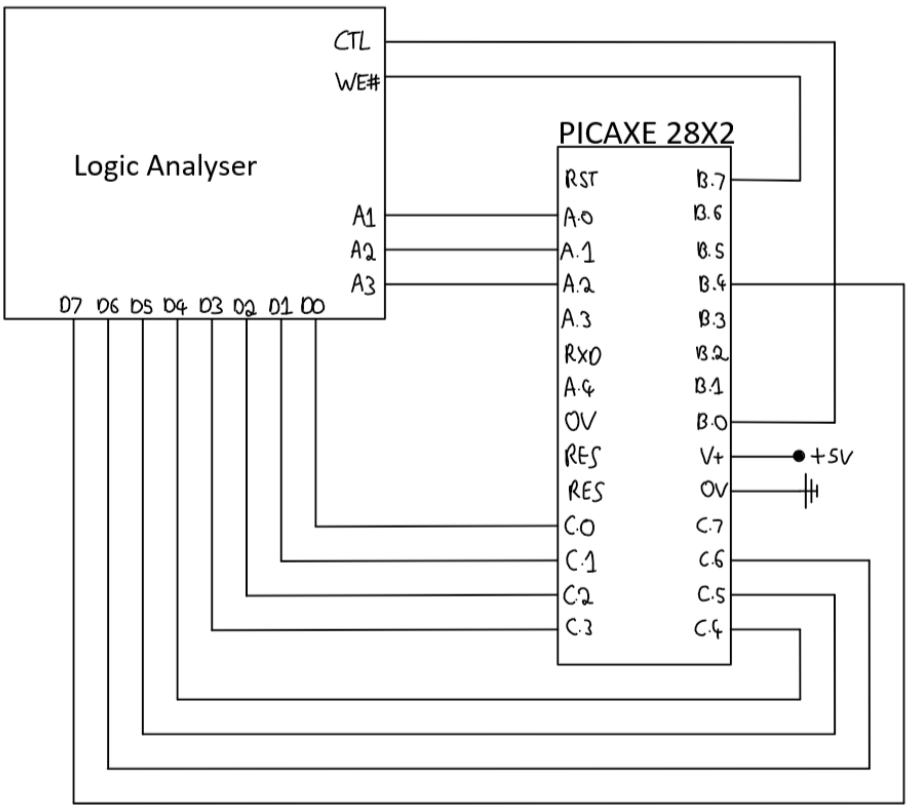

Figure 30 - Circuit diagram of test

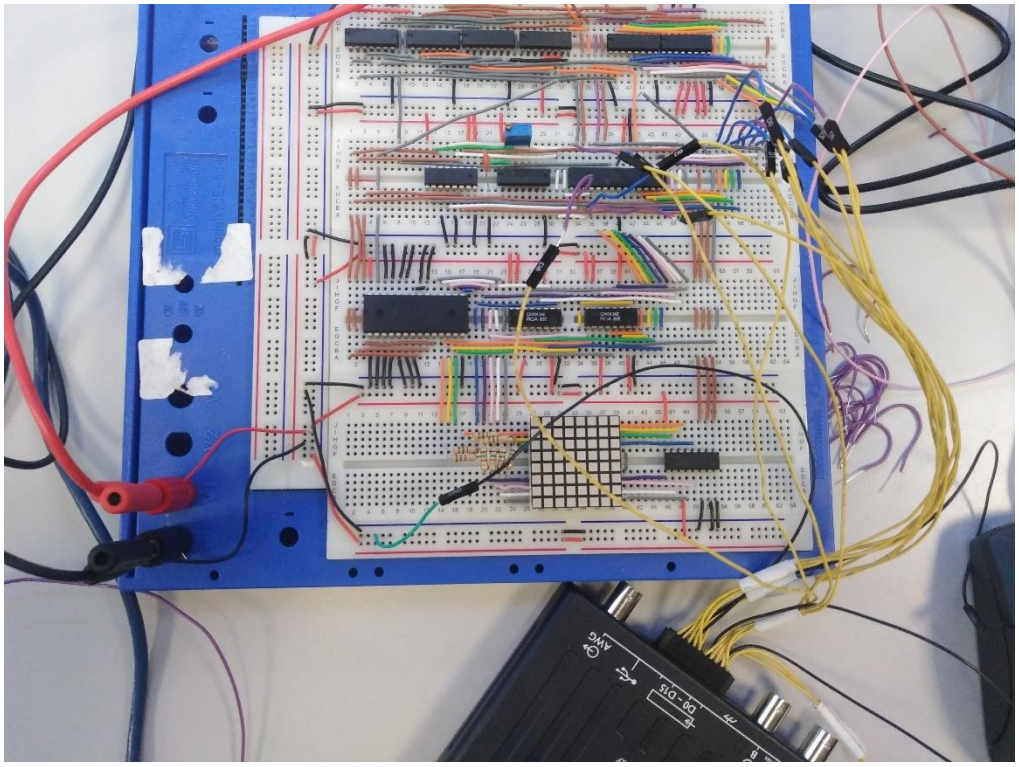

Figure 31 - Photo of test

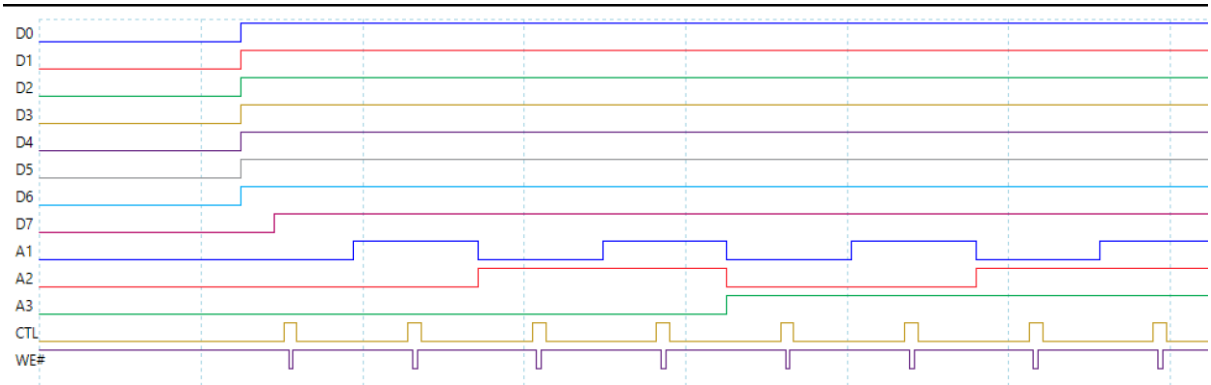

Figure 4 - Results of test. D_0 – D_7 are the data lines, A_1 – A_3 are the address lines, CTL is the first control line mentioned in the earlier sequence, WE# is the second

A_3	A_2	A_1	Denary
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Figure 32 - Sequence of address lines

The only thing to note about these results is that D_7 comes on after D_0–D_6. This is because D_7 is on a different output port from the other outputs, meaning that they can't be changed simultaneously. The correct output is still in place by the time the WE# pin brought low, so is inconsequential. Otherwise, the results are as expected with the address counter incrementing from 0 to 7 as shown in the above table. In conclusion, the circuit can execute both of the required sequences so meets the specification.

2.6 SUBSYSTEM #6 – PACKET SENDER

2.6.1 Overview

The purpose of this subsystem is to be able to receive a packet from the microcontroller, which will then be sent to the mouse in compliance with the PS/2 protocol.

Specification:

- Must be able to send PS/2 packets at any frequency between 10–16.7kHz
- Must be able to receive packets from the microcontroller

2.6.2 System Design

Packets sent to the mouse consist of twelve bits:

- A start bit
- Eight data bits
- An odd parity bit
- A stop bit
- An acknowledge bit

The instructions[6] to send a packet to the mouse are as follows:

1. Bring the Clock line low for at least 100µs
2. Bring the Data line low
3. Release the Clock line
4. Wait for the device to bring the clock line low
5. Set/reset the Data line to send the first bit
6. Wait for the device to bring Clock high
7. Wait for the device to bring Clock low
8. Repeat steps 5-7 for the other seven data bits and parity bits
9. Release the data line
10. Wait for the device to bring Data low
11. Wait for the device to bring Clock low
12. Wait for the device to release Data and Clock

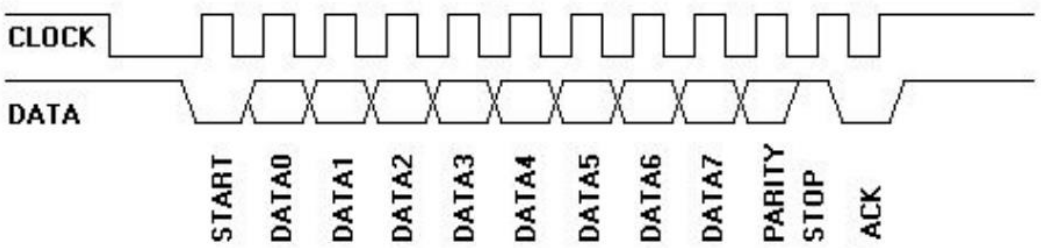

Figure 33 - Host to device communication

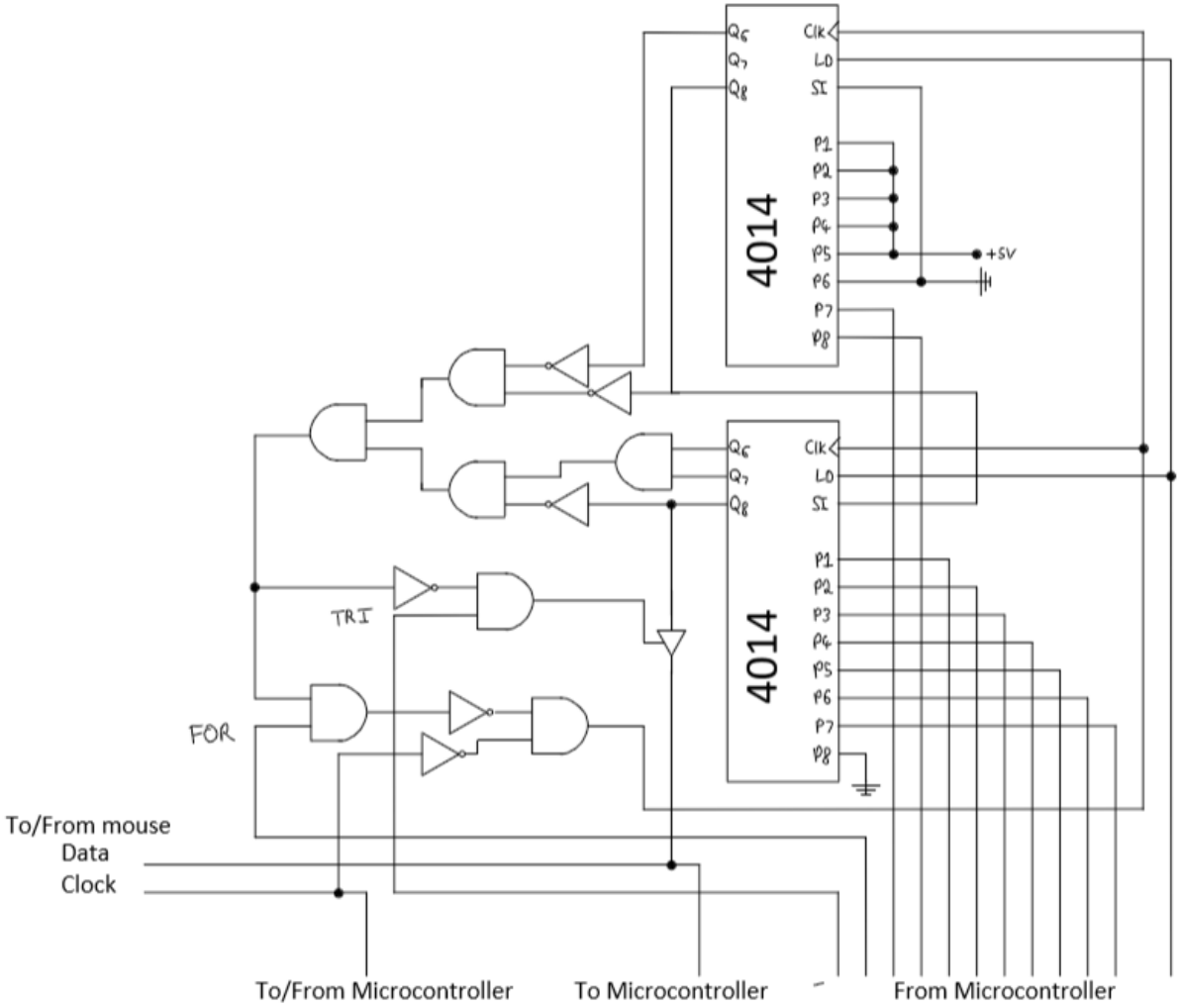

Figure 34 - Circuit diagram

To receive and send packets two eight-bit PISO shift registers are chained together. When a packet is to be sent, the following sequence occurs:

1. The microcontroller allows the inverse of the clock signal to be passed to the clock pin of the shift registers by bringing FOR low.
2. The microcontroller brings the load pin of the shift registers high, meaning they'll load in the presented value at the next rising edge of the clock pin.
3. The microcontroller pulls the clock low, loading in the value and initiating part 1 of the PS/2 sequence. The output of the shift register, Q_8, will be low as the LSB, p_8 is tied to ground.

4. The output of the shift registers changes at the newly loaded value, meaning they will receive the clock pulse without the microcontroller allowing the value to pass.
5. The microcontroller releases the load pins and FOR.
6. After waiting at least 100µs, the microcontroller will bring TRI high, activating a tristate to give the output of the shift registers control of the data line. This will pull the data line low, completing part 2 of the PS/2 protocol.
7. The microcontroller releases the clock line to complete part 3, the rising edge signalling the start bit to the mouse. This will appear as a falling edge at the clock pin of the shift registers, meaning there is no change.
8. The mouse will then generate eleven clock pulses, shifting out the rest of the packet information. On the falling edge of a pulse, the shift register will shift out the next bit in time for it to be read by the mouse on the next rising edge. This completes steps 4 to 8 of the protocol.
9. On the falling edge of the parity bit, the outputs of the shift registers will finally have reached a unique state which will cause the tri-state to be disabled and prevent further clock pulses from being seen by the registers. If the registers could still see the clock line, the acknowledgement bit would cause the registers to shift again meaning the output would no longer be a unique state. This would cause the tri-state to be activated again, which would invalidate the sent packet. This logic is needed as the microcontroller is too slow to know when the packet has ended to be able to revoke control from the packet sender. This completes step 9 of the protocol.
10. Steps 11 and 12 can be ignored, as receiving the acknowledgement is not required to send a packet.

Register	Bit	Start	1	2	3	4	5	6	7	8	Parity	Stop
Top	P_1	1	0	0	0	0	0	0	0	0	0	0
	P_2	1	1	0	0	0	0	0	0	0	0	0
	P_3	1	1	1	0	0	0	0	0	0	0	0
	P_4	1	1	1	1	0	0	0	0	0	0	0
	P_5	1	1	1	1	1	0	0	0	0	0	0
	P_6 (Q_6)	0	1	1	1	1	1	0	0	0	0	0
	P_7 (Q_7)	-	0	1	1	1	1	1	0	0	0	0
	P_8 (Q_8)	1	-	0	1	1	1	1	1	0	0	0
Bottom	P_1	1	1	-	0	1	1	1	1	1	0	0
	P_2	1	1	1	-	0	1	1	1	1	1	0
	P_3	-	1	1	1	-	0	1	1	1	1	1
	P_4	-	-	1	1	1	-	0	1	1	1	1
	P_5	-	-	-	1	1	1	-	0	1	1	1
	P_6 (Q_6)	-	-	-	-	1	1	1	-	0	1	1
	P_7 (Q_7)	-	-	-	-	-	1	1	1	-	0	1
	P_8 (Q_8)	0	-	-	-	-	-	1	1	1	-	0

Figure 35 - Table showing packet being shifted through. '-' is used to denote an unknown value

The table above shows a packet being shifted through the registers. All commands which can be sent to the mouse being with either 0xFF or 0xFE, meaning the bits marked in orange will always be '1'. The cells marked in blue are always '1' as these pins are connected to +5V. A '0' is loaded in to the top after each shift as the serial input of the top register is connected to ground. A cell is marked in green when that output either is or could be the value being looked for. It is only when all of the five looked at outputs are green that the tristate is deactivated, releasing the data line. As can be seen from the graph, it is only on the falling edge of the parity bit that all five observed outputs are green, no matter which packet is sent. This ensures that the data line is always released at the correct moment – at the falling edge of the parity bit.

2.6.3 Testing

To test this subsystem, the values of an 0xF4 packet were manually loaded into the shift registers by following the steps outlined earlier. A 10kΩ pull-up resistor was attached to the data line and a signal generator was set to produce a 16.7kHz square wave to simulate the mouse. The clock and data line were then observed with a picoscope. The expected result is to see the value b'0001011110' output the moment the load pins are brought low, followed by the releasing of the clock line.

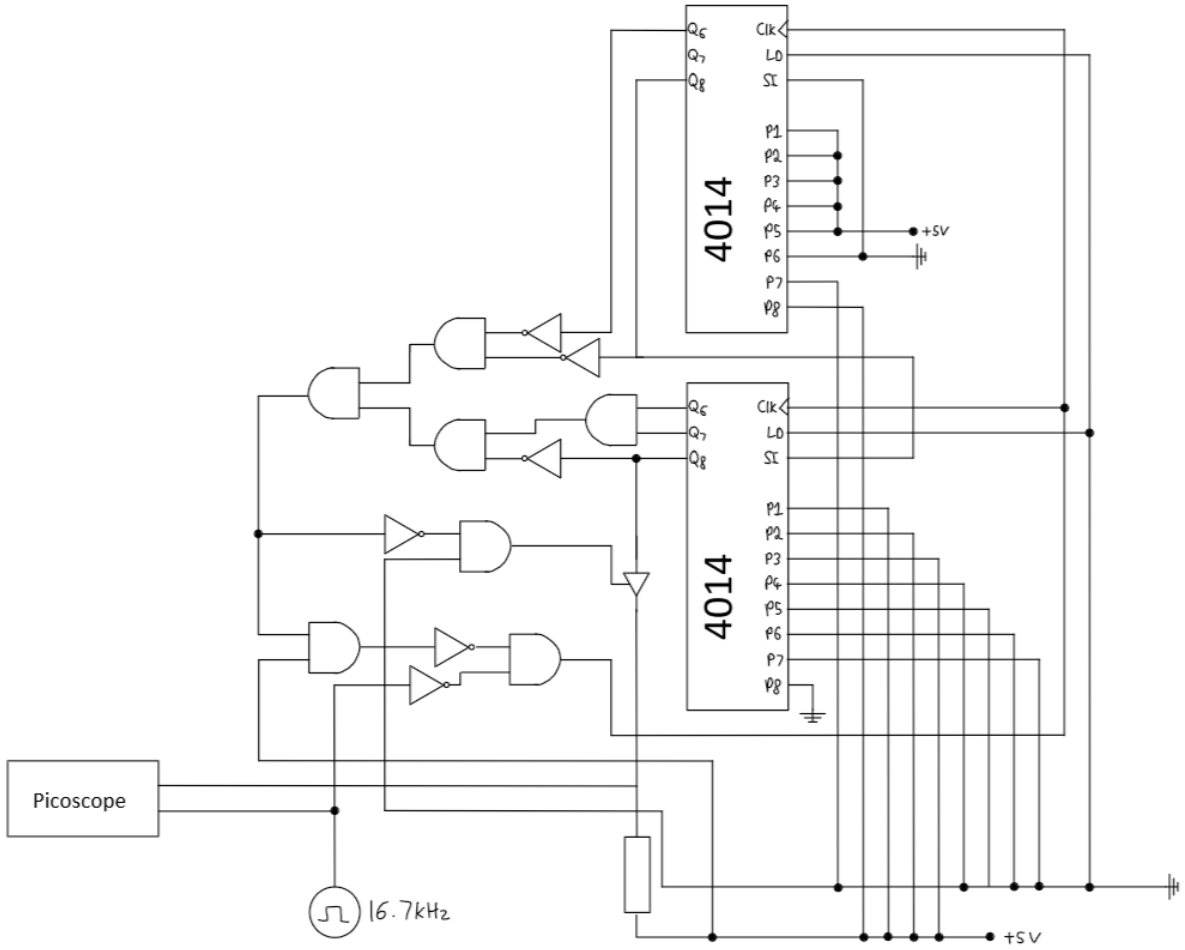

Figure 36 - Circuit diagram of test

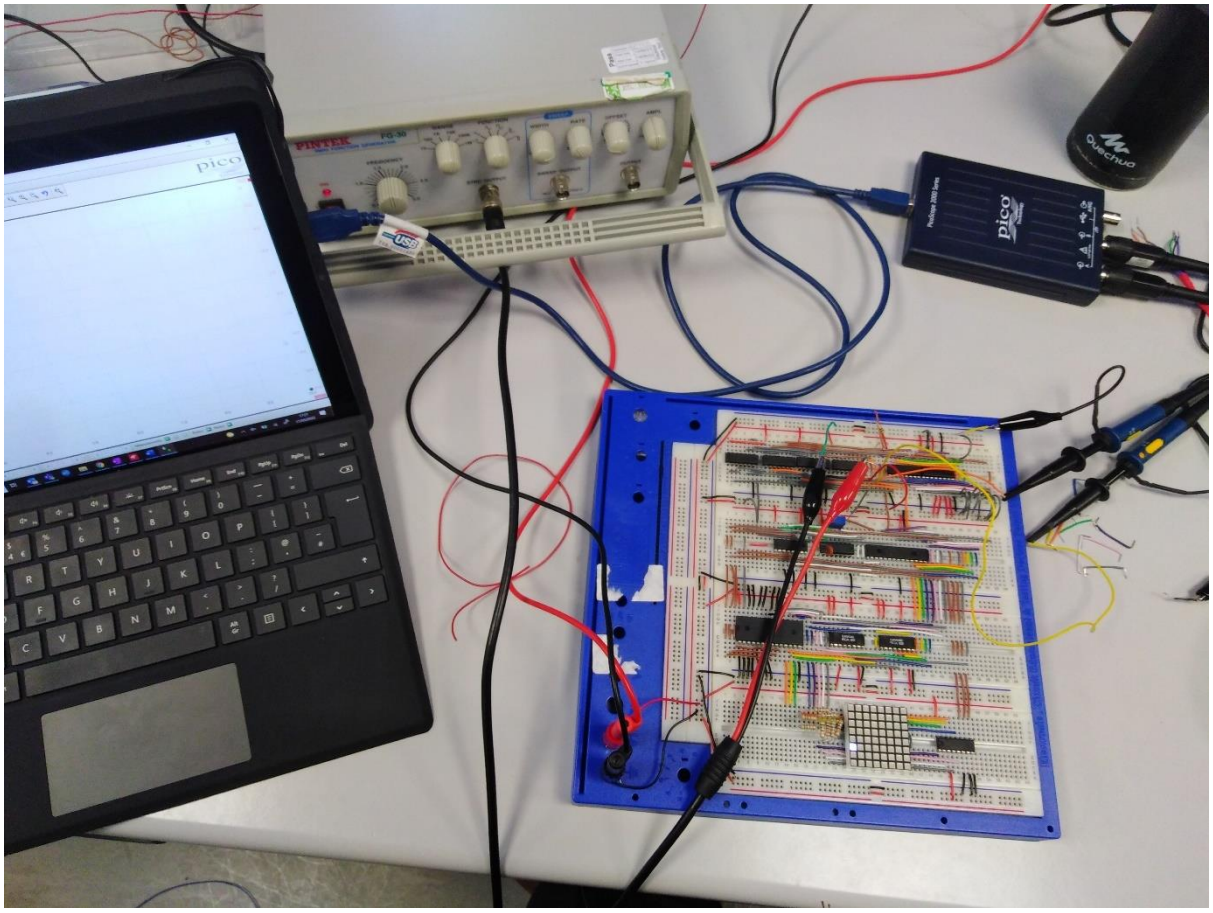

Figure 37 - Picture of test

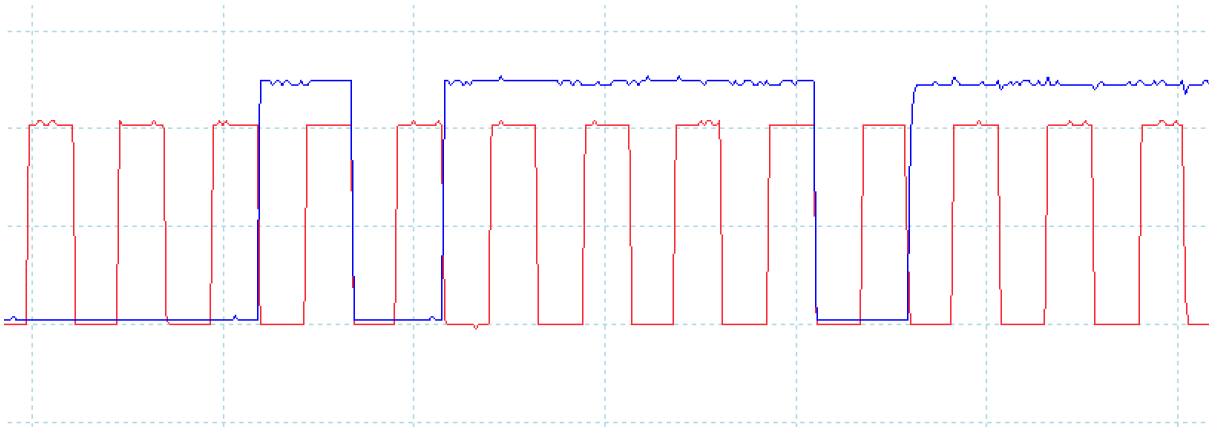

Figure 5 - Results of test. Red trace is the clock line, blue trace is the data line

The trace shows the expected result, with the data line being released at the end of the 10th clock pulse. This shows that a packet was successfully loaded into the shift registers and was then outputted according to the PS/2 protocol at the upper frequency limit of 16.7kHz. This subsystem, therefore, meets both specification points.

2.6.4 Alternate Design

Another way this subsystem could be built would be to use NAND equivalents for the logic gates.

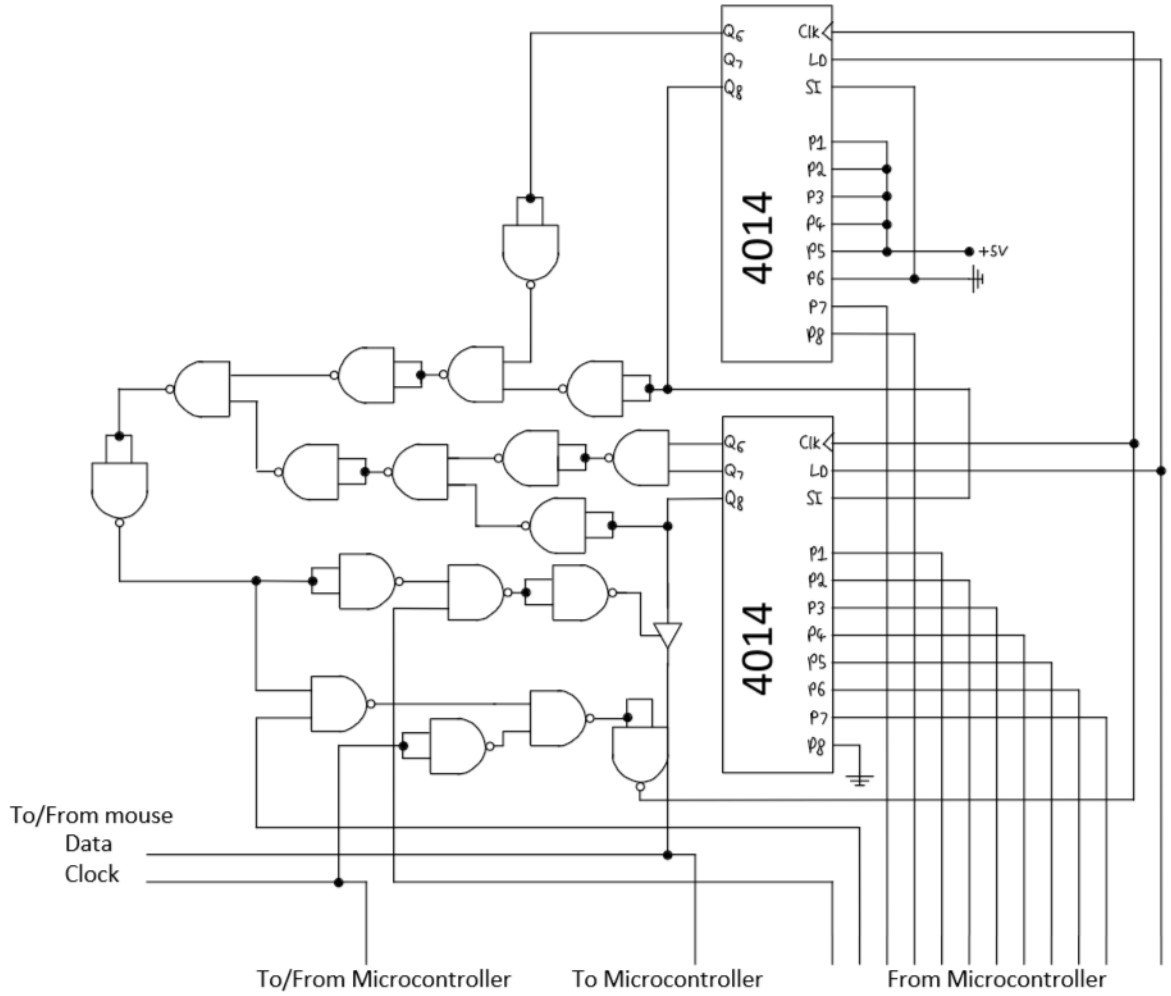

Figure 38 - Alternate circuit design

NAND equivalents can make building logic simpler, however in this case it has only served to complicate it. The outcome of this design will be the same and will still work, however it would require five IC chips instead of the three needed in the original design. This would result in more wiring and more space used on the board for the same outcome, which is why the original design was used instead.

3 SYSTEM REALISATION

3.1.1 Block Diagram

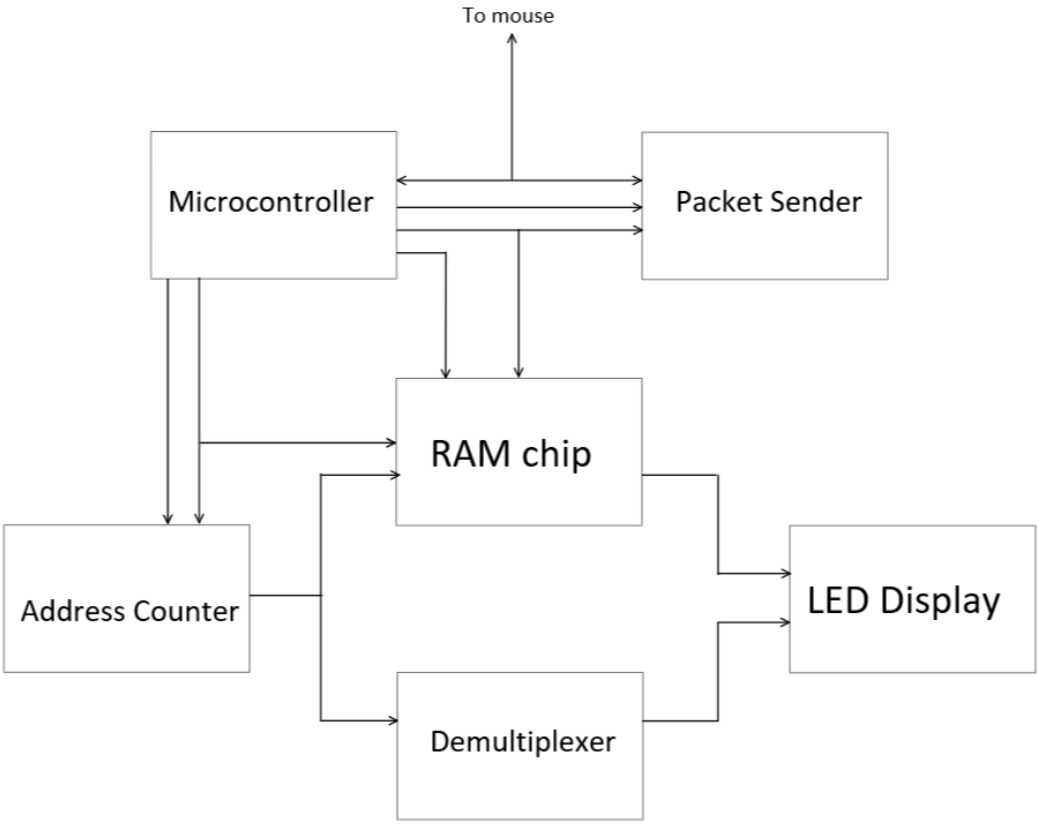

Figure 39 - Block diagram of full system

3.1.2 Circuit Diagram

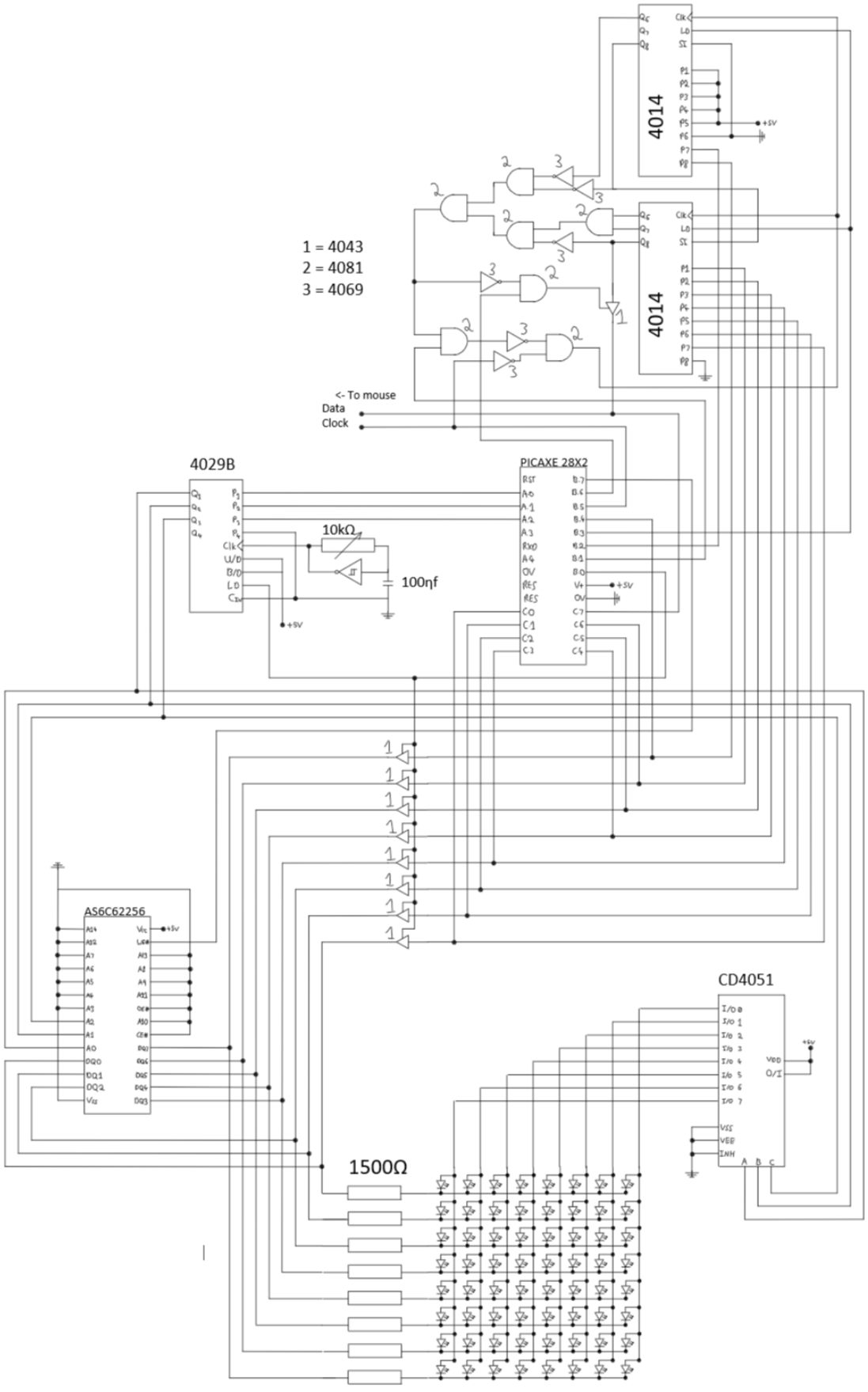

Figure 40 - Full system diagram

3.1.3 Picture

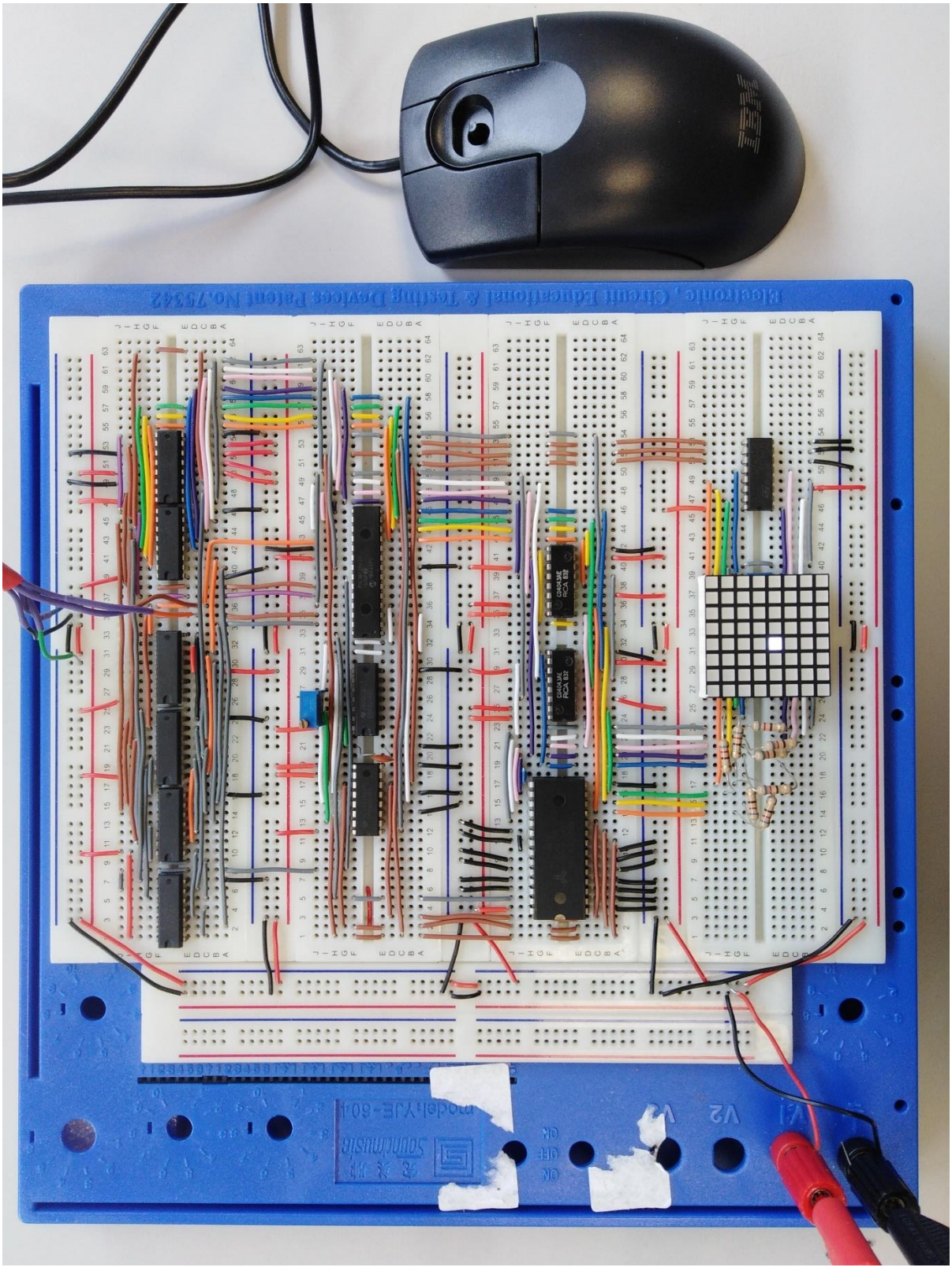

Figure 41 - Picture of circuit

3.1.4 Testing

3.1.4.1.1 Test 1

The system was first tested by trying to draw three objects on the screen using the mouse: a smiley face, a target and a house. The results were then photographed using a phone camera.

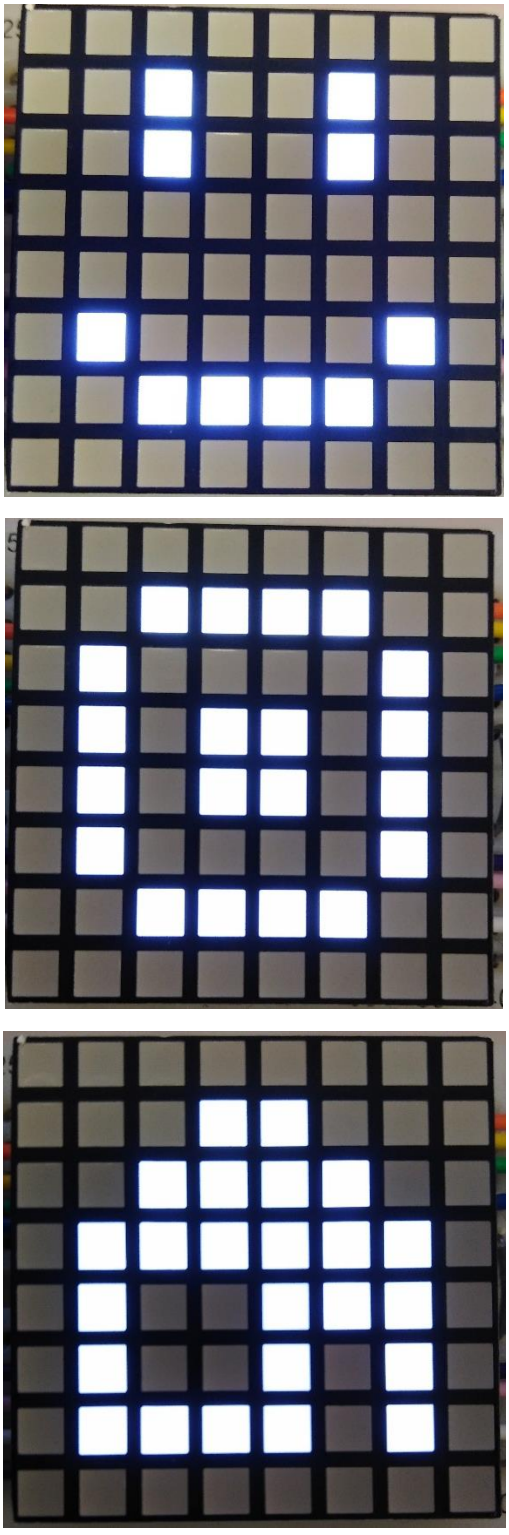

Figure 42 – Results of test

As can be seen from the photographs, all three objects were successfully drawn. The mouse was responsive, with the sensitivity being low enough that individual pixels could be easily selected but not so low that it was frustrating. The cursor followed the movement of the mouse and left clicking the mouse toggled the state of the pixel below the cursor, as per the specification. This test has also shown that pixel art can be drawn on the display, so the system meets the specification here too.

3.1.4.1.2 Test 2

This test was designed to see whether the system meets the third qualitative specification point. The cursor was first positioned on the left-hand side of the screen and a before photo was taken with a phone camera. The left mouse button was then held down and the cursor was to the right-hand side of the screen. Finally, the left mouse button released and an after photo was taken.

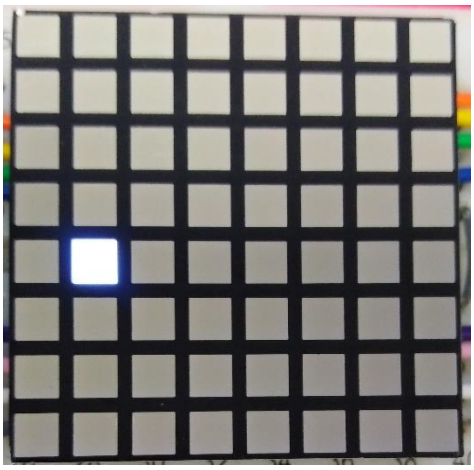

Figure 43 - Before image. The single pixel which is on is the mouse cursor

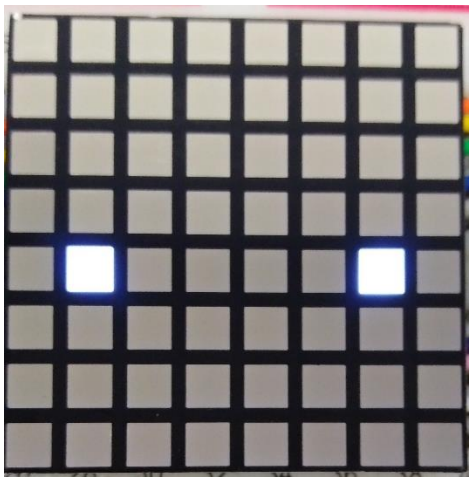

Figure 44 - After image. Pixel on the right is the cursor, pixel on the left is a drawn pixel

These images show that the system meets the specification point. If holding down left click changed multiple pixels, there would be a line of illuminated pixels between the two pixels visible in the

second image. The pixel on the left in the second image shows that when the mouse was held down, the original pressing of the button changed the state of the pixel.

3.1.4.1.3 Test 3

The purpose of this test was to find the refresh rate of the monitor. This was done by connecting the eight outputs of the demultiplexer to a logic analyser. The results should show the eight columns being cycled through repeatedly with one column being on at a given time. The time period between two consecutive occurrences of the same column being on is the refresh rate of the display.

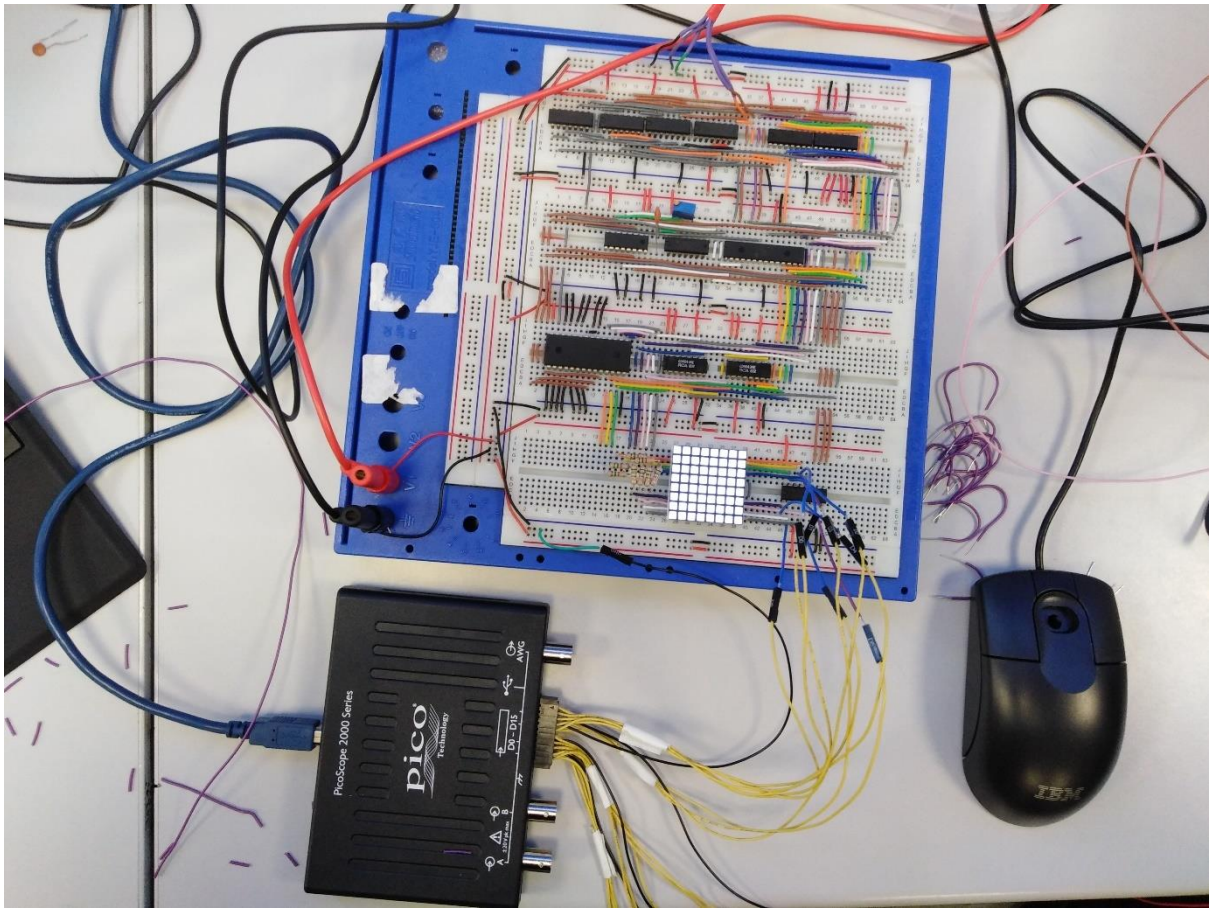

Figure 45 - Photo of test

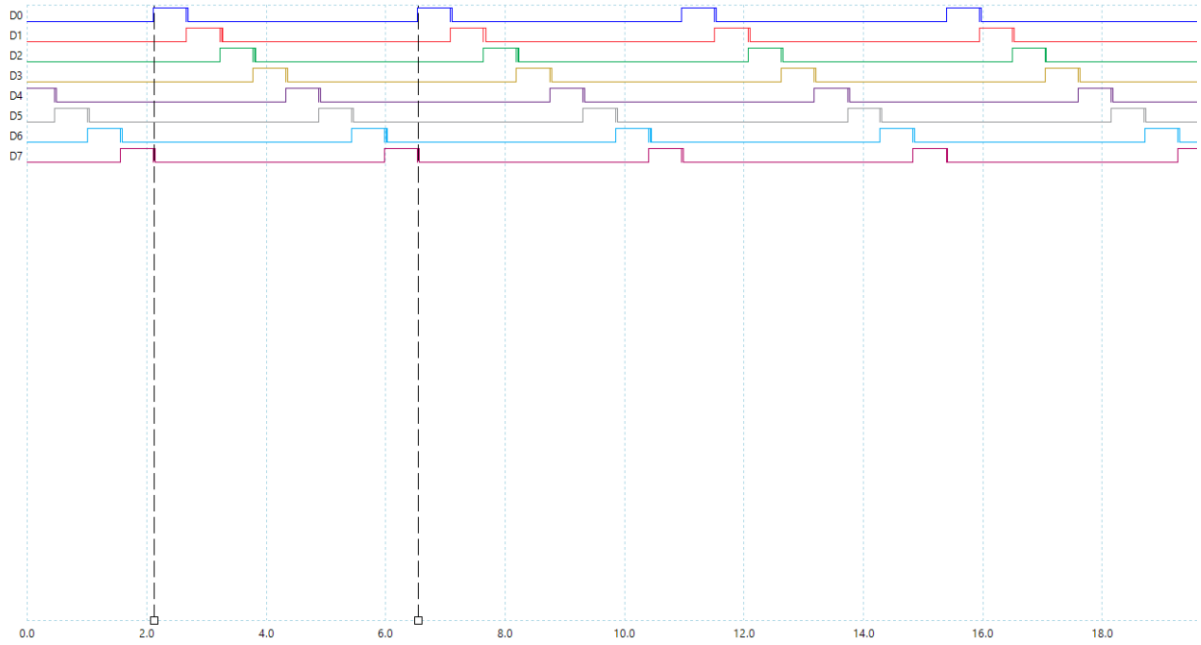

Figure 6 - Trace of results. D_0 – D_7 are columns 1-8 respectively, column 1 being the leftmost column

The trace shows the expected results, with the time period between the two points marked on the graph equalling 4.43ms (3s.f.). This means a refresh rate of:

$$f = \frac{1}{T}$$

$$f = \frac{1}{4.4274 \times 10^{-3}}$$

$$f = 226\text{Hz (3s.f.)}$$

This is well above the required 50Hz, so the system meets the specification here.

The graph also shows that there is some overlapping time at a column switch where two columns are on simultaneously. This overlap only lasts for 35µs so has no visible effect on the display.

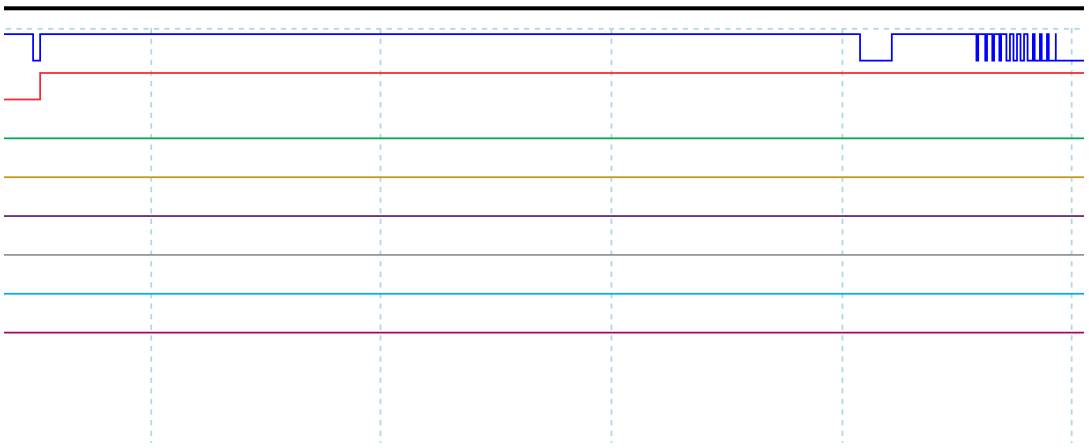

Figure 46 - Trace showing overlapping of columns more clearly

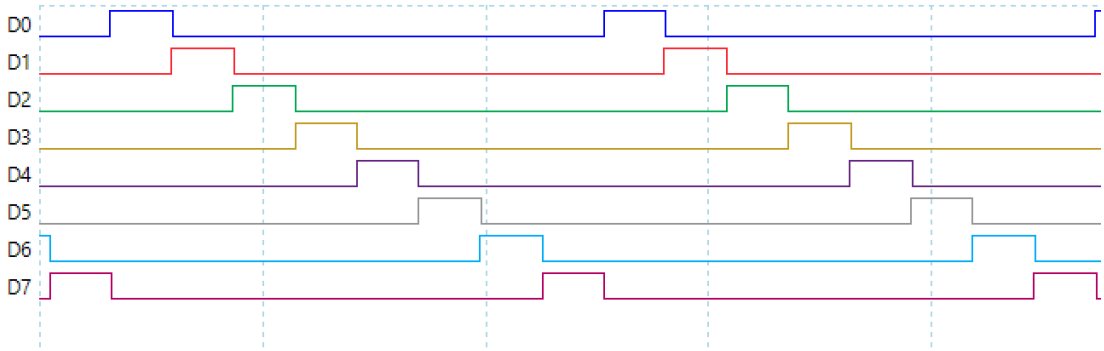

Figure 47 - The same test without the LED screen

There are also some fluctuations in the logic levels at these points which disappear when the LED screen has been removed. The cause for this is likely to be the LEDs requiring more current than expected and the demultiplexer not being able to supply enough. When a second column turns on, the current draw from the demultiplexer is higher than before, causing a large voltage drop across the internal resistance of the demultiplexer. This then means that the output voltage is near the logical threshold of the picoscope, causing the logic level to spike up and down. This should have no visible effect on the display since it only occurs for such a small time, so it is inconsequential.

3.1.4.1.4 Test 4

To test whether the system could send packets to the mouse, the data line and clock line were connected to a picoscope. The system was then turned on to observe the 0xF4 packet which is sent to the mouse by the system at start up. If the mouse successfully receives the packet, it will respond with an acknowledgement, 0xFA.

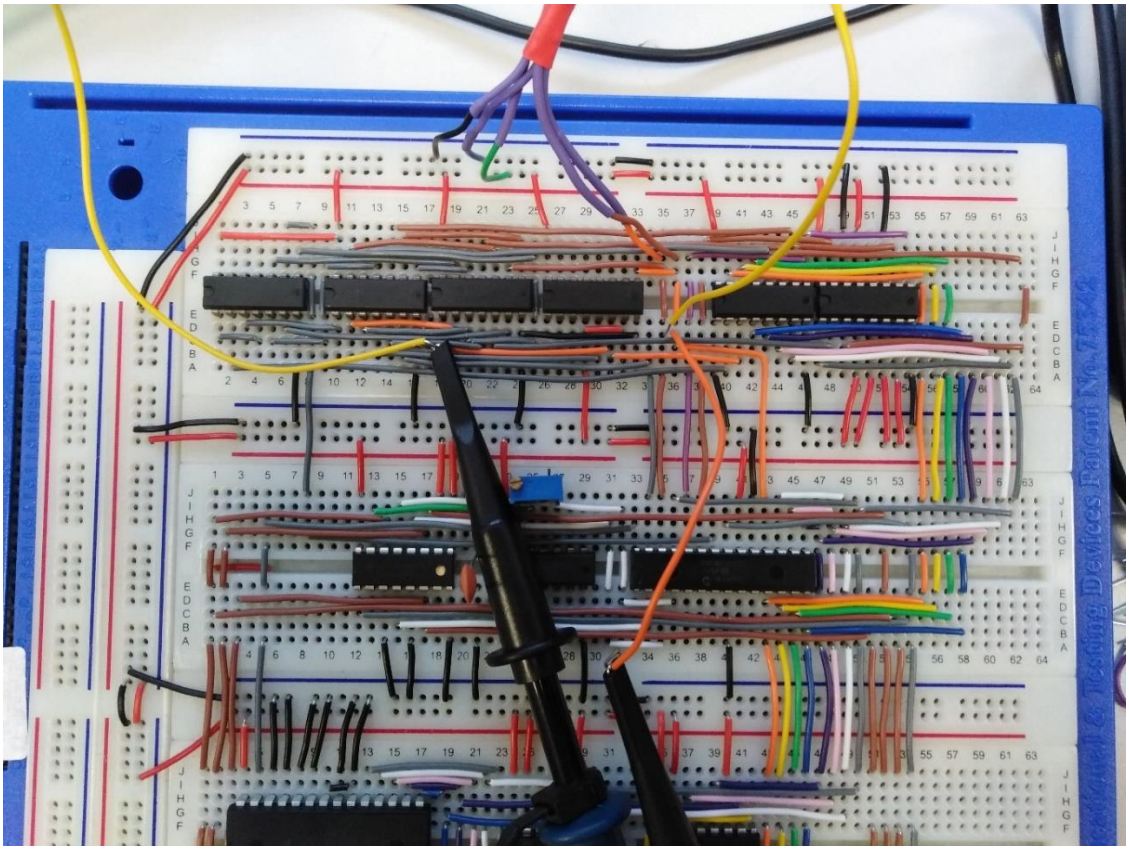

Figure 48 - picture of test

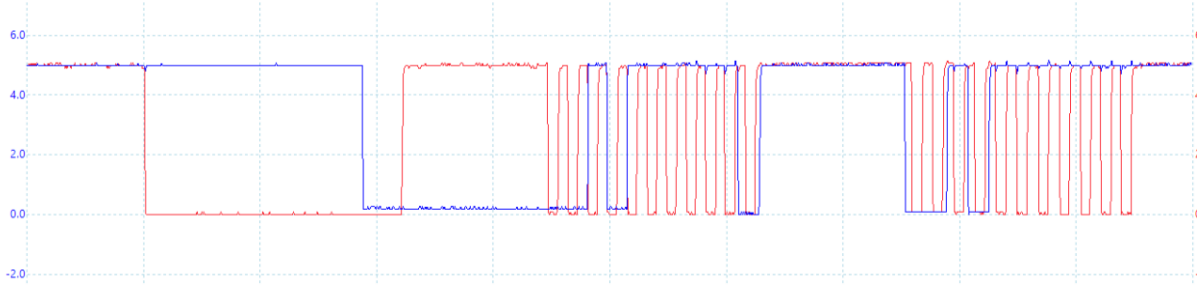

Figure 49 - Blue line is the data line, red trace is the clock line

As can be seen from the graph, the packet is sent to the mouse before the mouse responds after a short pause. This response from the mouse is:

Bit	Stop	Parity	8	7	6	5	4	3	2	1	Start
Value	1	1	1	1	1	1	1	0	1	0	0

This is equal to 0xFA, which is the mouse acknowledge command. This shows that the packet was sent successfully and therefore the system passed the specification. The time period of the clock was 80.1µs, meaning the frequency of this particular mouse is 12500Hz (3s.f). This is consistent with the frequency range of PS/2 mice of 10-16.7kHz

3.1.4.1.5 Test 5

The purpose of this test was to measure the input latency. When the system is first turned on, the mouse starts in a position (0,0), which is the top right corner. This means even the smallest upward movement of the mouse will cause the pixel to move from the top of the screen to the bottom. When the cursor moves to the bottom row, the voltage on the eighth row will drop to 0V the moment the pixel turns on. Therefore, the time between the start of the first movement packet sent by the mouse and the voltage at the eighth row dropping, plus the maximum possible input latency of the mouse, will give the input latency of the entire system.

To perform this test a logic analyser was connected to the data and clock lines, as well as to the bottom row of the LED matrix. The system was then turned on, and after a few seconds the mouse was moved upwards.

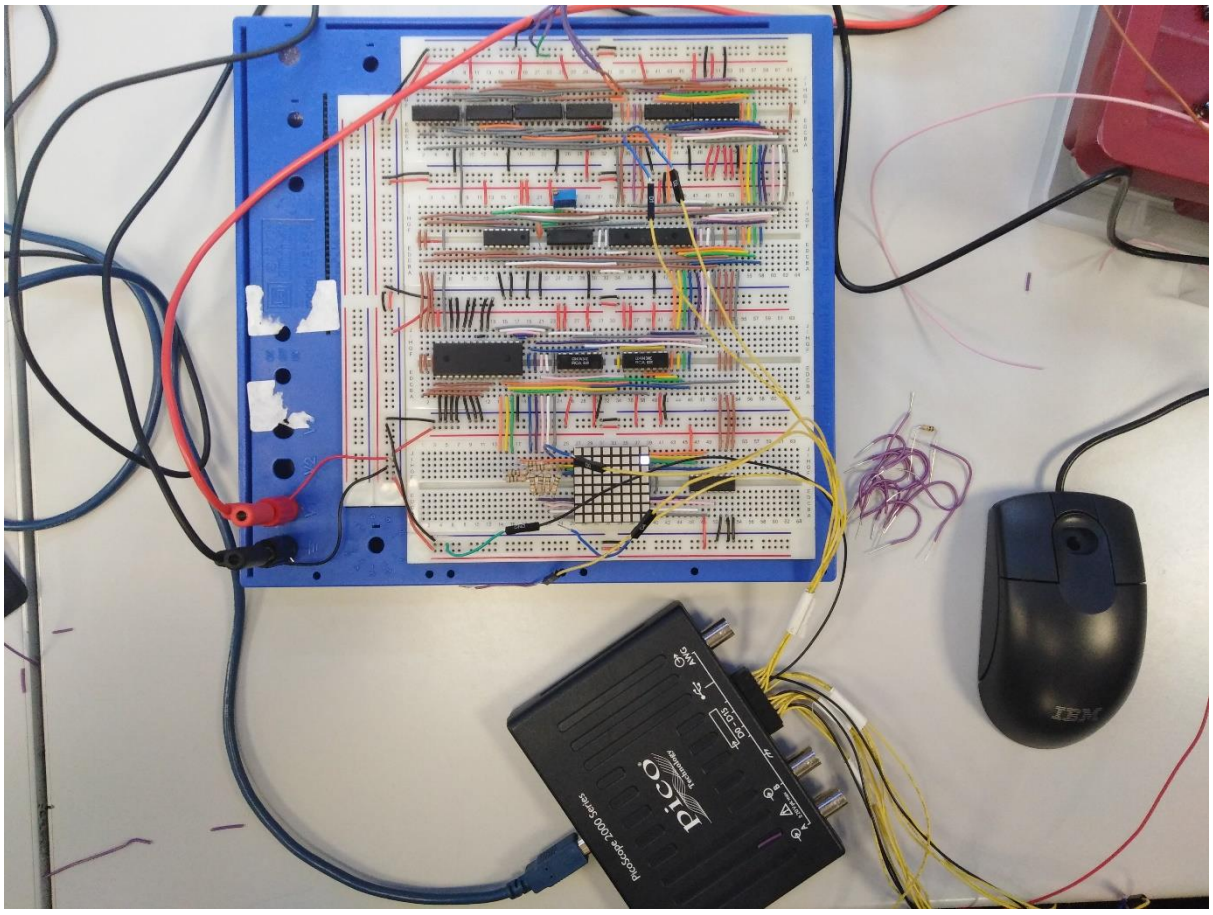

Figure 50 - picture of test

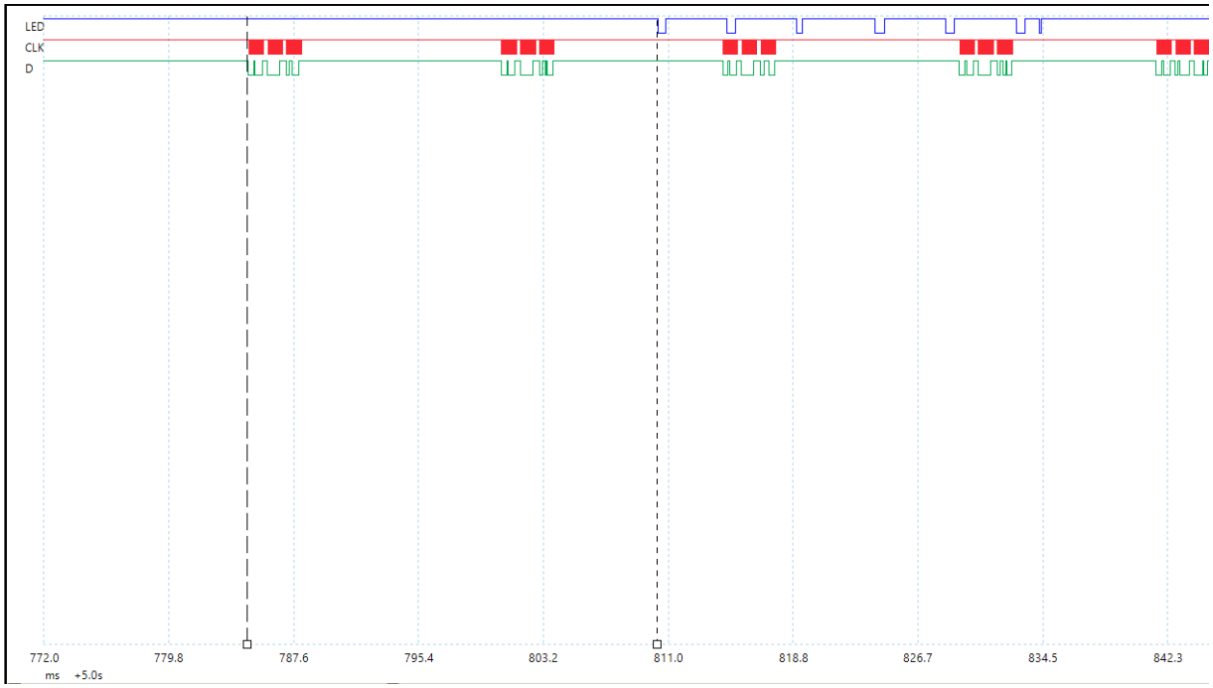

Figure 51 - Result of test

The start of the first packet and the moment the bottom LED turn on have been marked on the trace - the time period between these two points is 25.6ms. By default, the mouse will send packets when required at a maximum frequency of 100Hz. This means that if an input is made the moment after a packet is sent, the max time before that information is sent to the system by the mouse is 10ms. This means the maximum latency is 0.0356s which is below the 0.1s max latency. The system has therefore met the specification.

3.1.4.1.6 Test 6

This test was designed to see if the LED screen was visible from at least 5m away. To do this, two of my classmates and one of my teachers stood 5m away from the screen, the distance having been measured using a meter ruler. Next, a random image was drawn upon the screen which they were then asked to copy down. If the screen is visible from 5m, the image should be copied down accurately by all three people.

Figure 52 - Picture of test

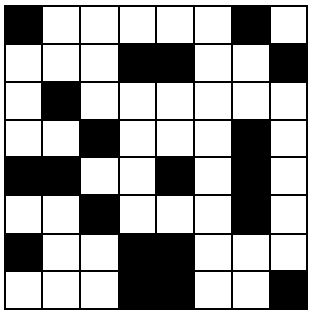

Figure 53 - First image displayed. Black squares indicate on LEDs

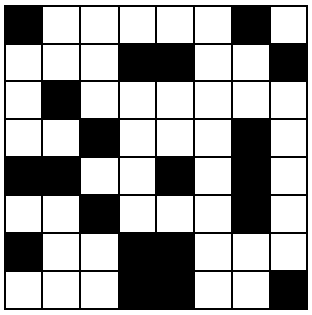

Figure 54 - Image drawn by first person

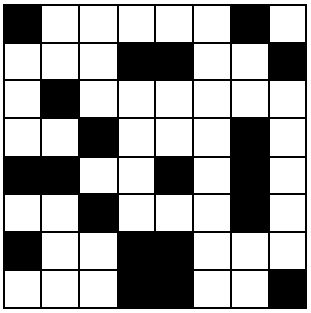

Figure 55 - Image drawn by second person

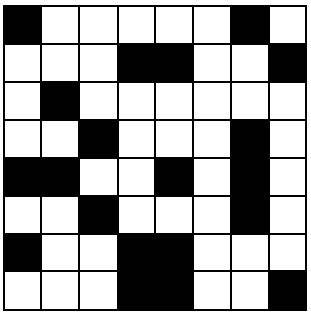

Figure 56 - Imagine drawn by third person

The results show that the image was copied down correctly on every occasion, showing that the screen is visible from 5m away and the system meets the specification.

3.1.5 Interfacing

An interfacing issue which had to be solved was how to get the system to communicate with the mouse. The original plan had been to use the microcontroller to directly communicate with the mouse for both sending and receiving packets, however the PICAXE chip that was used could only get up to speeds of around 4kHz, which was not enough to meet the required 16.7kHz. The PICAXE 28X2 has a clock speed of up to 16MHz, however it runs an interpreter rather than compiling code, which is why the final speed is so slow even when code is well optimised. This problem led to the design and creation of the packet sender subsystem to act as an intermediate step so a packet could be sent fast enough.

For reading a packet the solution was to use the 'hserin' command. This command can be used to read serial data following a specific format (one start, eight data, one stop). This does not match up with the format sent by the mouse as the mouse also has a parity bit, however this can just be ignored. This is why the 'hserout' command could not be used for sending data as it follows the same format as 'hserin' however the parity bit cannot be easily added.

Reading a packet using hserout does not use the PS/2 clock line and only listen to the data line, however it can still read the data as long as the baud rate is matched up with the frequency of the clock line.

To use 'hserin' the 'hsersetup' command must first be issued, along with a value to specify the baud rate. From the PICAXE website[7], the formula for calculating the value is as follows:

$$n = \frac{oscillator\ frequency}{4 \times baud\ rate} - 1$$

As the PICAXE 28X2 was ran at 16MHz and the desired baud rate is 12500Hz (from earlier when the frequency of the clock pulse generated by the mouse was measured), this meant that the required value was:

$$n = \frac{16000000}{4 * 12500} - 1$$

$$n = 319$$

This is why 319 was passed to the 'hsersetup' command in the code shown earlier. The 'hserin' command can only be used on the serial-in pin, which for the PICAXE 28X2 is C.7. The eight data lines going to the RAM chip and packet sender therefore had to be split across two different ports, which also made creating the program harder.

3.1.6 User Guide

When the system is first started up, a pixel will appear in the top right corner after a few seconds – this is the cursor. The cursor will follow the motion of the mouse. To draw, move the cursor over a pixel and press the left mouse button to toggle the pixel between being on and off. Note that this change will not be immediately visible as the cursor will still be over the pixel, so the cursor must first be moved for this change to be visible.

3.1.7 Component List

Quantity	Component
1	IBM MO098k mechanical ball mouse
1	4069 IC chip (hex NOT gate)
2	4081 IC chip (quad AND gate)
2	4014 IC chip (8bit shift register)
3	4043 IC chip (quad tri state)
1	PICAXE 28X2 microcontroller
1	4029B IC chip (4-bit binary counter)
1	10kΩ potentiometer
1	100ηF
1	40106B IC chip (hex inverting Schmitt trigger)
1	AS6C62256 (8-bit SRAM)
8	820Ω resistor
1	KEM-12288-AW (8x8 LED matrix)
1	4051 IC chip (8-bit multiplexer/demultiplexer)

3.1.8 Risk Assessment

During the building process of this project, wire cutters were used cautiously. Wire cutters are sharp and could cause bodily harm if used carelessly. There were also power supplies in the lab capable of producing voltages greater than 20V, which could deliver a painful shock - especially to anyone working with wet hands when the resistance of skin can be as low as 1000Ω.

4 EVALUATION

Overall, this project is a success and achieved its aim - to be able to draw pixel art with a computer mouse. All of the points laid out in the earlier specification were met:

- The refresh rate of 226Hz is faster than the required 50Hz, meaning that the display looks smooth
- Packets can be sent at any frequency within the specified range with the particular mouse used operating at 12500Hz
- The latency of 0.0356s is lower than the maximum 0.1s, ensuring the system feels responsive
- The display is visible from 5m away, demonstrated by the fact that three people could copy the display down accurately at that distance. This is despite the issue with the LED display not receiving the 10mA of current that it is supposed to
- There is a cursor which follows the movement of the mouse, represented by a single glowing LED
- Clicking the left button of the mouse changes the state of the pixel under it
- Holding down the left click button only changes the state of the initial pixel the cursor was over
- A smiley, a target and a house were drawn, demonstrating that pixel art can be drawn

Each of the six subsystems performed their intended function and interacted with each other as intended:

- The microcontroller can successfully read data sent by the mouse and can send a packet to the packet sender, as well as execute the required sequence to get the packet sender to send said packet to the mouse
- The packet sender successfully sends this packet to the mouse, meaning overall the system can interface fully with the mouse. The microcontroller successfully processes the data sent by the mouse, evident from the cursor following the mouse and pixel art being able to be drawn
- The address counter counts from 0 to 7 repeatedly as intended, cycling through the addresses on the RAM chip and changing which column is selected by the demultiplexer. The counter is also able to have a value loaded into it by the microcontroller, and the microcontroller is able to perform the series of outputs required to do so - there would be no moving cursor if the microcontroller couldn't do this
- The RAM chip can have values written to it by the microcontroller, which it will then display to the LED display. It also acts as a barrier between the microcontroller and LED display, allowing the microcontroller to send packets to the packet sender without interfering with the display
- The demultiplexer successfully takes the value from the address counter and uses it to turn on a single column
- The LED display takes the information that is to be output from the RAM chip and demultiplexer, displaying it to the user

There are still issues with the circuit however, and there are things which could be improved. The demultiplexer not supplying enough current causes two issues - the LEDs were dimmer than they should have been and the brightness of LEDs in a column will change depending on how many LEDs are on in that column. Three things could improve this:

1. Increasing the input voltage to the demultiplexer to 10.92V
2. Increasing the resistance of the protective resisters in the LED display to 2000Ω
3. Adding transducer drivers between the demultiplexer and LED display

This would result in a more consistent and brighter display.

The second issue is with how the cursor is represented. The cursor is a single solid pixel, meaning it can be easy to lose if there are other pixels which have been turned on. It also means the state of the pixel underneath the cursor can't be seen, so a user does not know the state of the pixel. To solve these problems, the cursor could instead be represented by a pixel flashing on and off, so it can be differentiated from other pixels. The other issue could be solved by adding an additional LED separate to the display which shows the state of the pixel beneath the cursor. This would allow a user to know the state of the pixel even if they can't see it on the display.

The third issue is that the display is quite small. Only very simplistic pixel art can be drawn on the 8x8 display, so a larger display would allow for more complex images to be drawn. However, this would require significant changes to the circuit to allow for the large number of bits. For example, a large demultiplexer would be required, as well as a RAM chip with wider addresses.

Despite the aforementioned flaws and areas for improvement mentioned above, the system as a whole is both functional and reliable, demonstrating that it has achieved the initial design specification and goal.

5 BIBLIOGRAPHY

[1] Source for flicker fusion threshold: https://www.ccohs.ca/oshanswers/ergonomics/lighting_flicker.html

[2] Source for responsiveness times: https://www.nngroup.com/articles/response-times-3-important-limits/

[3] Datasheet for LED matrix: https://cdn.shopify.com/s/files/1/0174/1800/files/KEM-12288-AW.pdf?7106087606921464074

[4] Datasheet for AS6C62256: https://www.alliancememory.com/wp-content/uploads/pdf/AS6C62256.pdf

[5] Datasheet for 4051: https://global.oup.com/us/companion.websites/fdscontent/uscompanion/us/pdf/microcircuits/students/logic/cd4051-ti.pdf

[6] Instruction for PS/2 mouse protocol: https://www.avrfreaks.net/sites/default/files/PS2%20Keyboard.pdf

[7] Hsersetup command for PICAXE chip: https://picaxe.com/basic-commands/serial-rs232-interfacing/hsersetup/