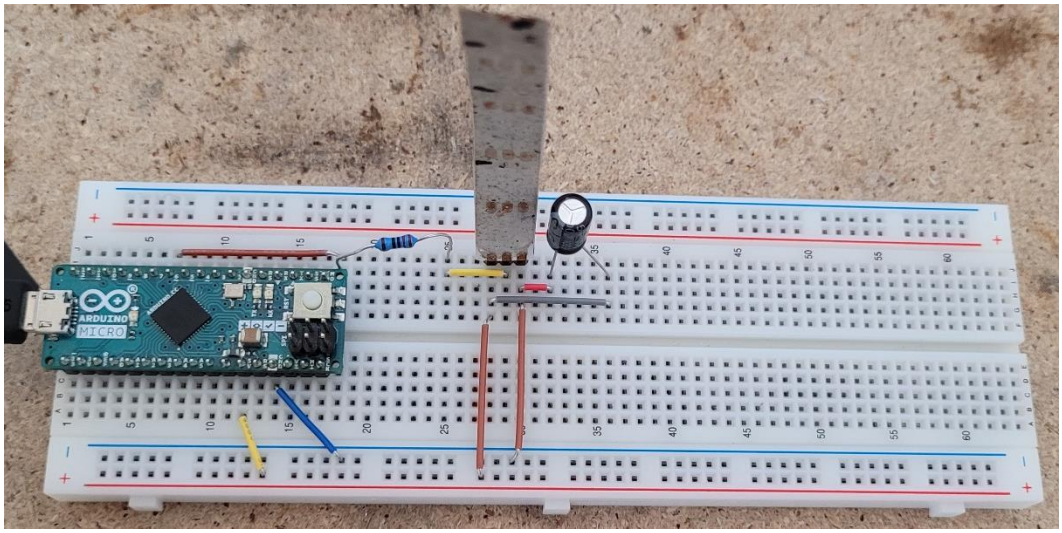
Übung 5

1.

1.1. Habe keinen 47µF Kondensator. Da er möglichst groß sein soll, habe ich hier einen 100µF Kondensator genommen.



1.2. Da der Aufbau gleich war und die entsprechende Library-Installation hier relativ einfach war, habe ich nur das entsprechende Video hochgeladen. Ist hier im Ordner zu finden.

1.3. Die entsprechenden Befehle konnten direkt aus der Adafruit-Library genommen werden. Einziges kleines Problem: Der Scope. Schreibt man die Funktionen hinter die Setup() bzw. Loop() stehen sie davor noch nicht zur Verfügung. Das passende Video zum Ergebnis ist ebenfalls hier im Ordner.

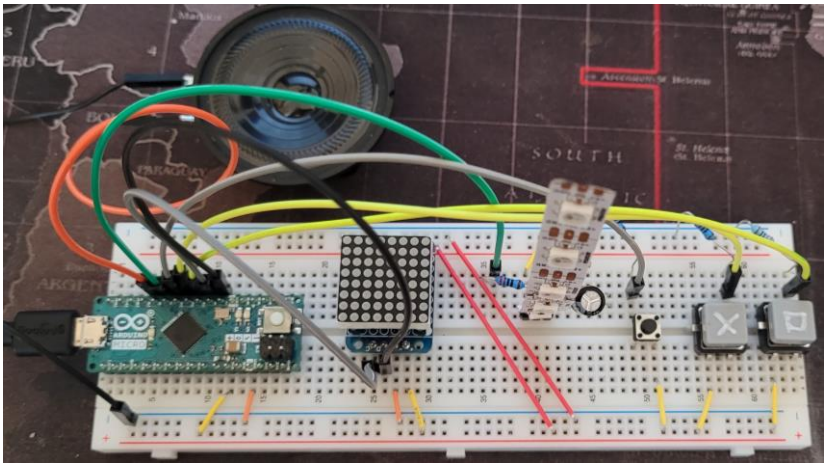
1.4. Bei der Aufgabe hatte ich erstaunlich wenig Probleme. Ich hatte relativ schnell eine Lösung, die auch funktioniert hat. Da ich aber mehr eigenen Code geschrieben habe und keine Erfahrung mit C++ hatte zuvor, war ich mir bei zwei Punkten etwas unsicher und habe mir Feedback dazu geholt:
Zum einen habe ich mehrere Timer benutzt. Damit ich mir den Aufewand der Variablen sparen kann, habe ich in der If-Abfrage das Ganze mit Modulo gelöst: millis()%Intervall <= 3. Nach dem Hinweis, dass das Ganze so zu Problemen führen kann (da nicht deterministisch), habe ich noch etwas recherchiert, inwiefern diese Methode angewendet wird. Tatsächlich habe ich einige Beispiele (z.B. HIER) gefunden, die das Ganze benutzen. Der größte Teil empfiehlt aber die Benutzung eines Threshholds, da so mehreren Problemen vorgebeugt werden kann. Zuerst gibt es das Code-Problem. Die Komplexität des Codes entscheidet darüber, wie lange die Ausführung dauert. Sehr simpler Code kann z.B. mehrere Male in der Millisekunde ausgeführt werden, sehr komplexer Code nur alle paar Sekunden. Bei der Modulo-Operation legt man den Ausführrahmen selber fest und um ein genaues Ergebnis zu

erzielen, müsste man die Code-Dauer genau bestimmen. Bei unterschiedlichen Code-Pfaden entstehen dabei zusätzlich noch unterschiedliche Laufzeiten. Durch den Threshhold, kann diese Problematik komplett umgangen werden, da der Code immer ausgeführt wird, sobald eine bestimmte Marke überschritten ist.
Als nächstes kann es passieren, dass das Inkrement der millis()-Funktion um zwei, statt üblich eins, steigt. So werden also Millisekunden-Zyklen und damit potentiell Code-Zyklen übersprungen und nicht ausgeführt. Auch hier sorgt der Threshhold für garantierte Ausführung. Dementsprechend habe ich meinen Code dann wieder auf den klassischen Timer mit Threshhold umgestellt.
Die zweite Unsicherheit war in Bezug auf die Ausführung von Methoden. In meinem Code wird die selbe Methode in der Loop immer wieder ausgeführt. Meine Idee war, dass die Funktionen als Art Instanz aufgerufen werden und mehrere Funktionsausführungen parallel existieren können. Hier war der Hinweis, dass die Funktion eigentlich blockieren sollte. Während der Ausführung wurden bei 6 LED's aber 3 Impulse dargestellt.
Der passende Code und das Video zur Aufgabe sind im Ordner zu finden.

2. Ganz zu Anfang habe ich die LED's und die Ground-Verbindung in die selbe Reihe auf dem Breadboard gesteckt. Dadurch ist der Strom nicht durch die LED's geflossen, sondern direkt zum Ground.
   Nachdem das Problem gelöst wurde, war der Rest relativ gut zu lösen. Mein erster Ansatz für die Weitergabe des Zahlenwertes an die shiftOut()-Funktion war es, das Ganze dezimal zu rechnen. Im Code habe ich die entsprechende Methode timer() drin gelassen. Nach dem Feedback zu einem vorherigen Übungsblatt, dass es eleganter ginge und die Lösung selber nicht besonders schön fand, habe ich noch ein wenig weiter recherchiert. Das hat mich darauf gebracht, dass für bitweise arbeitende Hardware, bitweise arbeitender Code gut geeignet wäre. Daraus ist die Methode time() entstanden, die doch sehr viel besser gelungen ist. Die Konvertierung der Methode war allerdings nicht so ganz problemlos. Anfangs habe ich das Ganze als Integer angelegt. Bitweise Operationen auf Integern führen aber zu komischen Ergebnissen. In der Dokumentation wird dann darauf hingewiesen, dass der Eingabe für shiftOut() aber ein byte sein sollte, wobei im anschließenden Beispiel ebenfalls Integer benutzt werden. Nichtsdestotrotz habe ich den Typ der Methode dann auf byte gesetzt. Die Fehler waren allerdings immer noch vorhanden. Das return-Statement lautete zu dem Zeitpunkt: return minutes << 6 + seconds. Wie ich bereits vorher feststellen durfte, führen bitweise und klassiche Rechenoperationen zu komischen Ergebnissen. Also habe ich auch hier noch eine bit-Operation eingefügt.
   Der passende Code ist zusammen mit dem Ergebnis-Video (vorgespult) im Ordner zu finden.

3. Schaltung:

Einzig neuer Bestandteil der Schaltung war die LED-Matrix, die allerdings durch den HT16K33 Controller angesteuert wurde. Nachdem ich die Arduino Pins für SCL und SDA vertauscht habe, brauchte ich etwas Zeit, um den Fehler zu finden. Alles Folgende funktionierte dann problemlos. Da ich leider nur zwei große Button zur Verfügung hatte, habe ich einen Kleinen für das „Kreis“-Symbol benutzt. Hier die fertige Schaltung:



Auch beim Software-Teil war nur das Ansteuern der LED-Matrix neu. Durch die Verwendung des Mikro-Controllers war dies aber auch recht einfach. Zur Erstellung der Byte-Repräsentationen habe ich zusätzlich ein externes Applet benutzt.

Code:

Im Code wird eine zufällige Wartezeit zwischen 0 und 10 Sekunden generiert. Um falsche Eingaben zu vermeiden, wird die Wartezeit mit delay() ausgeführt. Eine Umsetzung mittels eines Timers ist ebenfalls denkbar, sodass das Spiel direkt neu gestartet wird, auch bei verführter Eingabe. Ich habe mich aber für die blockierende Methode entschieden. Wird ein zufälliges Symbol angezeigt, wird durch die While-Schleife solange gewartet, bis ein Input erfolgt oder die Reaktionszeit, hier 3 Sekunden, abgelaufen ist. Bei richtiger Eingabe kommt die passende Melodie, ich habe mich für die Victory Fanfare entschieden, und das Spiel wird solange fortgesetzt, bis der Spieler 5 Punkte erreicht hat oder eine falsche bzw. keine Eingabe tätigt. Bei falscher bzw. keiner Eingabe wird der „Falsche Entscheidung“-Sound einer Fernsehshow (keine Ahnung mehr welche) abgespielt und der GameState zurückgesetzt, sodass das Spiel wieder von vorne startet.
Der Code und das Video können hier im Ordner gefunden werden.