ELECTRICAL & ELECTRONIC ENGINEERING
STELLENBOSCH UNIVERSITY

Simple Beat Matrix Sequential Drum Machine

Student Name:	Pieter Goos
Student Number:	19231466-2015
Study Leader:	Dr. Lourens Visagie
Date:	June 2019

UNIVERSITEIT iYUNIVESITHI STELLENBOSCH UNIVERSITY
100 1918·2018

Report submitted in partial fulfilment of the requirements of the module Project (E) 448 for the degree Baccalaureus in Engineering in the Department of Electrical and Electronic Engineering at the University of Stellenbosch.

Acknowledgements

Firstly a thank you goes to Dr. L. Visagie for allowing me to pursue this topic in the first semester as well as all the guidance provided.

Further thank you goes to Wynand van Eeden for the knowledge of PCBs and patience with all the 3D prints.

I would like to thank my girlfriend, Daena Voges, for standing by my side over the past years and dealing with all of my ramblings and frustrations. You have given me nothing but love and support over not only this project, but since the beginning of my studies.

My friends, thank you for the laughter and help you've given me. Particularly Richard Batt, having someone share this journey with nothing but jokes has been a lifesaver; and Alex Faustmann for the support and keeping me earthed for the future.

To my mother, Catty Goos, thank you for all the support you have given me over all these years and allowing me to study what I wanted with all the support I needed.

Finally, my dad, Danny Goos †, thank you for all your sit down talks and motivation boosters over the years. I miss you and I know you'd be proud of how far I've come. *Uw maateke is er geraakt, op naar het volgende advontuur, Love you!*

Plagiarism Declaration / Plagiaatverklaring

1. Plagiarism is the use of ideas, material and other intellectual property of another's work and to present is as my own.
   Plagiaat is die oorneem en gebruik van die idees, materiaal en ander intellektuele eiendom van ander persone asof dit jou eie werk is.

2. I agree that plagiarism is a punishable offense because it constitutes theft.
   Ek erken dat die pleeg van plagiaat 'n strafbare oortreding is aangesien dit n vorm van diefstal is.

3. I also understand that direct translations are plagiarism.
   Ek verstaan ook dat direkte vertalings plagiaat is.

4. Accordingly all quotations and contributions from any source whatsoever (including the internet) have been cited fully. I understand that the reproduction of text without quotation marks (even when the source is cited) is plagiarism.
   Dienooreenkomstig is alle aanhalings en bydraes vanuit enige bron (ingesluit die internet) volledig verwys (erken). Ek erken dat die woordelikse aanhaal van teks sonder aanhalingstekens (selfs al word die bron volledig erken) plagiaat is.

5. I declare that the work contained in this assignment, except where otherwise stated, is my original work and that I have not previously (in its entirety or in part) submitted it for grading in this module/assignment or another module/assignment.
   Ek verklaar dat die werk in hierdie skryfstuk vervat, behalwe waar anders aangedui, my eie oorspronklike werk is en dat ek dit nie vantevore in die geheel of gedeeltelik ingehandig het vir bepunting in hierdie module/werkstuk of n ander module/werkstuk nie.

Signature *Handtekening*	Student number *Studentenommer*
Initials and surname *Voorletters en van*	Date *Datum*

Summary

A drum machine allows for a musician to sequence multiple percussion sounds without the need for a drum kit or a drummer. This report not only focuses on the history, and thus the different variants of a drum machine, but also the design decisions made for not only hardware but also software. Every component is first designed in hardware and then driven by a controller through software. Each subsection of the system is discussed in detail, with various options on how to achieve similar results mentioned and explored. Measurements coming from the various parts of the device are shown and discussed. All of this cumulating to a finished, working, design allowing a user to schedule and play back their own sounds on a compact drum machine.

Opsomming

'n Drommasjien maak dit vir 'n musikant moontlik om verskeie perkussie geluide te orden sonder dat daar 'n dromstel of drommer nodig is. Hierdie verslag fokus nie net op die geskiedenis, en dus ook die verskillende veriante van 'n drommasjien, nie, maar ook op die ontwerpbesluite wat nie net vir hardeware gemaak is nie, maar ook vir sagteware. Elke komponent is eers ontwerp as hardeware en deur middel van 'n beheerder bestuur via sagteware. Elke onderafdeling van die stelsel word in detail bespreek, met verskeie opsies oor hoe om soortgelyke resultate te verkry en gebruik. Mate, wat van die verskillende dele van die toestel kom, word getoon en bespreek. Al hierdie versamel mooi tot 'n voltooide en werkende ontwerp wat 'n gebruiker toelaat om hul eie klank monsters op 'n kompakte drommasjien te skeduleer en hoor.

Contents

List of Figures

List of Tables

Listings

List of Abbreviations

AC Alternating Current

ADC Analog to Digital Converter

ASCII American Standard Code for Information Interchange

Bd baud

BPM Beats per Minute

CPU Central Processing Unit

DAC Digital to Analog Converter

DMA Direct Memory Access

FAT File Allocation Table

FATFS File Allocation Table (FAT) File System

FSM Finite State Machine

GPIO General Purpose Input-Output

I^{2}C Inter-Integrated Circuit

I^{2}S Inter-IC Sound

IC Integrated Circuit

LCD Liquid Crystal Display

LED Light-Emitting Diode

MIDI Musical Instrument Digital Interface

mux multiplexer

OTG On The Go

PCB Printed Circuit Board

PCM Pulse-Code Modulation

PISO Parallel In Serial Out

RAM Random Access Memory

RIFF Resource Interchange File Format

SIPO Serial In Parallel Out

SPI Serial Peripheral Interface

UART Universal Asynchronous Receiver/Transmitter

μC Micro-Controller

USART Universal Synchronous/Asynchronous Receiver/Transmitter

USB Universal Serial Bus

Chapter 1

Introduction

1.1 Project Background

Drum machines aid musicians by allowing them to generate drum, or other percussion, sounds on a schedule without a drum kit. Modern drum machines allow for the musician to program their own rhythms into the device. Since the beginnings of the drum machine in the 1930s the functionality has evolved in many ways. Several different programming and scheduling techniques are used by different manufacturers.

The first style of programmable drum machines placed different sounds in the rows and consecutive time steps in the columns. This drum machine will essentially act as a sequencer as it steps over the time steps repeatedly. This is the style of drum machine that will be used for this project as it proves one of the simplest to understand for new users. Some features will be added on top of the designs from the 1970s by using more modern luxuries such as a display and the ability to use one's own audio samples.

1.2 Project Aims

The aims of this project are as follows:

- Design and Manufacture a controllable matrix of indicators to schedule audio sample playback
- Design and Manufacture a method for the user to hear the sequenced samples
- Design and Manufacture a method to change the tempo and volume of the sequenced samples
- Have the ability to indicate the current state of the device's volume, tempo, and position in the sequence
- Design and Manufacture a PCB for the system
- Design and Manufacture an enclosure for the completed system
- Select a μC for controlling all of the device's features
- Design and implement a method for a Universal Serial Bus (USB) Storage Device to store the audio samples

Chapter 2

Background on Drum Machines

To design a drum machine one must first determine what that is and could be. Unfortunately it is not as straight-forward as one would believe as there are several forms of drum machines. The complexity and mere perception of what it is have changed since its beginnings in the 1930s, thus a point in time will need to be selected to be the basis of this project.

2.1 The Beginning of Drum Machines

Before drum machines could be programmed to the musician's liking it would make simple or pre-recorded sounds on a schedule. This meant that the *drum machines* of the time, the 1930s into the 1950s, acted mostly as timing to be used in conjunction with other instruments [1].
The earliest device one could conceive as a drum machine was the *Rhythmicon* by Léon Theremin and commissioned by Henry Cowell [1], [2]. The *Rhythmicon* was completed in 1931 and could only produce sixteen different rhythms [3]. These rhythms would all be more rapid than the previous, all based on a periodic base rhythm. This device would, however, only act as a concept for the next two decades.
In the late 1950s, large devices such as the *Chamberlin Rhythmate* or the *Wurlitzer Sideman* came on the market. These two offered a more pleasant sound to listen to by playing back audio tapes (at various speeds), and a system similar to a music box respectively [1], [2]. The latter, the *Sideman*, had rhythms styles including the Samba, Waltz and more[4].
Progressing to the 1960s, drum machines started to be manufactured with, at the time, new solid-state transistors. These transistors allowed the units to shrink in size drastically. In Japan, a new trend was encouraged by companies such as Ace Tone, and Keio-Giken to create organ accessories. Outside of Japan, the idea would be taken in a different perspective by creating pre-set rhythm boxes, such as the *Elka Drummer One*. For these devices to progress, a new level of control would be required [1].

2.2 Programmable Drum Machines

First one will have to understand what is meant by *programmable*: this refers to the ability for musicians to create their own patterns of internal sounds [1].
The first programmable drum machines came about in the 1970s with the *Eko Compute Rhythm* in 1972, as seen in Figure 2.1. This product featured a beat matrix, something

Figure 2.1: An Eko Compute Rhythm in Use [5]

that can still be found today [2]. Beat Matrices will be discussed in Section 2.3. Roland introduced the first drum machine to feature a microprocessor in 1978 with the *CR-78 CompuRhythm*. This allowed the user to store entire sequences [6]. Several years later they brought out the *TR-808* with a set of programmable synthesized analogue sounds [2], [7]. At this point, the demand for real drum sounds was high and thus came the Sample-Based Drum Machines.

In 1980, the *Linn Electronics LM-1* was released to be the first drum machine to use recorded samples as opposed to synthesized sounds [2]. Synth manufacturers began to enter the rhythmic device market with a product that made use of swappable memory cards, had Musical Instrument Digital Interface (MIDI) integration, among other features [1].

The most important feature to be added in the forthcoming years was being able to sample sounds on the device itself. This was pioneered by the *Akai MPC* series of drum machines. These featured sixteen responsive pad controllers and had all the features of a drum machine [1], [2]. This wave of new machines throughout the 1980s paved the way for stations such as the more recent *Native Instruments Maschine* series [1], an example of which can be seen in Figure 2.2.

Figure 2.2: Native Instruments Maschine Mk3 [8]

2.3 Beat Matrices

A beat matrix is a term used to describe how different sounds are scheduled and indicated on certain types of drum machines or sound sequencers. Each row is different sound, or beat, that can be sequenced on each column. In other words, each row represents a sound and each column a time step. An indicator will progress over the matrix's columns to show the current beat that is being played. In Figure 2.3 a simple inverted column indicator is used, meaning that if a sound in this column is selected the indicator will not show here if a is sound is not selected then the indicator will show. Alternatively, a separate indicator can be used to indicate the currently selected column. As the indicator column shifts right, to the next column, the sounds from each row that are selected in the current column will be played simultaneously. The pattern of sounds that are scheduled repeats once the indicator reaches the last column. The indicator column will then wrap around when it comes time to progress to the next step and indicate the first column again. The user has full control over which sounds will be scheduled at any given point on the grid by simply selecting a sound (row) at a given point in the sequence (column).

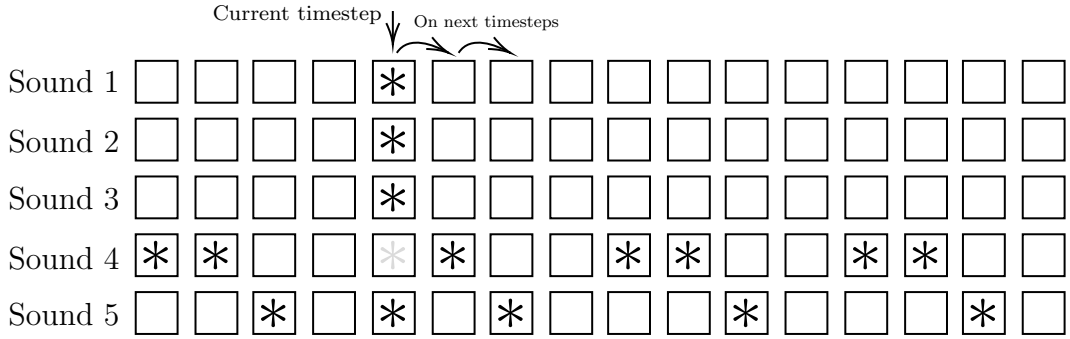


Figure 2.3: Beat Matrix Example

Shown in Figure 2.3 is an example of a beat matrix with the indicator in column five. A selected sound at a time point is indicated by an asterisk ($*$). At this moment 'Sound 4' would be played. At the next time step, which is determined by the tempo set in software, 'Sound 4' will play again, followed by 'Sound 5' and so on. The lighter coloured asterisk indicated that this is a selected sound but that the indicator has turned off as it is currently in the current time step column.

Chapter 3

Hardware Design

To produce a product that meets the specifications mentioned prior one has to design appropriate hardware options. All the options considered will be mentioned for each subsection as well as reasons for why a specific technique was selected.

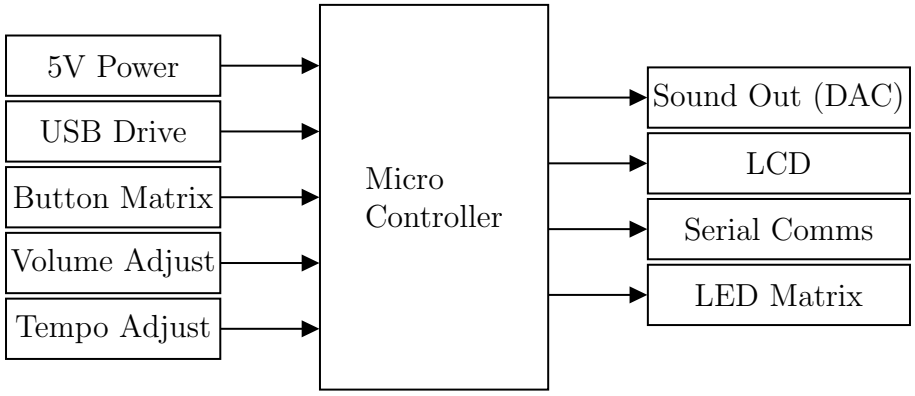


Figure 3.1: A Basic System Diagram

Figure 3.1 shows a basic system diagram based on the project aims. Each of the components in this figure will be explored further in this chapter. There will also be a detailed final system diagram at the end of the chapter.

3.1 Button and LED Matrix

Being the main way for the user to interact with the drum machine the button and LED matrices are integral to the correct usage of the device. Many LEDs and buttons have to be driven for the interaction to be usable, however, this requires a better understanding of how the matrices are to be set up.

3.1.1 Chosen Beat Matrix Design

For the physical drum machine, the same technique from Section 2.3 is used. LEDs are used as indicators and thus need to be placed in a matrix. Similarly, to change the states of the grid elements a matrix of buttons will be used. A design of sixteen beats (columns) was chosen with five different sounds (rows). This gives us a matrix of 16 by 5, hence 80 buttons and 80 LEDs. These values were chosen by assuming 4 beats in a bar and four bars, hence 16 beats. The number of different sounds, or rows, was determined by spacial

restrictions for the PCB as well as determining the maximum number of audio files that the μC can handle simultaneously.

3.1.2 Viable techniques to reduce pin count

The problem posed by having, in this case, 80 buttons and 80 LEDs is that driving all those would require at least 21 pins on the μC for each if it is being multiplexed. This is unrealistic and thus a method must be used to reduce the pin count required to drive the two features.

3.1.2.1 Basic Driving

As mentioned above, it is not viable to connect each node to a pair of pins individually as μCs generally do not offer that many pins. For this, there would have to be up to 320 pins to connect the 160 buttons and LEDs.
An alternative method is to consider the matrix as a set of buttons or LEDs at a certain row and column. Using this idea one can drive one of the columns or rows and read or write states on the alternate to select a certain button or LED. This technique is referred to as single multiplexing. The downside of this technique is that only one row or column of buttons or LEDs can be selected at one time, thus the system will need to cycle through the columns or rows respectively. This technique uses the same amount of pins as there are columns and rows, thus 42 pins for all the buttons and LEDs. This amount is still too high and should be reduced to make it more viable.

3.1.2.2 Driving a Row

A technique that can be used to reduce pins even further is to drive the columns of the matrix using a separate IC. There are three ICs that can be used to achieve this: Shift Registers, Decoders, and Decade Counters.

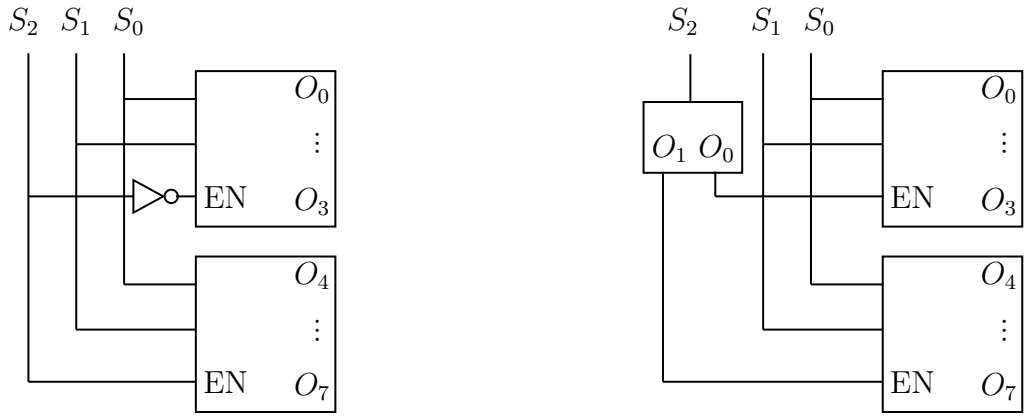
IC Type	**Signal Type**	**Input pins**	**Simultaneous Outputs (Max)**	**Extra logic for extension required**	**Reversed versions available**
SIPO Shift Register	Serial	3	n^*	✗	✓ (PISO)
Decade Counter	Serial	2	1	✓ (AND Gate)	✗
Decoder	Parallel	$\log_2(n^*)$	1	✓ (NOT Gate / Decoder)	✓ (Encoder)

Table 3.1: Input Pin Reduction Techniques using ICs

* - Note that in Table 3.1 the value n refers to the number of columns.
SIPO Shift Registers can be used to take in a serial, clock, and possibly a latch signal. As the name implies the chip will take in serial and output a parallel signal. The chip can be driven with a high level on the first clock pulse and a low for all that follows. Each subsequent clock cycle will then shift the signal up one pin on the output side of the IC. Alternatively, a serial signal can be shifted through and latched at the end to select multiple columns simultaneously. To extend the number of parallel output pins one can

make use of the inverted final output pin to use as the serial pin for consequent matching IC.

Decoders are the simplest to understand as they set the pin high which corresponds to the binary bits being input. The downside of this is that only one column can be selected at a time and that the column must be selected manually through serial. A decoder IC can be extended to have a greater number of outputs by using a NOT gate or further decoders to split the binary value between ICs as shown in Figure 3.2.



(a) Using NOT Gate (b) Using 1:2 Decoder

Figure 3.2: Extending Decoder ICs Schematics

Decade Counters shift a high bit along with the output pins at every clock cycle. This allows for one column to be selected at a time with the knowledge that the next column will be selected at the next clock cycle. Decade counters too can have the number of outputs extended, however, this requires AND gates as can be seen in Figure 3.3. This setup causes the first counter to stop whilst the second is active and vice versa. A notable issue would be that some outputs, such as the first pin on the 2nd chip, are no longer correct or usable and the whole system is shifted. Thankfully the system still only requires two input signals, the clock and reset lines, regardless of how large the loop becomes.

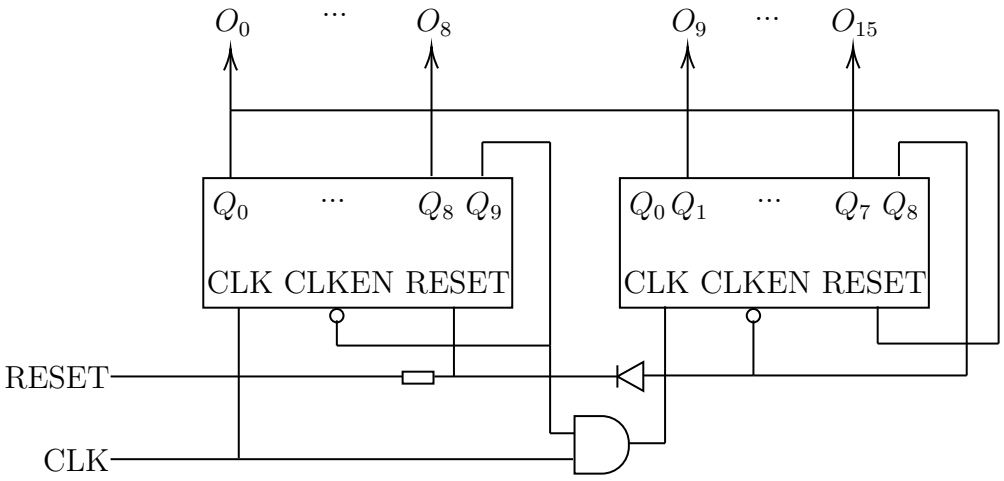


Figure 3.3: Extending Decade Counter ICs Schematic

3.1.2.3 Input

It can prove useful to drive large amounts of pins with very few connections to a controller, as in Section 3.1.2.2, but equally useful is reading the current state of a large number of pins and encoding this in a way for a controller to understand. A column in Table 3.1 shows that two of the circuits can also execute this task if an alternate version of the respective IC is used. For the SIPO shift register, a PISO shift register can do the inverse job: taking in a large number of parallel pins and outputting a serial signal. Similarly, a decoder acts in the opposite way of an encoder. The encoder will take a parallel signal and convert it to the respective binary value. These two ICs are not the only options for reducing the number of pins to read a large number of input states, another is a multiplexer (mux). The mux design would require the μC to step through each binary value on the select pins of the mux and check if the output pin's state has changed. This technique is however rather redundant as an encoder essentially is being designed with just one extra pin.

IC Type	**№ Connections to μC**		**Max Simultaneous inputs**
	Control	**Data**	
PISO Shift Register	3	1	n
Encoder	0	$\log_2(n)$	1
Multiplexer	$\log_2(n)$	1	1
Direct Connection	0	n	n

Table 3.2: Input Pin Reduction Methods

Table 3.2 shows the number of pins required to control various techniques of reducing pin count for reading a large number of pin states as well as the number of connections needed to read the data value. The number of input states the technique can read simultaneously is also listed, this is equivalent to how many pins of the parallel input can be high at the same time.

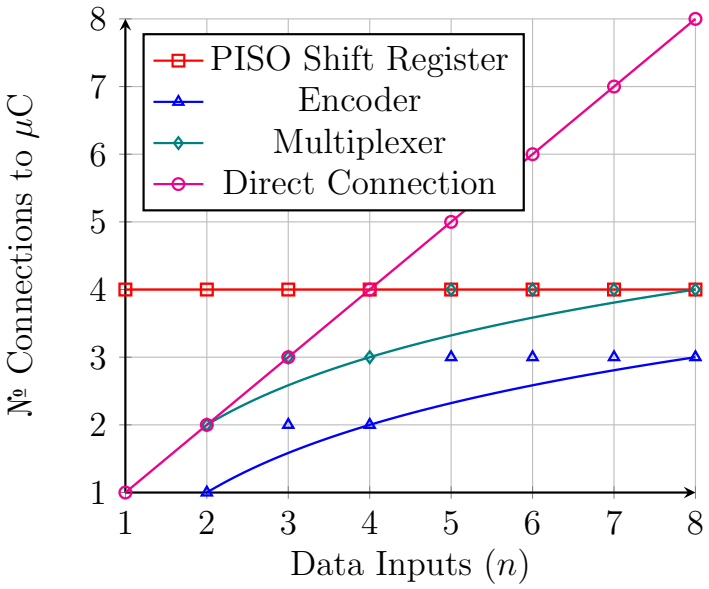


Figure 3.4: № of connections to the μC for Input Pin Reduction

Figure 3.4 shows the graphs of each technique's number of data inputs to a number of connections required to the μC. The data for this figure comes from Table 3.2. The lines for the Encoder (dark blue) and the Multiplexer (teal) are indicated differently to the other lines. The marker of the matching colour shows the actual value of needed connections to the μC for every given number of inputs. This is done as there cannot be a fraction of a pin used, thus the ceiling of the formula is used here. For example, if using an encoder with 3 data inputs $\log_2(3) = 1.58$, this is not a valid number of connections to the μC, thus this will need to be rounded up to two, as is shown in the figure. It should also be noted that these two techniques only begin at two data inputs as a 1 to 1 encoder or mux would simply be a short circuit.
Consulting Table 3.2 and Figure 3.4, it can be seen that if the more than one high data state can be expected simultaneously the direct connection method proves the most efficient up until 4 data inputs. After this point the PISO Shift Register set-up will be more efficient. Alternatively, if only one high data state is expected at a time, the Encoder set-up will always prove one pin on the μC more efficient, as explained before.
Techniques to expand the number of data input pins over multiple ICs will not be discussed as for this project it will not prove necessary. There are five rows, which falls under the eight-pin input of most ICs, hence this does not need to be looked into more.

3.1.2.4 Driving Rows and Columns as a chain

If no data needs to be received back by the μC both rows and columns can be controlled by a single chain of ICs. This technique can be used in the case of the LEDs as the positive and negative pins will be set high and low depending on its intended state. Seeing as the LEDs require a potential difference to light up there two methods of turning the LED off could be used: Firstly driving the positive side of the LED low to match the low state of the negative pin, or secondly matching the negative pin high to that of the positive pin. Both methods will cause there to be no potential difference over the LED and thus it will not light.

3.1.2.5 Driving Serial Peripherals from the μC

Up to this point no technique for driving the clock, serial data and other pins have been mentioned. The simplest one to consider is Serial Peripheral Interface (SPI) as this provides not only a clock signal but also handles the serial data transmission. This interface could also be further expanded upon by manually setting General Purpose Input-Output (GPIO) pins from the μC to control ICs related functions, such as to enable or chip select. An example of a place where this could be used is for a Shift Register. In that case, the Serial Data and RCLK pins are controlled by the SPI interface and the SRCLK pin will be controlled manually to set the state low before and high after transmission.
An alternative technique is 'bit banging'. This is where software on the μC directly controls the states of all the pins as GPIO manually. This would have the μC run a loop with a "clock" changing state as well as outputting the corresponding data bit as fast as the μC can run the code. This may allow for more control of each signal, but the increased coding complexity and lack of known timing may cause issues for some ICs. Overall, this method is not recommended if an alternative, such as SPI, is available on the μC being used.

3.1.3 Basic Button and LED connection

Up until this point buttons and LEDs have only been mentioned but no thought has been given as to how they will be connected in the matrix. Thanks to the LED being a diode it will only allow current to pass in one direction. Using the formation shown in Figure 3.5b it can be determined that to power any given LED in this matrix its corresponding row must be set high and column set low.
For the buttons, it is slightly more complex as now a state of a certain point must be read by the μC. The button connections in the matrix can be seen in Figure 3.5a. To achieve this the columns will be driven high one at a time with the rows being pulled low through a resistor. Depressing the button will cause the column to be shorted to the row through a diode. The diode is there so that multiple buttons can be depressed simultaneously, and the state of each can still be read independently. The potential difference on each row can be read by the μC. If it is high then the button at the respective row and column's button is being depressed and vice-versa.

(a) Button Connection (b) LED Connection

Figure 3.5: Basic button and LED Diagrams

3.1.4 Chosen Designs

There are several designs to choose from as discussed in the previous section. Each has its own advantages and disadvantages. Several iterations of chosen designs were made until a final form was reached. These chosen designs will be discussed in the section for buttons and LEDs respectively below.

3.1.4.1 Button Matrix Driver

For the button matrix, two set-ups of connections must be used to drive the rows and columns separately. This is necessary as there has to be a state set for the columns and the respective states must be read back in from the rows.
To read in the states from the matrix's rows a technique for five inputs should be considered with the ability to read all five pins simultaneously. Consulting Table 3.2 and Figure 3.4 the options can be reduced to the PISO Shift Register or Direct Connection techniques. The most efficient option, in terms of the number of required pins on the μC, would be the Shift Registers. This technique could prove more tedious to deal with in software and as another point of failure. The direct connections may use one more pin (five instead of four), but the simplicity that comes with this technique allows for faster response in software on the μC. Using the Shift Register technique would also require an additional clock interface which is not available on the μC. For these reasons the direct

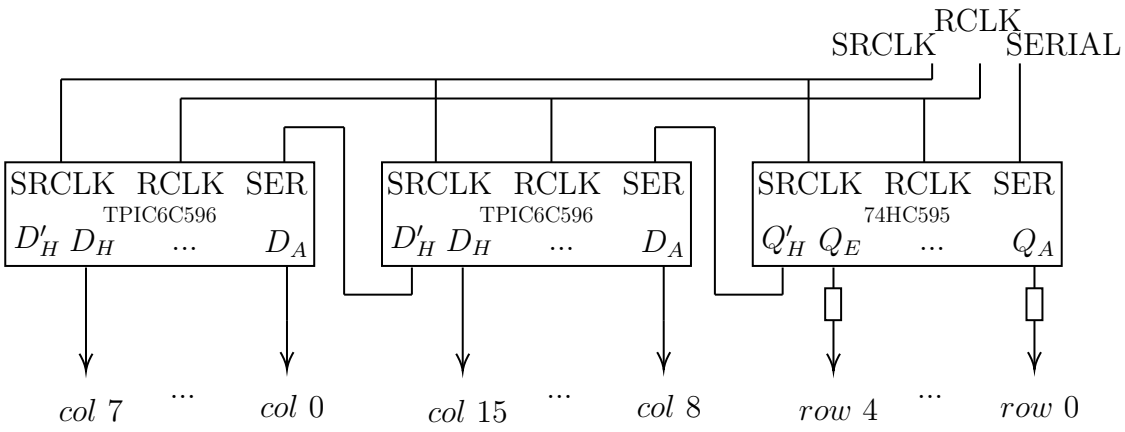
connection technique is used.
To drive the columns the techniques from Table 3.1 are considered: Shift Registers, Decade Counters, and Decoders. For this application only one column will be driven at a time thus all the solutions would work. Of the three methods, the decoder would be the most difficult to implement in software as well as requiring the most amount of pins. This method would need $\log_2(16) = 4$ pins, as a pose to the maximum of three which the Shift Register technique requires. Both other techniques require the μC to keep track of the currently selected column. This, therefore, leaves the choice between a reduced pin count (two vs three), and the simplicity of controlling and building the circuit. Even though the Decade Counter technique requires a more complicated circuit, as is seen in Figure 3.3, it is simpler to control from the software. The Shift Register technique requires the software to trigger the data manually, whereas the decade counter, in its current formation, will step from pin zero through pin fifteen (sixteen pins) and automatically reset. For this reason and the reduced pin count, the decade counter technique is used.
The ICs needed for this design are the 74HC4017 Decade Counter, and 74HC08 AND Gates. The circuit will need a diode, an FDLL4148 is used, and a 1kΩ resistor. The diode allows the system to be manually reset, without it the clock enable of the second IC, and all connected points, will remain near zero Volts, hence never resetting the circuit. The resistor is in place to allow the reset state from the μC to pull the value rather than setting the state. This allows the output from the second IC to pull the voltage high on the reset pin of the first IC as otherwise this would be pinned to low when the reset is low.

3.1.4.2 LED Matrix Driver

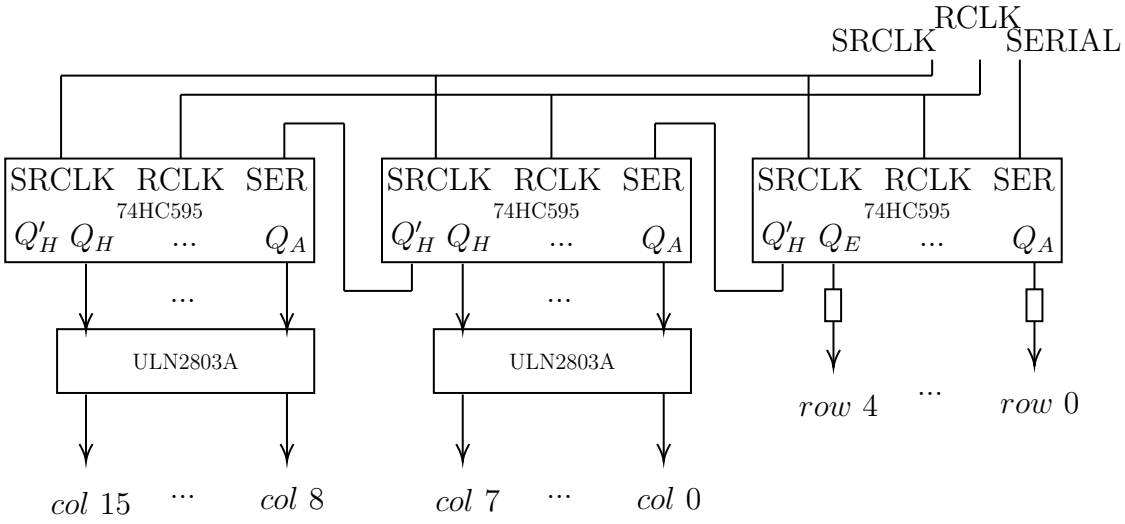
As explained previously, the LED driver can be made of a single chain of ICs. In this case, the ICs would have to be SIPO Shift Registers as the serial signal has to be passed through the entire system and then maintained there for a short time to light the appropriate LED. Two methods were considered to drive the columns of the matrix. These would require a method of draining the current generated by, up to, all the rows in said column. The first method made use of the TPIC6C596 IC. This IC allows for up to 100mA of continuous drain current with a peak capability, when pulsed, of 250mA per pin. This method would allow the IC to slot in the chain of SIPO Shift Registers, as shown in Figure 3.6a. The second method makes use of the same Shift Register as used for the rows, to which the ULN2803A Darlington Transistor Array IC will be connected on the outputs, as shown in Figure 3.6b. This IC allows for up to a 500mA collector current to be drained.
As can be seen in Figure 3.6, the chosen SIPO Shift Register is the 74HC595. This type of shift register will hold its value as it has an output buffer which in turn reduces flickering of the LEDs as they stay on for as long as possible.
Resistors will be connected to each of the driver pins, in this case, the rows. These resistors will restrict the current that can flow through the LED connected to the respective row.

$$\begin{aligned} R &= \frac{V_{CC} - V_{f(LED)}}{I_{max(LED)}} \\ &= \frac{3.3 - 2.0}{30m} \\ &= 43.33\Omega \approx 47\Omega \end{aligned} \tag{3.1}$$

The resistance calculations for the resistor in each row are shown in Equation 3.1. This uses the chosen LTST-C150KFKT LED's values for maximum forward current and for-



(a) TPIC6C596 Option to drive columns



(b) ULN2803A Option to drive the columns

Figure 3.6: Circuit Designs for driving the LEDs

ward voltage. The value of 43.33Ω represents the minimum resistance value for the resistor to have before the maximum forward current of 30mA is surpassed. To allow for some tolerance, as well as selecting a standard value resistor, the value of 47Ω is selected.
In the current configuration, each column pin will experience a maximum of 5 rows ∗30mA = 150mA. For this reason, the ULN2803A variant is the only one capable of draining the current of the rows effectively. Even though this system is being cycled through at high speed, it is safer to consider the continuous current ratings as opposed to the maximal pulsing ratings.
A design which could prove more efficient than that chosen would be to invert how the LEDs are places and use the rows to drain the current and columns to drive the state. If this technique was used there would be one less ULN2803A IC, and there would have to be fewer iterations in software as now up to sixteen LEDs can be lit up simultaneously as opposed to the five from the current technique. Considering the maximal current of the ULN2803A it can be seen that this technique would still fall in the safe boundary: 16 cols ∗30mA = 480mA.

3.2 Tempo and Volume Control

Perhaps the simplest methods of interaction for the end user are the tempo and volume control knobs. As the names imply, there will be two methods for the user to change the tempo (Beats per Minute (BPM)), and volume of the sounds scheduled and produced by the system.

3.2.1 Methods to Measure Rotation

Two methods to measure how much the end user has rotated a knob are considered: a potentiometer, and an incremental rotary encoder.
Rotary Encoders come in many variations, however, in this situation only one type, the incremental rotary encoder will be considered. This is the simplest type of rotary encoder as it simply has voltage supply lines with two outputs that have a 90° phase offset from one another. Figure 3.7 shows the two output pins from a rotary encoder. From this it

(a) Clockwise Rotation Output (b) Counter Clockwise Rotation Output

Figure 3.7: Rotary Encoder Outputs

can be seen that if signal B lags behind signal A the rotary encoder is turned clockwise and visa versa. Rotary Encoders allow for unlimited revolutions in either direction as they simply show which direction they are being turned, they, however, require two pins to read the signal as well as timers on the μC.
Potentiometers are a far simpler and more rudimentary technique. A potentiometer can be used as a voltage divider circuit by placing it between ground (0V) potential and the supply voltage (V_{cc}). The wiper pin can then be the voltage divided using Equation 3.2. In this equation, R_{gnd} refers to the resistance from the wiper to ground and R_{vcc} refers to the resistance from the wiper to V_{cc}.

$$V_{div} = V_{cc} * \frac{R_{gnd}}{R_{vcc} + R_{gnd}} \tag{3.2}$$

To interpret the voltage value V_{div} an Analog to Digital Converter (ADC) must be used on the μC. The ADC will measure the voltage digitally as the number of steps between zero and the size limit it is set to, for example, a 12-bit buffer will allow a value between 0 and $2^{12}-1 = 4095$. If V_{div} was $0.5V_{cc}$ then the value from the ADC will be $(0.5*2^{12})-1 = 2047$. The potentiometer method has the benefit of only having one wire going to the ADC on the μC. On the other hand, a potentiometer cannot turn more than a set angle, usually 270° or more.

3.2.2 Chosen Method

Due to the simplicity of the potentiometer circuit, it can be used for both the volume and tempo control. The limited amount which the user can turn the potentiometers allows an association between a value and a position. For example, if the user wants the volume at 50% then they can set the potentiometer for volume half way and know that that position will always yield the same volume value. Similarly, for the tempo, it could prove useful to have a set range limited in software. The chosen potentiometers have a value of 10kΩ and are linear over 270°.
An offset was added to the tempo potentiometer by soldering a resistor between the potentiometer and V_{cc}. This resistor reduced the maximum tempo and its value was determined with Equation 3.3. S_{max} is the fraction to which the tempo must be limited compared to its prior maximum.

$$\begin{aligned} S_{max} &= \frac{R_{gnd}}{R_{vcc} + R_{gnd} + R_{offs}} \\ R_{offs} &= -R_{vcc} - \left(1 - \frac{1}{S_{max}}\right) R_{gnd} \\ &= \left(\frac{1}{S_{max}} - 1\right) R_{pot} \qquad \text{when } R_{vcc} = 0,\ R_{gnd} = R_{pot} \end{aligned} \tag{3.3}$$

The desired fraction, S_{max}, is chosen to be 87% as this reduces the maximum BPM is low enough to be useful. This results in the R_{offs} value being 1494Ω, this is then rounded to 1.5kΩ.

3.3 LCD

The tempo and volume values that are controlled by the μC and potentiometers, as discussed prior, are displayed to the user using a Liquid Crystal Display (LCD). This LCD must update frequently enough for the user to constantly know the displayed values in real time.

3.3.1 Driving a Parallel LCD

The LCD that is selected is the PC1601A due to its availability at the University and it is used in prior projects, and readily available driver source code. This LCD is sixteen characters wide and one row high. It is driven by a 6800 4/8-bit parallel interface in 4-bit (nibble) mode, this way only four pins are needed to control the data flow to and from the LCD. Running the display in nibble mode means that four extra pins are being traded in for double the time to read or write one byte of data to or from the display. Other than the data pins, the LCD must also be controlled with three control lines: Enable (E), Read not Write (RNW), and Register Select (RS). All seven pins will be connected to the μC's GPIO pins. Even though the LCD is powered by a 5V supply it can be controlled by 3.3V logic coming from the μC as $V_{IH} \geq 2.2$V.
To control the contrast of the LCD there is a pin, labeled V_O, which must act as a voltage divider circuit (See Equation 3.2). As per the datasheet, a trimmer potentiometer of 20kΩ is used to divide between 5V and Ground, it is adjusted manually until the desired contrast is reached.

3.4 Serial Communication with a PC

The serial communication connection to a PC is an optional feature being used as a debugging tool during the development of the complete system.

3.4.1 Reason for implementation

This interface was added as a debugging tool for during the development phase, however, it was also included as an option to upgrade the system at a later stage, time permitting. This upgrade would come in the form of MIDI support over USB. MIDI is based on a serial communication protocol which can be substituted with the Universal Asynchronous Receiver/Transmitter (UART) protocol. This additional feature was not added due to time restrictions.

3.4.2 Circuit Design

The base circuit design, as well as the breakout board used, comes from the Design E314 Module. Shown in Figure 3.8 is the FT230-XS UART to USB IC is used for this feature

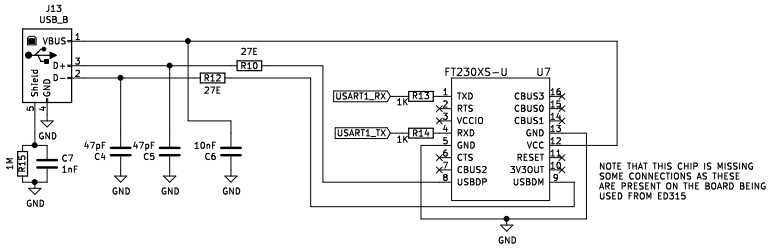


Figure 3.8: Circuit Diagram for Serial Communication

connected to a USB type-A connector. All necessary resistors and capacitors are included as per the typical configuration to be powered by USB bus power. Data and power lines from the USB connector are all connected through capacitors to ground, 47pF and 10nF respectively, to suppress any Alternating Current (AC) noise on these lines. Resisters placed on the USB data lines are 27Ω to reduce the maximum current. 1kΩ resistors are placed on the UART TX and RX connections to reduce parasitic currents between the μC and FT230-XS IC.

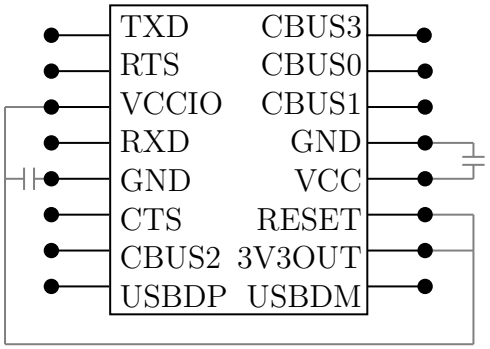


Figure 3.9: Pinout of FT230-XS breakout board

Further capacitors are placed on the breakout PCB as shown in Figure 3.9. These, again, are in place to reduce AC noise on the supply (V_{cc}) and 3.3V out lines.

3.5 Micro-Controller

A μC ties together the entire system as a whole by interfacing with all subsystems using the appropriate communication techniques. For this reason, it is important to pick one that not only supports the required features but also is approachable to use and program in the given time frame.

3.5.1 Picking a Micro-Controller

Picking an appropriate μC is essential, as without it all the systems cannot communicate with each other and produce the desired working of the complete device. Several μCs are considered for the drum machine, these are shown in Table 3.3.

	Board	UART	USB FS	ADC	SPI	DAC	RAM (kB)	Flash (kB)	
Renesas	YRPBRL78G13	✓ (2)	✗	✓ (12)	✗	✗	4	64	
ST Micro	NUCLEO-F334R8	✓ (3)	✗	✓ (42)	✗ (1)	✓	16	64	
ST Micro	32F469I Discovery	✓ (4)	✓	✓ (72)	✓ (6)	✓	384	2048	
ST Micro	32F411E Discovery	✓ (3)	✓	✓ (16)	✓ (5)	✓	128	512	
ST Micro	NUCLEO-F767ZI	✓ (4)	✓	✓ (72)	✓ (6)	✓	512	2048	

Table 3.3: μC Selection

The options shown in Table 3.3 are a small selection of development boards considered for this project. The list is composed of four options that the University has in stock and one that would have to be ordered (the Nucleo F767ZI). Ticks and Crosses in the table show whether this μC can support the needed features as have been discussed prior. A DAC, two SPI channels, two ADC Channels, USB host support, and one UART Channel are needed to allow all required features to work. These requirements immediately rule out the first two options, the YRPBRL78G13, and the NUCLEO-R334R8. Due to the immediate availability of the STM32F4xx boards, these will be considered first. The smaller STM32F411VE board is tested first as it was supplied with examples for reading audio wave files from a USB storage device. The STM32F469I Discovery board comes with a large display on the front which in turn means it is larger than the STM32F411. Storing audio files on the μC in Random Access Memory (RAM) may also play a role in making the decision of which processor to select. If, as is the case here, five sound files must be stored, a minimum of 15kB per audio file should be assumed, thus 60kB of RAM for audio files would be needed at a minimum. An array is also required to hold a buffer of 4096 16-bit samples, thus a further 8kB is needed in RAM. Audio files have to be stored in RAM as the flash memory cannot be accessed by the developer. Further information on the audio files and how they are handled is covered in Section 4.4.
As the STM32F411VE is physically smaller than the STM32F469I and has all the necessary features required for this project it is the chosen μC development board. RAM left for the μC code to use in this case would be at most 60kB, this should prove sufficient. The availability of the board itself and support from staff at the university also supported this decision.

3.5.2 Onboard Features

The chosen STM32F411VE Discovery board has some onboard features permitting some required product requirements. The onboard features are:

- L3GD20: 3 Axis Gyroscope
- LSM303DLHC: Magnetic sensor and 3D Accelerometer
- MP45DR02: Omnidirectional microphone
- CS43L22: Audio DAC with Class D Amplifier
- USB On The Go (OTG) micro-AB connector
- Debugging LEDs
- Two push buttons (User and Reset)
- ST-Link Programming Interface
- 3.3V Regulator

The most important of these features is the USB OTG connector and the DAC as these allow for the required audio playback and file reading from a USB storage device. The USB OTG connection is set up on the μC as a USB Host interface using five pins on the μC. To connect a USB storage device a simple USB OTG cable must be made and connected to the micro-USB port allowing the μC to communicate with the USB drive through FAT File System (FATFS).
The DAC, the CS43L22, is connected using the I^2S and Inter-Integrated Circuit (I^2C) protocols for sound and controlling the IC respectively. These two protocols take up seven pins on the μC. There is a 3.5mm audio jack connected to the headphone output of the DAC in stereo. This jack is the main connection point for a sound device, such as headphones or a speaker, to be connected to.
The ST-Link interface allows for the μC to be programmed / flashed by a computer via a mini-B USB cable. This onboard programmer not only allows for the code to be written onto the IC but also allows for debugging during development as it can relay information about the running code. This feature can be bypassed by moving the two jumpers on CN3.
The built-in voltage regulators ensure that the supplied 5V power from the mini-B USB cable can be supplied to the μC as 5V and 3.3V where needed. These voltages are also be used as a supply for other sub-systems on the board.
Further onboard features will remain unused as they would serve no additional purpose in the end product. All these features are initialized in software but will remain unused. The black Reset button will still be functional allowing the μC to reboot and restart the software in case this is desired.

3.6 Printed Circuit Board

The PCB is manufactured on University Campus using an *LPKF ProtoMat S103* PCB milling machine. A single A4 sized sheet of double-sided copper plated fibreglass PCB material was used. The design of the PCB was made in KiCAD and exported for use by the S103. Special attention was paid to the side on which the through-hole components would be connected. This is due to the fact that there would be no through-hole plating, thus the side which the pin is soldered to is the only connected side. To allow for traces to connect between layers vias are added and a thin wire is put through the hole and soldered on both sides. Testing headers are added throughout the board so that the behaviour of signals can be seen with a logic analyser or oscilloscope at the output and input of various subsystems. Further information, including the adjusted PCB designs, can be found in Appendix D.
The spacing between certain traces and the size of traces caused some errors when milling the PCB. These errors were torn traces, where the copper tore and caused an open circuit, and excess left over copper bridging traces. These were remedied with thin wire to bridge open circuits and a fibreglass brush and knife to remove excess copper between traces.
After soldering is completed the exposed portions of the board is covered in a conformal coating, for this board Q30 by Barrat Integra was used.

3.7 Minor Remaining Hardware

To prevent the μC and other components from touching the surface which the drum machine is placed on stand-offs are used and screwed through the PCB.
To cover the front side of the PCB and hold the potentiometers a panel is manufactured. The cover was designed in Autodesk Fusion 360 as a 3D body. Originally a pane of acrylic (Plexiglass) was to be milled using 2.5D milling on the *LPKF ProtoMat S103*, due to software issues this technique was not used. As the file for the milling process was already made it was converted so that it could be used in a MakerBot Replicator Z18 3D extrusion printer. This panel would also be screwed into the stand-offs from before using Gamebit screws to prevent tampering.
As there is now a panel in the way of interacting with the buttons a method to see the LED and press the buttons must be added. The chosen solution is cutting the plastic used in hot glue guns. This acts as a good light diffuser and is rigid enough to use as a button extender.

3.8 Final System Diagram

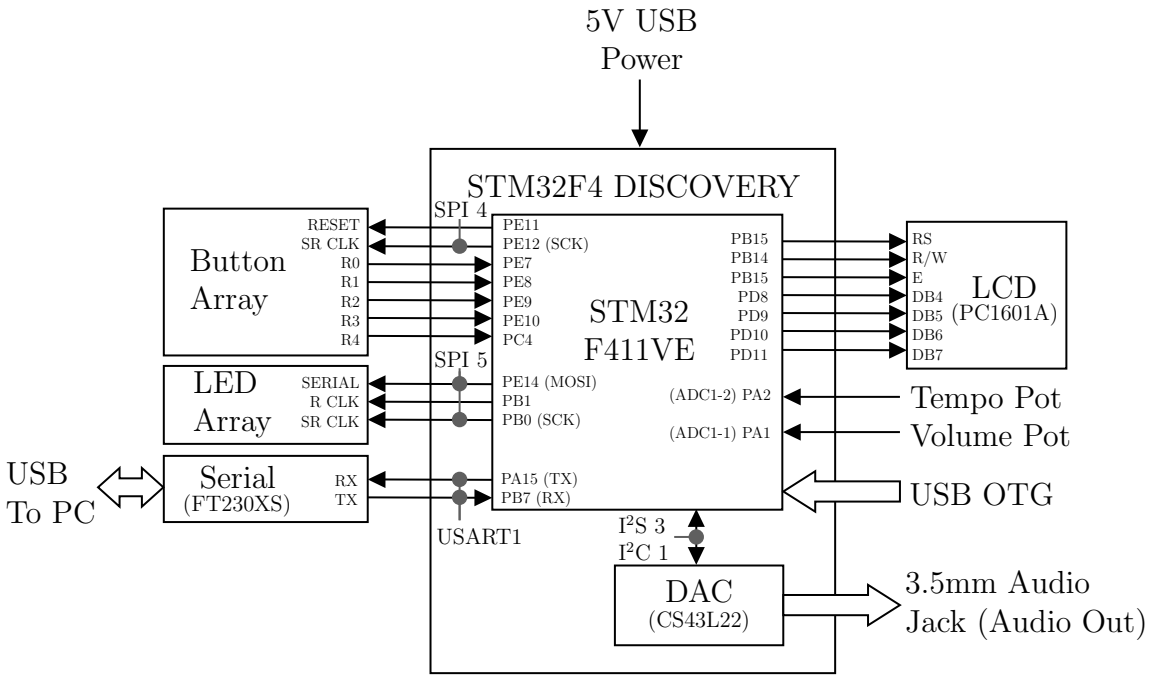


Figure 3.10: Complete System Diagram

Figure 3.10 shows the complete system diagram, however, it should be noted that there are portions of the STM32F411 Discovery board are not shown due to them not being used. An example of this would be the included onboard MEMS microphone **STM32Datasheet** This figure shows where the bounds of each sub-section of the completed system are and their means of communication.

Chapter 4

Software Design

4.1 Main Software Loop

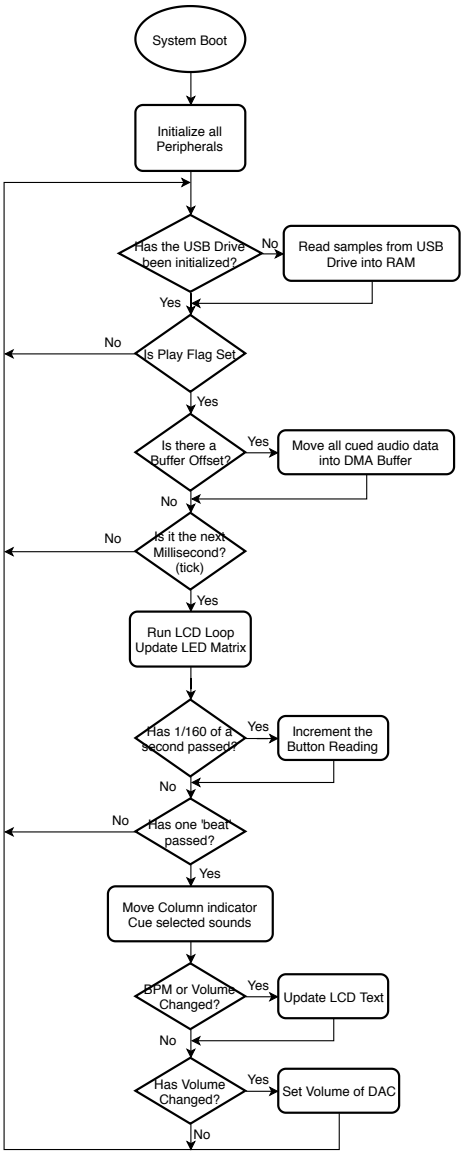


Figure 4.1: Full System Software Flow Diagram

When the μC begins to run the code there are two stages. Firstly there is the initialization stage where all code that must only happen once is run. The code here includes all the peripheral initialization and some variable manipulation and initialization. After this stage, an infinite loop begins. In this loop, all code to control the peripherals and subsystems is found. The μC will step through all the code in the loop and then it will begin back at the beginning of the loop. Several flags will need to be set to allow for conditional functions. The μC will run this code as fast as it possibly can, thus, seeing as a consistent time between certain functions is needed, a system tick (sysTick) will update every millisecond. This millisecond clock can then be used to see if a certain amount of time has passed and thus cause desired functions to run. As the μC runs faster than its one-millisecond timer a semi-asynchronous function can be run. This can be seen when the μC updates the audio buffer as soon as a flag is set. Efforts have been made to cut out functions that would run repetitively unnecessary, for example, the updating of the LCD text.
This chapter will discuss each of the major sub-sections' software implementation on the μC.

4.2 LED and Button Matrix Control

4.2.1 Setting LEDs

4.2.1.1 Updating the Driver ICs

Several variables are used to store the current LED states and use them to drive the LED matrix. To store the value of each row there is an array of 16-bit integers. Each integer in the array holds the values of the led in the corresponding row, so if the first and third LED in row zero is enabled the integer would be $(1010\ 0000)_2$. This variable is named cData. When sending data over the SPI protocol only one column at a time is selected. A tracker for the columns must be in place that increments often enough to create a believable persistence.

```
1  spiData[0] = (0x8000 >> c) & 0x00FF;
2  spiData[1] = ((0x8000 >> c) & 0xFF00) >> 8;
3  spiData[2] = 0;
4  for (uint8_t r = 0; r < 5; r++){
5      spiData[2] |= (((cData[r] & (0x8000 >> c)) << c ) >> (r+8));
6  }
7
8  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET);
9  HAL_SPI_Transmit(&hspi5, spiData, 3, 10);
10 HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
```

Listing 4.1: Driving a row of LEDs

In Listing 4.1 the code used to drive the LED matrix can be seen. Lines one and two are selecting the current column (c) by shifting $(1000\ 0000\ 0000\ 0000)_2$ right by the column number to pull high the chosen column physically. That value is then masked and shifted to allow the 16-bit integer into two eight bit integers. Lines three through six step over each row and modify the row variable. Each column variable is checked if the value is high or low at the specific column and it is placed from the eighth to the third bit of

the row variable (spiData[2]). If the first and fifth row is high in the selected column the result would be $(1000\ 1000)_2$.
Lines eight through ten set the RCLK pin low, transfer the data, and sets RCLK high again. The data transferred is spiData which is three bits long. As the shift registers are connected to the rows, the low columns and then the high columns the data must be transmitted in reverse: high columns, low columns, and then rows - hence spiData is placed in this order. SPI5 is used for this transfer in the Motorola Frame Format that is eight bits long and the least significant bit is sent first.

4.2.1.2 Changing the State Variable

To change the state of any LED at any time a simple toggle of a bit in cData can be used. The desired row's cData element is selected and is then XOR'ed with $(0001)_2$ shifted left $15 - c$, where c is the desired column.

4.2.1.3 Incrementing the Indicator Column

As time passes the indicator column must move on to the next column. The time is decided by the BPM value calculated in Section 4.3.2. Toggling of the LEDs in the required columns is quite simple. A loop will go over each row and toggle the LEDs in the required column. This can then be done twice, once to the new column and once to the current as this will revert the LEDs back to how they were before selection. If the next column is zero then the current will be column fifteen.

4.2.2 Reading a column of buttons

4.2.2.1 Reading the Selected Column

To drive the decade counter circuit being used for the column select a single clock input must be provided. To allow for actions to happen between each clock cycle a custom clock is used rather than one generated by the μC itself. To emulate this clock signal the corresponding GPIO pin will be toggled high and then low. Once the clock is toggled the selected column tracking variable will increment, or reset to zero if the value becomes sixteen. A two-dimensional array is kept with the previous state of each button.
A loop goes over each row and checks if the new state of the button does not match that of the prior. When this is the case the LED in that location will be toggled if the button is being pressed, not released. If the two states did not match the previous will be updated to match the current state.

4.2.2.2 Resetting the Button Matrix

Resetting the button matrix is made incredibly simple thanks to the decade counter circuit being used. This system allows for the Reset line to be pulled high resulting in all outputs going low and the position being reset. In the software, the selected column tracker variable must be set back to zero as well.

4.3 LCD with Tempo and Volume Control

4.3.1 LCD

The code for the LCD was adapted from a file supplied by Dr. L. Visagie designed for the STM32F344R8 Nucleo Board.

4.3.1.1 Sending Data

To send data to the LCD a nibble must be sent at a time as only four lines are in place. This decision was made to reduce the pin count to the μC. To send data to the LCD the Enable (E), and Read not Write (RNW) pins must be pulled low. The Register Select (RS) pin must be set to the appropriate state, low for commands or high for data. Just before the data is sent the Enable pin is pulled high and is kept that way until the nibble has been sent. Now the data nibble must be transmitted in parallel over DB4 through DB7.
To send a byte of data, rather than a nibble, the byte must be split in two. This can be achieved with masking and shifting. The high nibble must be sent first then only the low nibble. The controller on the LCD will reassemble the data as needed. Between each nibble being pushed a short delay must take place to ensure the correct separation of the data.

4.3.1.2 LCD Finite State Machine

State	1st Nibble (RS, Nibble)	2nd Nibble (RS, Nibble)	Time Delay after (ms)	Description
0	-	-	-	Idle
1	0,3	-	5	
2	0,3	-	1	
3	0,3	-	1	
4	0,2	-	1	4 bit mode
5	0,2	0,8	1	4 bit mode 2 line
6	0,0	0,14	1	Display On Cursor Off
7	0,0	0,1	1	Clear Display
8	0,0	0,6	1	Increment Cursor
9	0,8	0,0	1	Home
10	-	-	100	
20	0,8	0,0	3	Home
21	Push Data Byte		-	

Table 4.1: LCD Initialization and Run State Machine

Table 4.1 shows the states of the Finite State Machine (FSM) used to initialize and run the LCD. To initialize the display states one through ten must be run. Every state will cue the next state upon completion. Each of these states will send one or two nibbles using the method mentioned before. After the data is transmitted a delay, in milliseconds, must occur for the LCD to process the data sent. A description of the commands sent is also shown in Table 4.1. These states will only be called once when the μC is first started

up. The state variable will not be set to one again after state ten has been completed
States twenty and twenty-one are in place to reset the cursor location and send a byte of data respectively. The data byte is sent using the code for sending a byte of data discussed prior. When the data for the display is updated the state will be set to twenty so that this procedure runs.
State zero is in place as an idle marker. This is used to indicate when nothing must be done to the display. Calling this state is done once all other states in the FSM have been completed. A simple check before the FSM's switch statement allows the update of the time tracking feature to be skipped entirely. Note that this means that this state is separate from the rest of the finite state machine and thus does not progress on its own, the state must be manually changed.

4.3.1.3 Display Data

When writing the data to the LCD two loops are used to fill the first and second half of the display. This is done as the display stores the first and last eight characters from separate locations in RAM. Between the two loops, a command is used to shift the location of where data is being written. The command used for this transmitting a command byte: RS is low and The data is $(192)_{10}$.

x	y	z		B	P	M		V	o	l	:	a	b	c	d
BPMString								*VolString*							

Figure 4.2: The data format of the LCD

The data being pushed is placed in two arrays from another function separate to the FSM. This function takes in two integers, the BPM and volume values calculated using the ADC values. The format of the display is shown in Figure 4.2. Values x, y, and z are the BPM value as calculated from the ADC. Values a through d are the volume values, this is formatted as 'nnn%' when non-zero or 'Mute' when the volume is zero percent. For both of these cases, if the leading integers are zero they will be replaced with a space character rather than zero. The strings, or character arrays, are made sure to have a length of eight to ensure there is no misalignment of data or unwanted data being read. At the end of the function, the next state is set to twenty to write the new data to the LCD.

4.3.2 ADC Calculations

4.3.2.1 Direct Memory Access

Direct Memory Access (DMA) is used by multiple features on the μC to reduce the workload of the Central Processing Unit (CPU). DMA transfers a given array's data to a peripheral or from a peripheral into a variable without using CPU cycles. This can be set up to transfer in various lengths: byte, half-word, and word. Furthermore, the DMA can be set up to run linearly or circularly, this means that when the end of the buffer is reached it will either stop or restart reading respectively.
Figure 4.3 aids in explaining further concepts. This figure shows an example where the length of the buffer is 4096 samples and data is coming from the μC going to a peripheral. As the data is read from the buffer there are two callbacks that will occur. These are when the data read is halfway and when it has completed. In each case there will be a

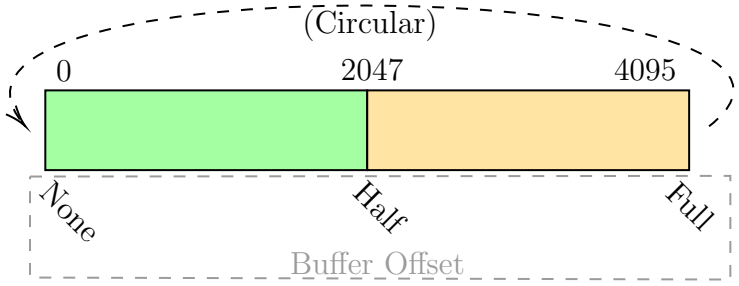


Figure 4.3: DMA Buffer Example

flag updated, the buffer offset, to reflect the current position. This flag can be used to start replacing data in the opposite half of the buffer when in circular mode. This implies that when the Half buffer offset is set the first, green, half of the buffer can get the data following on from the end of the second, orange, half.
DMA is used for the ADC readings. Once the transfer to the array of data coming from the ADC is complete the values in this buffer are moved into another array. This is done to prevent reading and writing to the same array location simultaneously resulting in corrupted data. These values from the second array are then used by the rest of the code. The DMA channel for the ADC is set up in circular mode transferring a full word. The full word is needed as the ADC is set up in 12-bit mode. As there are two channels of the ADC in use the two values will be placed consecutively in the desired array.

4.3.2.2 Manipulation of ADC Values

The two ADC channels measure the voltage from the two voltage divider circuits discussed prior. One channel will be used for the tempo measurement and the other for the master volume. The tempo measurement will be used to determine not only the BPM but also the milliseconds per beat. The latter of these is compared to the number of sysTicks from the previous update. sysTick is a timer integer that is incremented every millisecond. An integer is used to count a number of ticks until it is reset.

$$\text{mSPB} = 1000 * \frac{4096 - \text{ADC}}{4095} \tag{4.1}$$

Equation 4.1 shows the formula being used to determine when to run the matrix updating code as well as resetting the tick counter. In this equation, the 1000 is in place as this is how many ticks are in one second. The ADC value is subtracted from the maximal 12-bit value ($2^{12} = 4096$) and then divided by the same maximal value minus one. The reason for subtracting the ADC value from the maximum is due to the value read by the ADC being inversely proportional to the time taken to refresh, hence when the voltage divider is at its maximum value the time between each update should be at its minimum.
Once the matrix update code runs there are two more values required for the LCD. These are the BPM value and volume. To determine the BPM a similar equation to Equation 4.1 is needed.

$$\text{BPM} = 60 * \frac{4095}{4096 - \text{ADC}} \tag{4.2}$$

Equation 4.2 is the formula used for this exact purpose. This formula inverts the fraction from the mSPB formula as now it is beats per time instead of the other way around. The

value of the ADC is at most 4095, thus subtracting this from 4096 will never result in dividing by zero. The sixty in the formula is as there are sixty seconds in a minute, as is needed for Beats per Minute. If the value of the voltage divider is at a maximum, the BPM value will also be at its maximum.

$$\text{VOL} = 100 * \frac{\text{ADC}}{4095} \tag{4.3}$$

To determine the volume a simple percentage equation is used. Equation 4.3 shows exactly this. This ADC channel is also 12-bit, hence to get a fraction of the maximal value (4095) the measured value is simply divided by the maximum. To convert a fraction into a percentage value the fraction must be multiplied by one hundred. This outputs a value between zero and one hundred, exactly what is desired for a percentage value.

4.4 Audio File Reading and Playback

4.4.1 The WAV file format

To have a better understanding of what the μC must interpret when reading a .wav file the header of such a file must be understood as well as the basic structure. The wave format is a type of Resource Interchange File Format (RIFF) file with a single WAVE chunk. The WAVE chunk has two sub-chunks, 'fmt ' and 'data'. These contain the data format information and audio data respectively.

Pos.	Size	Endian	Field Name	Data in this Position
0	4	big	ChunkID	"RIFF"
4	4	little	ChunkSize	4 + (8 + SubChunk1Size) + (8 + SubChunk2Size)
8	4	big	Format	"WAVE"
12	4	big	SubChunk1ID	"fmt "
16	4	little	SubChunk1Size	16
20	2	little	AudioFormat	1 (PCM)
22	2	little	NumChannels	1 (Mono), 2 (Stereo), etc.
24	4	little	SampleRate	8000, 16000, 44100, etc.
28	4	little	ByteRate	SampleRate * NumChannels * BitsPerSample / 8
32	2	little	BlockAlign	NumChannels * BitsPerSample / 8
34	2	little	BitsPerSample	8, 16, 32
36	4	big	SubChunk2ID	"data"
40	4	little	SubChunk2Size	NumSamples * NumChannels * BitsPerSample / 8
44		little	data	Sound Data

Table 4.2: RIFF Header Format for WAVE files

Table 4.2 shows what the header of a RIFF file looks like for Pulse-Code Modulation (PCM) wave data. Items that are shown in double quotes ("") must be literal in the file at the specified location. For this explanation, only PCM will be considered. This is when there is a numeric value given to a time point's value of the analogue audio wave.

The sample and hold technique is used for PCM. This value has to be between 0 and 255 for 8-bit samples or between -32768 and 32767 in 2's complement for 16-bit samples. The sample rate is the speed at which the audio was recorded at and thus should be played back at. The value represents the number of samples in a given time, for this variable the value is in Hz, or per second. Given this, a sample rate of 16000 means that there are 16000 samples to be played in one second.
Audio samples will follow on each other consecutively, for two channels (stereo) a left sample will come first, then the accompanying right sample. Together they make one audio sample, this can be expanded to any number of channels.

4.4.2 Selected WAV file Settings

The used variables for the headers discussed prior are as follows. The audio files will be mono, one channel, at a 16kHz sample rate. The file's audio data is 16-bit PCM.
These choices were made to accommodate the number of other processes taking place on the μC as well as spacial restrictions in the μC's RAM. Further detail on the processing can be found in Section 4.4.4. RAM on the selected μC is only 128kB of which approximately 65kB is available for audio files according to the debugging software used.

4.4.3 Audio Driver

The driver for the CS43L22 DAC was adapted from a file and tutorial by Mohamed Yaqoob. This driver only allows for one sample rate that is set up when initializing the I^2S protocol. 16kHz is selected as this allows for enough detail to be heard in the audio stream. The I^2S is set up to be the master transmit using the Phillips standard with 16-bit Data on a 16-bit Frame. DMA is set up to transfer the audio from RAM to the CS43L22 DAC. As the audio stream will be continuous the DMA is set up in circular mode with the half-word (16-bit) data width for the 16-bit PCM data. I^2C is used to control the DAC using the default set up.
An adjustment to the driver was made for audio volume adjustments. The DAC expects volume to be between 52 and 280, that is a difference of 228.

$$\mathrm{VOL}_{DAC} = \frac{\mathrm{VOL} * 228}{100} + 52 \tag{4.4}$$

The volume in Equation 4.4 is a value between zero and one hundred, basically a percentage. This value is then scaled to be out of 228 and shifted up 52. This value is then written to $(20)_{16}$ and $(21)_{16}$ in memory on the DAC as this stores the master volume.

4.4.4 Reading and Using the Wave File

4.4.4.1 Connecting to the USB Drive

The μC is set up to act as a USB Host, as opposed to a USB Device, using the USB OTG File System together with FATFS. This set up allows for the μC to connect with a storage device to read and write data. The File System driver, FATFS, allows the μC to read files on the FAT Filesystem, thus requiring the USB Drive to be formatted as FAT32.
After all the peripherals are initialized, the μC will remain idle until a USB drive is inserted via the USB OTG connector. Once the USB drive has been mounted by the FATFS driver on the μC further actions will take place to initialize the audio files.

4.4.4.2 Initializing Audio Files

The initialization of the files takes place one cycle after the USB OTG drive is mounted. If the root folder of the USB drive can be read successfully the system will continue and look for the five audio files. A loop will step over each file for processing. Each file has the naming scheme of 'wave$N$.wav' where N is a number between zero and four. Each step in the loop will open the N^{th} file and read the header of the file, as described in Section 4.4.1. The file size, as well as the sample rate, are extracted from the header. The file size from the header is used to allocate the correct amount of RAM to store the audio file locally. Once space is allocated, assuming it was successful, the file's data will be read into the reserved space. The file is then closed after this step. Between each file being read a check is made to make sure that all the sample rates match, these should all be the same so that the same code can be used to play them back.
Once the files are all initialized the Audio Driver is initialized and started to allow for the audio buffer to be appended without having to stop and start the playback. A flag is set at this point as well to allow the main loop of the code to run (the Play Flag).

4.4.4.3 Cueing Audio Streams

Once the audio data has been prepared and saved in RAM the μC can cue audio files to play. As the indicator column progresses to the next column the now selected column's selected audio must be cued to be played back. An array is used to keep track of how much of each audio file has been read. As each row in the selected column's state is checked the data in the array is updated. If the LED is off in this location it can be considered selected and vice versa. When the node is selected the file will be cued by appending the array with the corresponding row's audio file size, halved. The reason for halving the file size is as the values are 16-bit but the file size value is 8-bit. Thus as samples are read as half words (16-bit) the remaining size actually indicated the remaining samples, not bytes. All nodes in the selected column that are not selected will have a zero appended in the array, essentially indicating the file has finished reading.

4.4.4.4 Filling the Audio Buffer

Separated from the loop that only runs once per millisecond is the filling of the DMA buffer for the audio. As the value of the buffer offset variable is set to either half or full, see Section 4.3.2.1, filling half of the DMA buffer begins. If the buffer offset is set to half the first half of the buffer can then be filled, similarly, when it is full the second half can be filled.
As a first step, the appropriate half of the audio buffer is set to zero so that number manipulation will not be skewed. A loop steps over every element in the array containing the remaining sample sizes. If the number of remaining samples is less than a quarter of the audio buffer's size (1024) the number of remaining samples is set to zero. This is not a problem as at most $1024/16000 = 0.064$ seconds will be ignored, this is short enough to not really be noticeable. If the number of samples is not zero a portion of them will be loaded into the audio array. Only a quarter the size of the audio array will be read from the samples as the samples must be duplicated to mirror the sound left and right in the audio array. Every copied sample is divided by the number of audio files, in this case, five. This allows for up to five audio tracks to be added together without clipping the audio. Furthermore, the volume of all the audio files will be normalized. After each

of the samples have been copied, the array element for the remaining samples is updated to reflect the number of samples read. Finally, after the audio buffer is completely filled the buffer offset flag is reset.

4.4.4.5 Reading Sounds from RAM vs Directly from USB

For optimal operation the sound samples will all be loaded into RAM on the μC, however, this places a limit on the length and number of samples that can be buffered. The current method has five separate wave files on the USB Storage Drive with a total combined size of 64kB. Each of these files is less than one second long to conserve space. Further set up information is found in Section 4.4.2. It is possible to play back longer samples than can fit into RAM by reading them directly from the USB drive during playback. Each sound file would have sixteen pointers to allow for the software to play back up to sixteen copies of the same audio data simultaneously. Using this technique would only require a pointer to be kept for every node on the matrix (80) in place of the complete audio files. Other than being longer, the files could also be stereo, a higher sample rate or even 32-bit PCM. It was found however that it is not possible to read simultaneously from five files using USB whilst still meeting the timing requirements of playing back the audio stream at 44.1kHz. Furthermore, the USB drive would have to be connected permanently throughout the usage of the drum machine which may prove inconvenient.

4.5 UART Communication

As mentioned before, the UART communication was designed to be used as a debugging tool with the opportunity to be used for MIDI output. In the final state of the drum machine, the UART driver is not used but is, however, initialized and set up. The μC's Universal Synchronous/Asynchronous Receiver/Transmitter (USART) peripheral is set up in asynchronous mode, becoming UART, at 9600 baud (Bd) with an 8-bit word length, one stop bit and no parity bits. Two DMA channels are also set up to handle the data coming in and going out to and from the μC. The channels are set up in circular mode as there will be roll-over between each transmitted or received character. The word size of the channels is in bytes (8-bit) as this is how the UART channel is set up. The serial communication will transfer American Standard Code for Information Interchange (ASCII) characters which are one byte each.

Chapter 5

Final Results

5.1 LED Matrix

It should be noted that when measurements of this driver were taken data was being sent incorrectly as rows are stepped over rather than columns.

5.1.1 Serial Signal to LED Driver

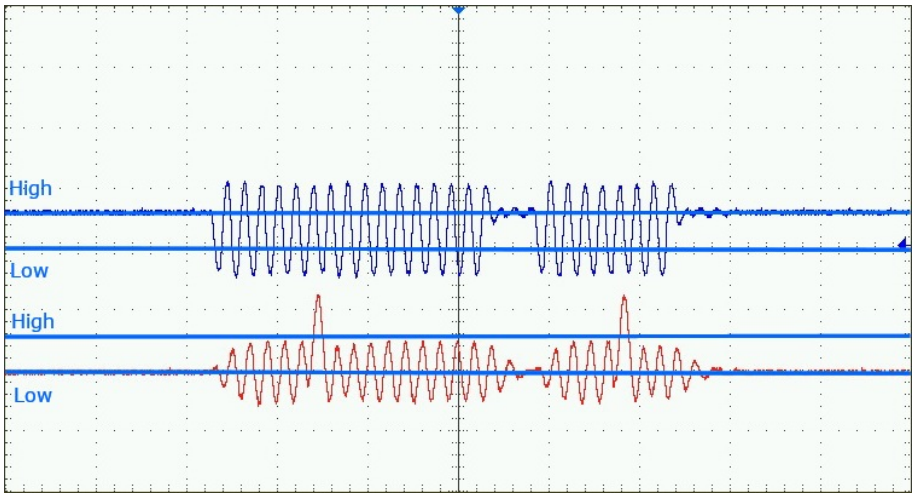


Figure 5.1: Serial Data and Clock Signal to LED Driver Circuit

Figure 5.1 shows the signal being sent from the μC to the LED driver circuit used. This signal is in the form of three consecutive bytes in serial. A signal that is not shown is the enable pin that is pulled low while the data is being sent, and otherwise is pulled high. The top signal (dark blue) is the clock used for the SPI transmission, the lower signal (red) is the data being sent in serial. As the data value is sampled at every rising edge of the clock pulse the data can be seen as $(0000\ 0100\ 0000\ 0000\ 0000\ 1000)_2$. This translates to the column data being $(0000\ 0000\ 0000\ 0100)_2$ and the row data being $(0000\ 1000)_2$ as the data is sent through as the lower column byte, the higher column byte, and the row byte. Thus, the signal being sent in this figure is lighting up the LED on the twelfth column and last row. The oscillations and overshoot seen in both signals can be attributed to the very high toggling speed of the SPI clock. This, however, does not cause an issue with the Shift Registers as the sampled voltages are below or above the minimum high and maximum low voltages.

5.1.2 LED Driver Output

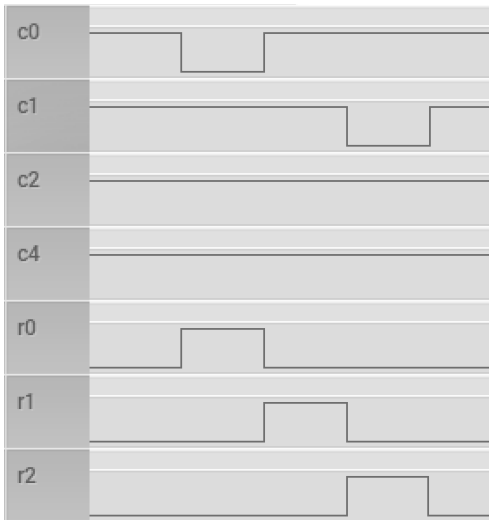


Figure 5.2: Output Signals from LED Driver

Figure 5.2 shows parts of the output from the LED matrix driver. Columns zero through four, and rows zero through two are shown. Two LEDs are selected, powered on, in this figure. These LEDs are at the first row and column (c0;r0), and the second column and third row (c1;r2). This figure not only shows the selected LEDs but also that the delay between each following state is negligible.

5.2 Button Matrix

5.2.1 Serial Signal to Button Driver

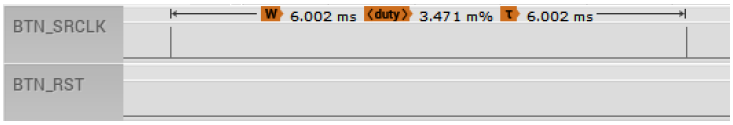


Figure 5.3: Serial Data to Button Matrix Driver

The driver for the button matrix simply requires a clock signal to increment, and a reset signal in the case it is needed. The signal used to increment the output state does not have to be at a near 50% duty cycle, a short pulse is sufficient. Figure 5.3 shows the time between each column incrementation for the button matrix, with no reset occurring. The time between one incrementation and the next can be measured to be 6.002ms with a duty cycle of $3.471(10)^{-3}$%, which works out to 208.33ns for the incrementation pulse.

5.2.2 Button Driver Output

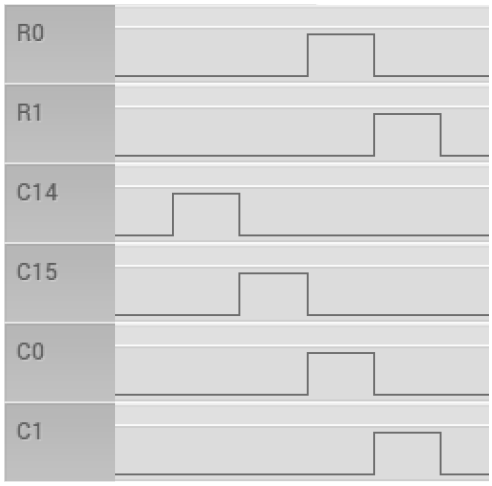


Figure 5.4: Button Driver Output and Selected Rows

As designed for in hardware, the selected column is incremented over and wraps around, once it reaches the final column, to the first column. Figure 5.4 shows not only the columns incrementing and wrapping correctly but also two rows with some depressed buttons. When a button is selected a high voltage can be measured on the appropriate row pins. A sample of these row pins is taken every time the columns are incremented. The selected buttons in this figure are therefore at the first row and column (R0;C0), as well as the second row and column (R1;C1). This can be shown as when column zero is selected (high) row zero gets pulled high, indicating a selected button. This technique is replicated for all time states (columns).

5.2.3 Timing Between Signals

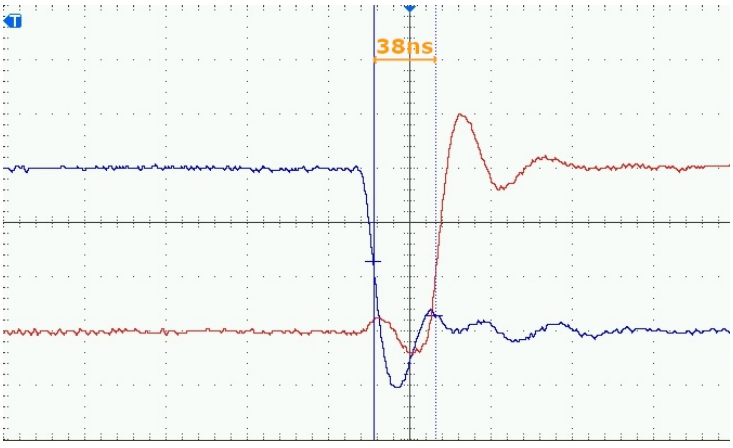


Figure 5.5: Timing for Button Driver between Column 15 and 0

As the longest delay between button states would be observed between column 15 and column 0 when incrementing, due to requiring the ICs to reset, the delay was measured in Figure 5.5. The time between column 15 going low and column 0 going high is measured to be approximately 38ns. This leads to this delay being considered negligible.

5.3 LCD

5.3.1 Initialization Sequence

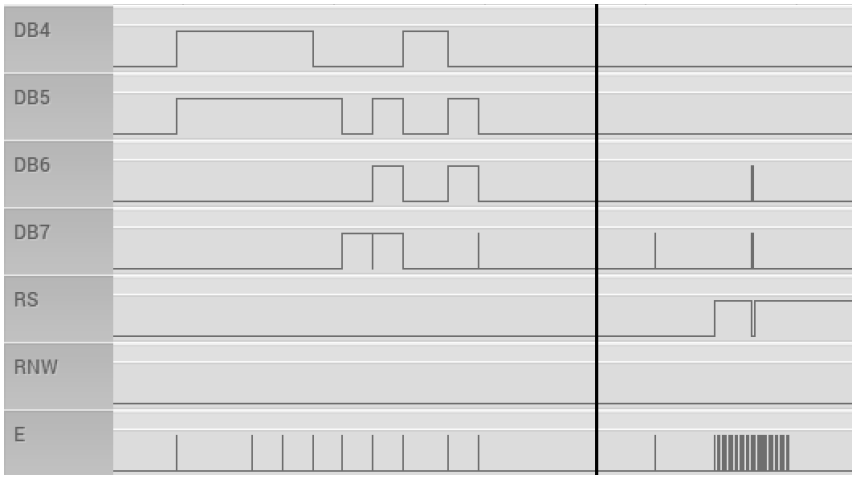


Figure 5.6: Initialization Sequence of the LCD

During the first milliseconds of the main software loop running, the LCD panel is initialized. The data being sent to the LCD and time between commands can be found in Table 4.1. This data can be visualised in Figure 5.6. The data nibbles that are transferred are visible as a parallel signal with the respective control signals below. The spikes shown in the row labeled E represent the data being sent according to how the controller on the LCD expects the data signals. The first data point to be sent over can be seen to be $(0011)_2 = (3)_{10}$ with RS = 0, this matches the required data as seen in the aforementioned table. The black vertical bar indicates a lapse in time as there is a large gap in this location due to the 100ms delay. The right-most data being sent is the initial display data, this is covered in more detail in the next section.

5.3.2 Screen Update

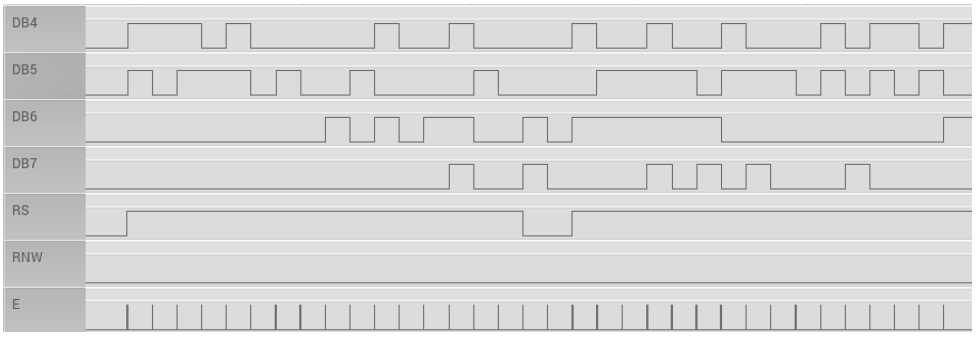


Figure 5.7: Data being sent to the LCD

Once the display has been initialized data can be sent from the μC. This data is reconstructed on the LCD and displayed. The signal shown in Figure 5.7 displays the message "120 BPM Vol: 81%". Each byte of ASCII data is split over two E toggles, better put, over two nibbles. Near the middle of the figure, RS goes low for the memory location change command.

5.4 UART Communication

5.4.1 Receiving Data

Figure 5.8: Receiving UART Data from the FT230XS

As the UART transceiver is unused no transmitted signal can be measured, however, data can still be sent from a PC to the μC. Figure 5.8 shows decoded text data coming into the μC from a PC. This message was sent at 9600 Bd with 8-bit data, one stop bit, and no parity bits.

5.5 Audio Data

5.5.1 I^{2}S Data

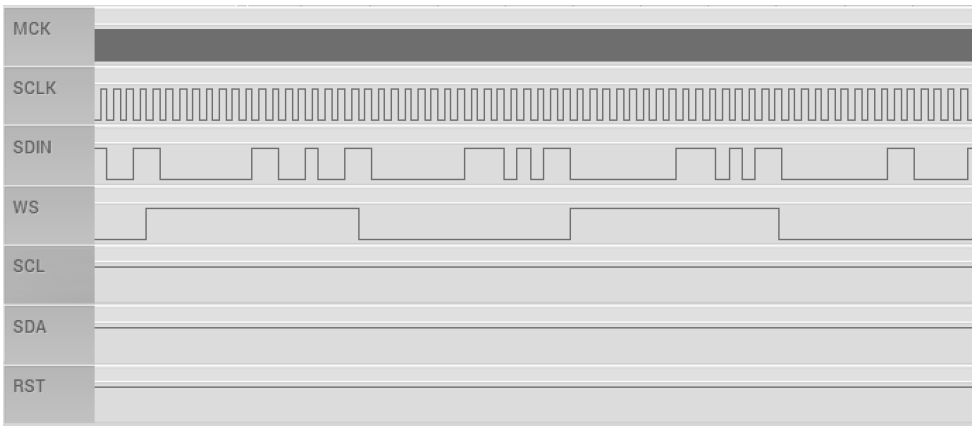


Figure 5.9: I^{2}S Data being sent to the DAC

Transmitting the PCM data from the μC to the DAC is handled by I^{2}S. The signals shown in Figure 5.9 represent the clocks, data and select used to transmit data over this standard. MCLK represents the Master Clock and is only used to synchronise internal DACs and ADCs, not for the data transfer itself, thus the much higher frequency. SCLK is the serial clock used to sample the data on SDIN line. WS, or word select, is used to indicate which channel is being transferred, low is Left Channel, and high is Right. The other pins are used in the I^{2}C standard, but as no settings on the DAC are being changed these all remain low in the figure. It should be noted that there are sixteen SCLK cycles per channel, as each PCM sample is 16-bit.

5.5.2 Audio Out

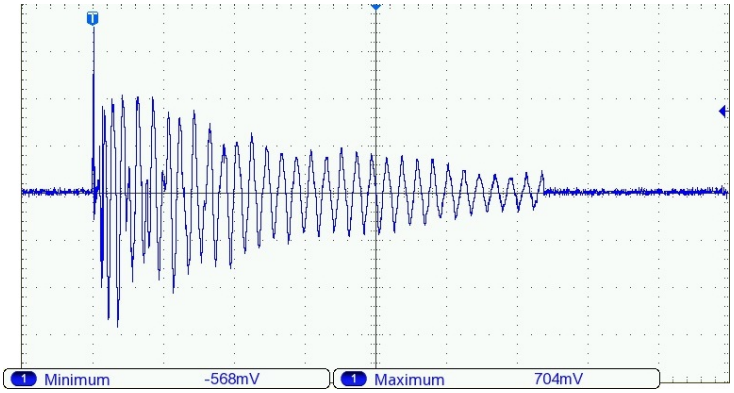


Figure 5.10: The Audio Signal Produced by the DAC

Figures 5.10 and 5.11 show the same audio file as generated by the DAC and shown on a PC respectively. The majority of the signal can be observed to be nearly the same, as is expected. The most notable differences, apart from the dead space before and after the audio in Figure 5.10, are the sound being cut off at the end, and a loud spike at the start of the audio. The audio stopping 'early' can be attributed to the DMA buffer for the audio never getting the last, up to, 1024 samples, thus ending the sound as it does. A further explanation of this can be found in Section 4.4.4.4. The other issue, the spike at the start of the audio, could possibly be attributed to the file header also being read as audio data. This header is short enough that the time it takes to play back is inaudible.

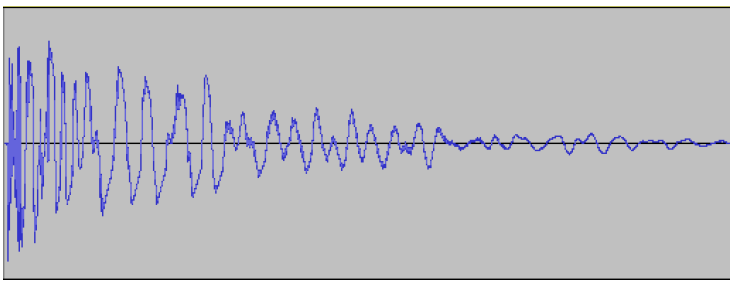


Figure 5.11: The Audio Signal Visualized on a PC

5.6 Complete Drum Machine

Figure 5.12 shows the Completed Drum Machine running with several beats selected. More detailed images can be found in Appendix E.

Figure 5.12: Completed Drum Machine Running

Button extenders and LED diffusers are not present in the figure above. This will be implemented by making these out of hot glue as this is a rigid material that diffuses light well.

Chapter 6

Conclusions and Recommendations

6.1 Outcomes

The Drum Machine functions correctly and satisfies all the requirements mentioned in Section 1.2. The system successfully displays the current tempo and volume to the user. These values can be changed within reasonable margins by making use of two potentiometers as voltage dividers. The matrix of LEDs and buttons is driven by only ten pins, greatly reduced from driving each row and column individually. The refresh rate of the LEDs and buttons is fast enough to not be noticeable to the user. A top panel was designed and 3D printed to fit the PCB with stand-offs. UART communication is possible. The tempo timing is accurate, allowing for correct scheduling. Playback of audio is clear and is scheduled as expected. The USB interface successfully transfers wave files to the μC for playback. Considering all this, the project is seen as a success.

6.2 Shortcomings

Though the Drum Machine does function some of the features have shortcomings. The rate at which the LCD data is updated is directly linked to the tempo of the system. This causes reduced responsiveness when the tempo is low. The maximum tempo is currently determined by an in-line resistor, this causes the limit to be approximate, a limit in software would prove more accurate. The current LED driver circuit design can be improved by driving up to sixteen LEDs simultaneously rather than the current five. The method to achieve this is mentioned in Section 3.1.4.2. The current software implementation reflects how the hardware should be set up, this is also incorrect here, limiting the maximal current to the LEDs. Pressing the buttons under the 3D printed panels is currently difficult as there are no button extenders and light diffusers. This can be implemented by using hot glue, this is further explained in Section 5.6. Perhaps the largest shortcoming is the limit placed on the size of the audio files. This could possibly have been reworked to be read dynamically, as explained in Section 4.4.4.5, allowing larger file sizes and more channels.

6.3 Possible Improvements

Even though the Drum Machine functions well, there are several improvements that can be made to make this a better product. Some of the items mentioned would improve the way in which the user will interact with the device, others would simply add more features.

To achieve the former several items could be integrated. Connections could be broken out on a separate PCB allowing the user to be able to plug in all necessary connections, such as the audio out, USB OTG drive port, and power in, in one location. This can then be installed at the rear of the device. An alternative method to power the device can be implemented, this would allow for more than 500mA to be used by the entire system. All of these additional features could be encased in a plexiglass, plastic, or metal enclosure that covers more than the top. The device could also be made thinner by incorporating all the hardware onto a single PCB rather than using daughter boards, such as the μC development board.

Further features that could be implemented on the drum machine could be some of the following. Flash memory could be incorporated allowing larger audio files to be saved on the device. Red-Green-Blue LEDs can be used to allow for differently coloured indicators. New software could connect via USB on a PC and manage the audio samples on the drum machine. The device could also support MIDI to connect to other MIDI devices. The potentiometers could be replaced with a single rotary encoder and selection buttons to change settings on the device. A more detailed display could also be used to indicate more settings or information about the selected sounds. A final feature that could be implemented is an on-board synthesiser, removing the need for actual sound files as the system could generate them locally.

Bibliography

[1] D. Orkin and N. Gustafson. (Oct. 4, 2018). Video: A bried history of the drum machine, Reverb.com LLC, [Online]. Available: https://reverb.com/news/video-a-brief-history-of-the-drum-machine (visited on 03/27/2019).

[2] I. Medina, "And the beat goes on: The story of the drum machine," Master's Thesis, California State University, Monterey Bay, May 2017. [Online]. Available: http://digitalcommons.csumb.edu/cgi/viewcontent.cgi?article=1098&context=caps_thes_all (visited on 03/30/2019).

[3] M. A. Schedel, "Anticipating interactivity: Henry Cowell and the Rhythmicon," *Organised sound: An international journal of music technology*, vol. 7, no. 3, pp. 247–254, Dec. 2002, ISSN: 13557718.

[4] O. Wang, "Hear the drum machine get wicked," *Journal of popular music studies*, vol. 26, no. 2-3, Jun. 2014, ISSN: 15242226.

[5] (Jul. 28, 2015). Eko computerhythm part3, [Online]. Available: https://www.youtube.com/watch?v=vSmROHIRSZ8 (visited on 03/30/2019).

[6] *Cr-78 service notes*, English, Roland, Japan, Jun. 20, 1979. [Online]. Available: http://www.synthfool.com/docs/Roland/Roland_CR78_Service_Notes_BW_medium.pdf (visited on 03/30/2019).

[7] *Tr-808 service notes*, English, 1st ed., Roland, Japan, Jun. 15, 1981. [Online]. Available: http://www.synthfool.com/docs/Roland/TR_Series/TR808/TR808_service_manual.pdf (visited on 03/30/2019).

[8] (). Maschine : Production systems : Maschine : Downloads — products, Native Instruments, [Online]. Available: https://www.native-instruments.com/en/products/maschine/production-systems/maschine/downloads/ (visited on 03/30/2019).

Appendix A

Project Planning Schedule

Week Of	Task Completed
04 Feb	Design Specifications
11 Feb	Design Specifications
18 Feb	Initial setup of ST board, CubeMX, GIT. Edited Example Code. Made Arduino Test Code.
25 Feb	Layout and PCB design.
04 Mar	PCB Design and Manufacture.
11 Mar	LED Driver Coding with PCB Tidy Up
18 Mar	PCB Tidy Up and Assembly
25 Mar	PCB Design Updated and LED Driver Coding
01 Apr	Button, UART and ADC (Tempo) Coded
08 Apr	LCD Coding and STM Audio Driver Added
15 Apr	Figuring out STM Audio Driver
22 Apr	File Reading from USB drive Coding
29 Apr	File Reading from USB drive Coding and Playing with STM Audio Driver
06 May	Changed Audio Driver and Pre-Load Audio Files
13 May	Top Panel Manufacture and Report Writing
20 May	Report Writing
27 May	Report Writing
03 Jun	Hand In and Oral/Poster Preparations
10 Jun	Oral Presentation and Poster Hand In

Table A.1: Simplified Project Planning Schedule

Appendix B

ECSA Outcome Compliance

ECSA ELO	Compliance
ELO 1. Problemsolving: Identify, formulate, analyse and solve complex engineering problems creatively and innovatively.	Several issues appear during the hardware design phase, each of which has multiple solutions proposed and explained. A notable example is that of driving large numbers of pins with as few pins as possible. This is identified in Section 3.1.2, various solutions are proposed and chosen from with thorough reasoning.
ELO 2. Application of scientific and engineering knowledge: Apply knowledge of mathematics, natural sciences, engineering fundamentals and an engineering speciality to solve complex engineering problems.	Mathematical concepts are used throughout the software in terms of bit-wise logic and formulas to generate certain values (such as for BPM or timers). Some values, such as the time between each button matrix incrementation, are chosen as an educated guess. Including noise suppression capacitors throughout the system allows for more accurate signals. The problem of reducing pin count with ICs can be seen as an open ended problem. A basic understanding of music and the usage of the device are also required.
ELO 3. Engineering Design: Perform creative, procedural and nonprocedural design and synthesis of components, systems, engineering works, products or processes.	The design of the most basic functionality of a drum machine was completed first. The time constraints do not allow for many additional features unless ahead of schedule. The research into what the product is was completed first. Design of hardware and software followed after that. The hardware design has several options for sub-systems to choose from, as discussed in Chapter 2. The device is legal to use and could benefit the client as it satisfies all their needs.

ECSA ELO	Compliance
ELO 4. Investigations, experiments and data analysis: Demonstrate competence to design and conduct investigations and experiments.	Measurements are made and reported on in this written work. The history and types of drum machines is explored in Chapter 2. During the development of the Audio Playback software registers of the microcontroller were manually read and compared to the appropriate manuals. Pins used to drive certain features had to be changed as the development board has on board pull down resistors, thus pulling the connected signal low. Logic analysers and oscilloscopes were used to analyse the signals produced by the various sub-circuits. Conclusions were taken from the results of each sub-circuit's behaviour as well as signals shown on the oscilloscope.
ELO 5. Engineering methods, skills and tools, including Information Technology: Demonstrate competence to use appropriate engineering methods, skills and tools, including those based on information technology.	Software is written for the selected microcontroller in embedded-C. This code controls all the communications between sub-circuits and handles all of the data. Oscilloscopes, logic analysers, and the debugging tool in Atollic TrueStudio allowed for signal states at various points of the system to be tracked and compared to expected results. Design tools, Autodesk Fusion 360 and KiCAD, are used to create the 3D printed panel and PCB respectively.
ELO 6. Professional and technical communication: Demonstrate competence to communicate effectively, both orally and in writing, with engineering audiences and the community at large.	The written work is structured as to allow the reader to follow along the thought process of the design in all phases: going from what the device is, to designing it in hardware and software, to finally testing it. Figures and tables are placed throughout the document where they seem most appropriate and to aid in the explanation of certain concepts or designs. The written work was completed in LaTeXand contains images from oscilloscopes and logic analysers. The oral presentation is presented in English with the use of a slideshow presentation and possibly the device itself.

ECSA ELO	Compliance
ELO 8. Individual work: Demonstrate competence to work effectively as an individual.	The entirety of this written work, design of hardware (including the PCB and 3D printed cover plate), design of software, measurement evaluation, oral presentation, and poster was all completed individually, with only minor guidance on possible changes given.
ELO 9. Independent Learning Ability: Demonstrate competence to engage in independent learning through well-developed learning skills.	Knowledge on how to design the PCB and cover plate was gained externally from the University. Further design ideas, as referenced, for circuits were also obtained, modified and applied in the final system. The history, and thus type of drum machine to build, was explored and used to create a greater understanding of the final product. Several mistakes were made and were corrected or mentioned, as well as many alternate designs for certain circuits being measured. The assumption of the STM provided audio driver to work was challenged when it had to be replaced with that of a third-party which was adapted for this product.

Table B.1: ECSA ELO Compliance

Appendix C

Circuit Diagram

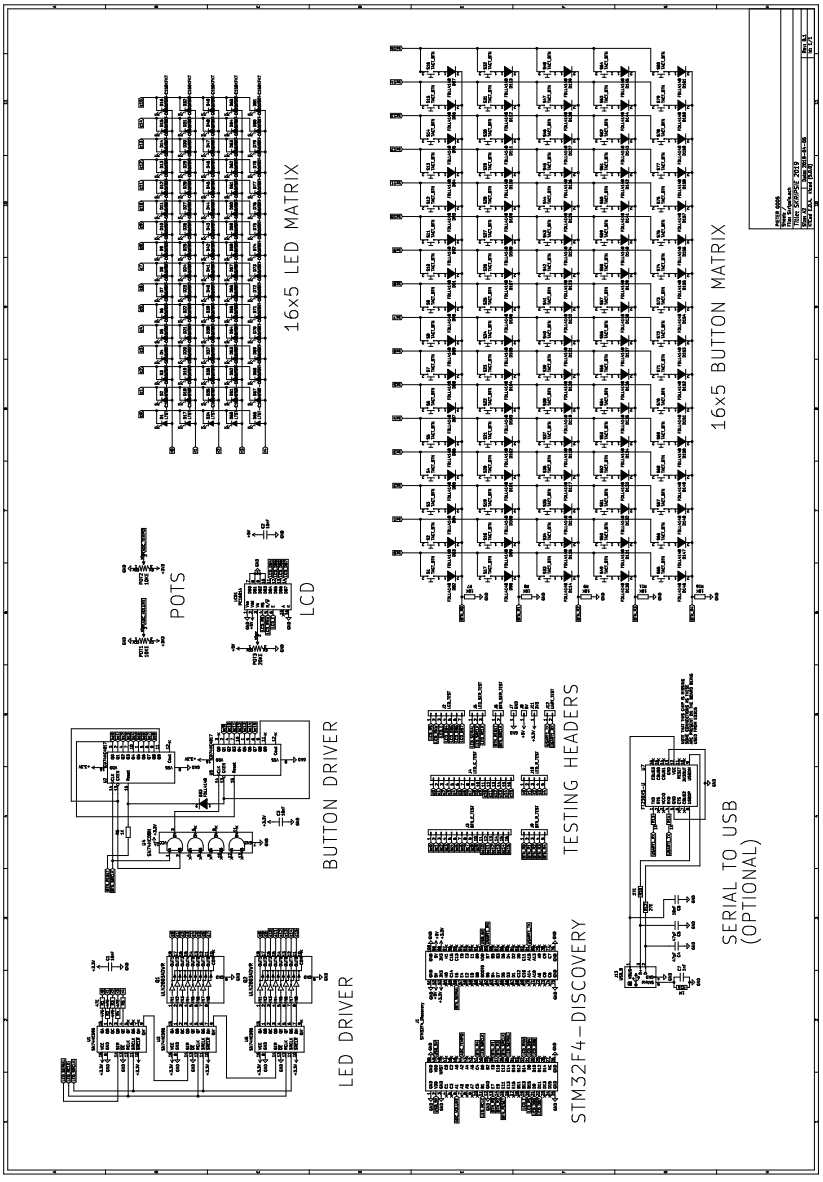


Figure C.1: The Final Revision (Rev. B.1) of the Circuit Diagram

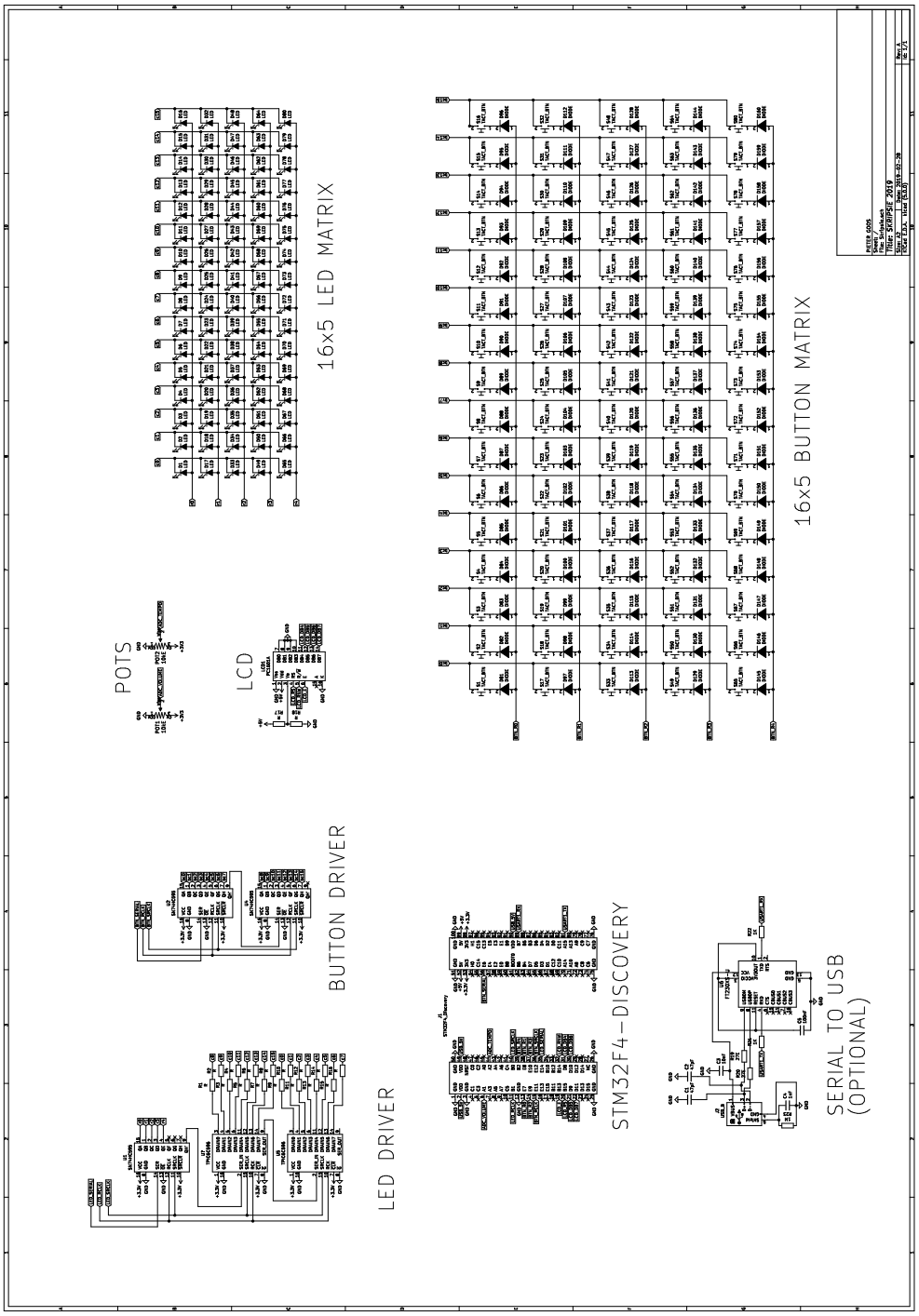


Figure C.2: Revision A of the Circuit Design

Appendix D

PCB Design

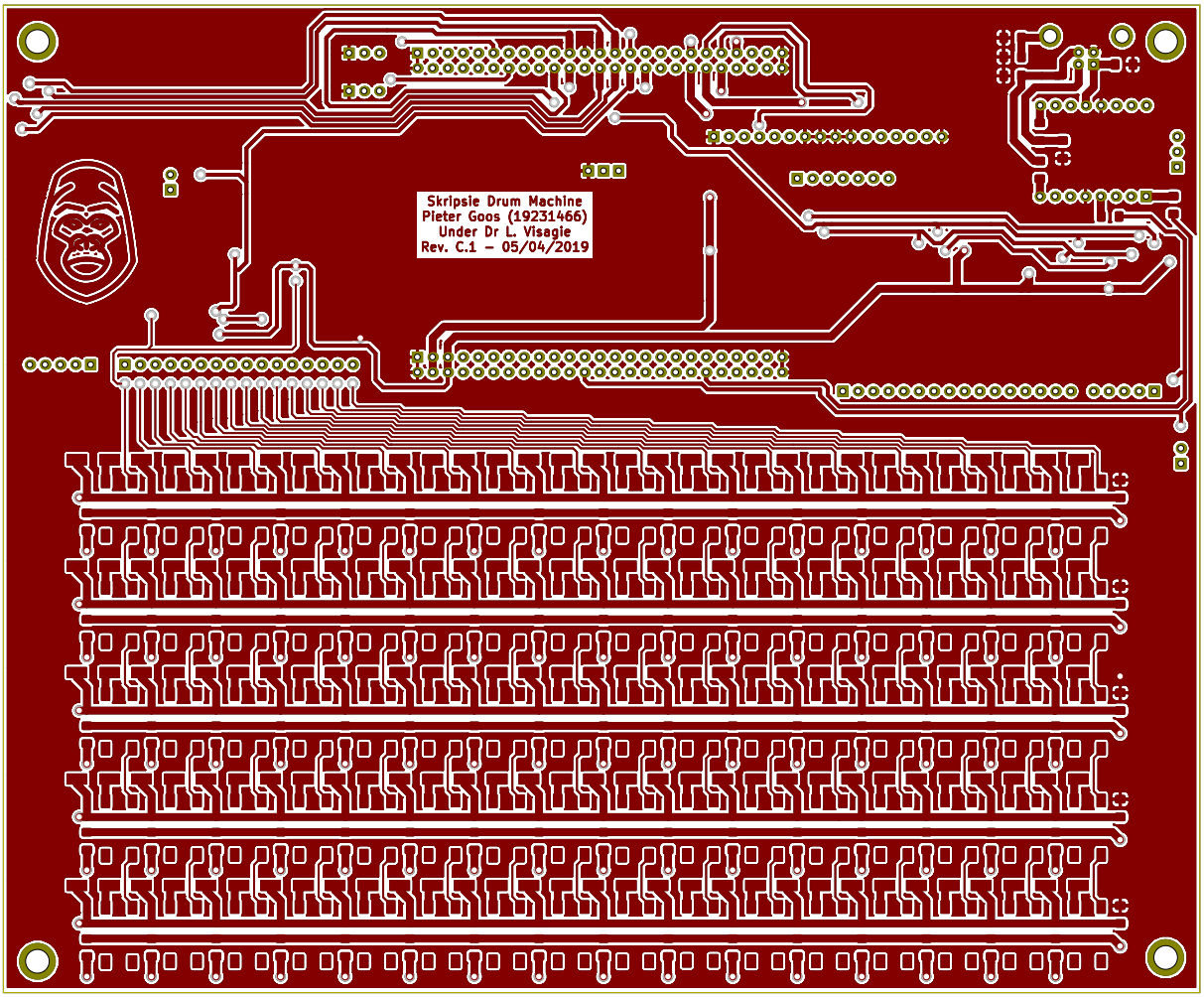


Figure D.1: Front Copper Layer of Revision C.1 of the PCB

It is worth noting that this board (See Figures D.1 and D.2) shown is Revision C, however the board used to construct this project on is Revision B. The changes made between the two versions are simply forgotten and moved traces, and mounting holes have been added. When looking at the reverse of the board (As in Figure D.2) it is worth noting that the shown diagram is not mirrored. This implies that the diagram shown's top left corner

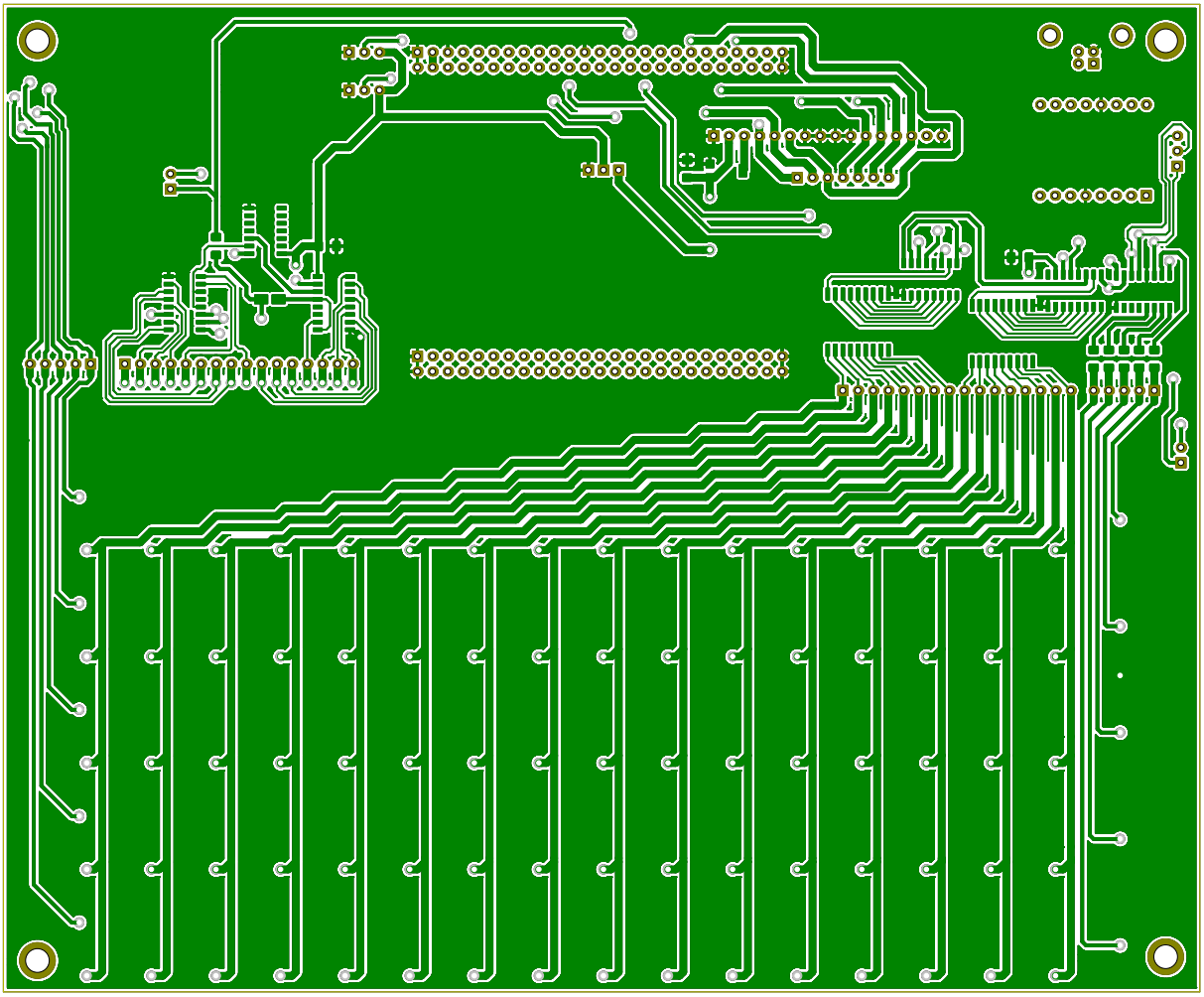


Figure D.2: Back Copper Layer of Revision C.1 of the PCB

matches that of the front copper layer, thus one would have to mirror this from left to right to see what you would see in reality.

Appendix E

Images of Completed System

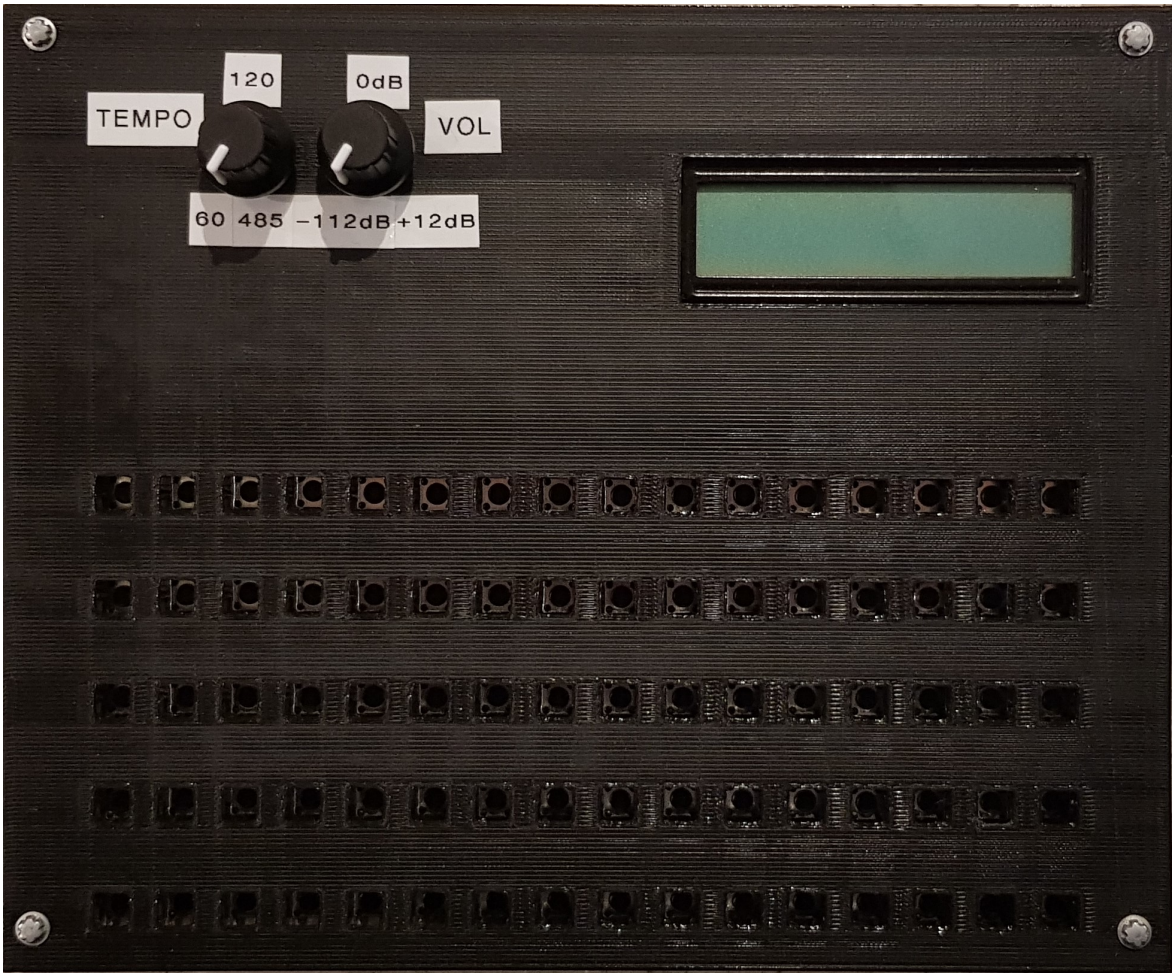


Figure E.1: The top of the completed Drum Machine with 3D Printed Cover

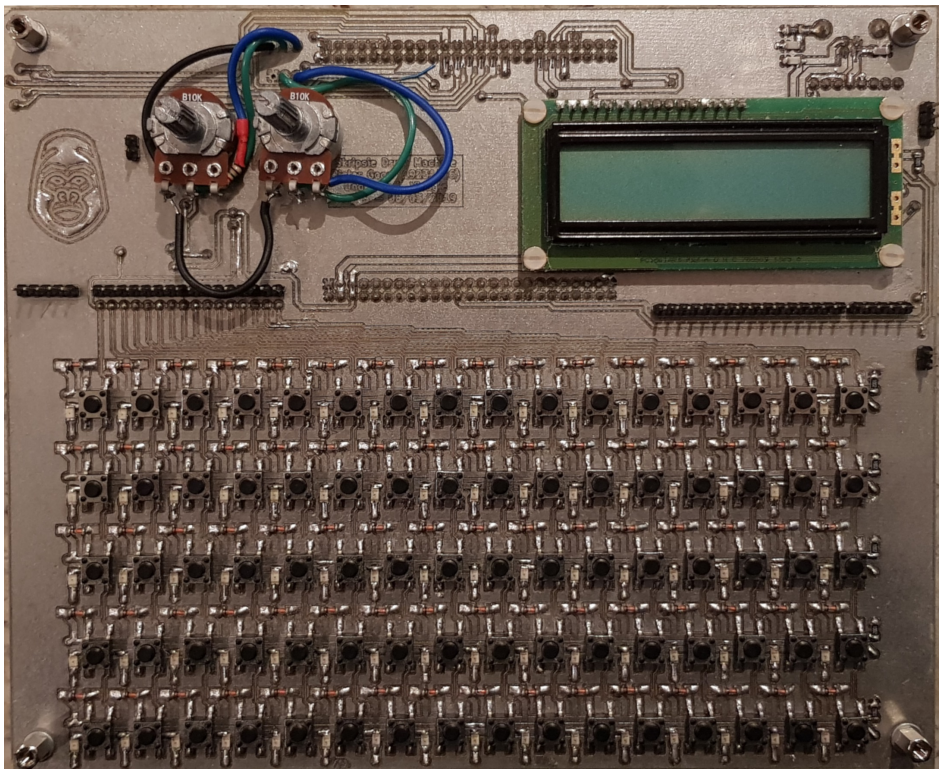


Figure E.2: The top of the completed Drum Machine without the Cover

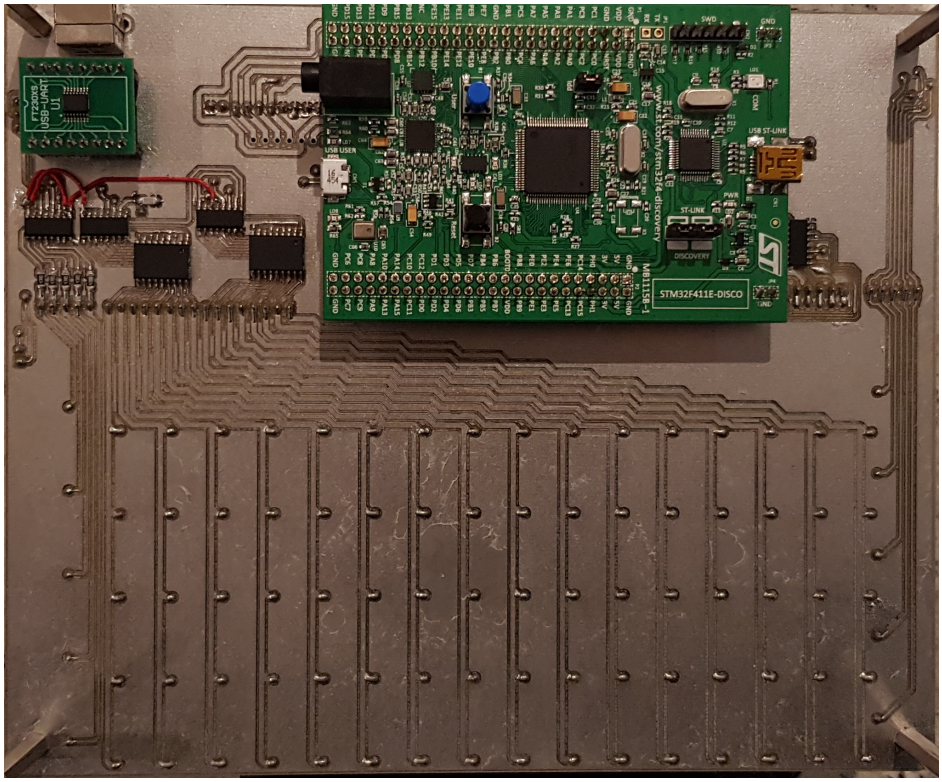


Figure E.3: The botton of the completed Drum Machine

Appendix F

Micro-Controller pinout

Pin		Type	Signal	Label
1	PE2	Input	GPIO_Input	DATA_Ready [LSM303DLHC_DRDY]
2	PE3	Output	GPIO_Output	CS_I2C/SPI [L3GD20_CS_I2C/SPI]
3	PE4	I/O	GPIO_EXTI4	INT1 [LSM303DLHC_INT1]
4	PE5	I/O	GPIO_EXTI5	INT2 [LSM303DLHC_INT2]
5	PE6	I/O		
6	VBAT	Power		
7	PC13	I/O		ANTI_TAMP
8	PC14	I/O	RCC_OSC32_IN	PC14-OSC32_IN
9	PC15	I/O	RCC_OSC32_OUT	PC15-OSC32_OUT
10	VSS	Power		
11	VDD	Power		
12	PH0	I/O	RCC_OSC_IN	PH0-OSC_IN
13	PH1	I/O	RCC_OSC_OUT	PH1-OSC_OUT
14	NRST	Reset		
15	PC0	Output	GPIO_Output	OTG_FS_PowerSwitchOn
16	PC1	I/O		
17	PC2	I/O	I2S2_ext_SD	
18	PC3	I/O	I2S2_SD	PDM_OUT [MP45DT02_DOUT]
19	VDD	Power		
20	VSSA	Power		
21	VREF+	Power		
22	VDDA	Power		
23	PA0	I/O	GPIO_EXTI0	WKUP
24	PA1	I/O	ADC1_IN1	ADC_VOLUME
25	PA2	I/O	ADC1_IN2	ADC_TEMPO
26	PA3	I/O		
27	VSS	Power		
28	VDD	Power		
29	PA4	I/O	I2S3_WS	I2S3_WS [CS43L22_LRCK]
30	PA5	I/O	SPI1_SCK	SPI1_SCK [L3GD20_SC/SPC]
31	PA6	I/O	SPI1_MISO	SPI1_MISO [L3GD20_AS0/SDO]
32	PA7	I/O	SPI1_MOSI	SPI1_MOSI [L3GD20_SDA/SDI/SDO]
33	PC4	Input	GPIO_Input	BTN_R4
34	PC5	I/O		
35	PB0	I/O	SPI5_SCK	LED_SRCLK

Pin		Type	Signal	Label
36	PB1	Output	GPIO_Output	LED_RCLK
37	PB2	I/O		
38	PE7	Input	GPIO_Input	BTN_R0
39	PE8	Input	GPIO_Input	BTN_R1
40	PE9	Input	GPIO_Input	BTN_R2
41	PE10	Input	GPIO_Input	BTN_R3
42	PE11	Output	GPIO_Output	BTN_RESET
43	PE12	Output	GPIO_Output	BTN_CLK
44	PE13	I/O		
45	PE14	I/O	SPI5_MOSI	LED_SERIAL
46	PE15	I/O		
47	PB10	I/O	I2S2_CK	CLK_IN [MP45DT02_CLK]
48	VCAP1	Power		
49	VSS	Power		
50	VDD	Power		
51	PB12	I/O	I2S2_WS	
52	PB13	Output	GPIO_Output	LCD_E
53	PB14	Output	GPIO_Output	LCD_RNW
54	PB15	Output	GPIO_Output	LCD_RS
55	PD8	Output	GPIO_Output	LCD_DB4
56	PD9	Output	GPIO_Output	LCD_DB5
57	PD10	Output	GPIO_Output	LCD_DB6
58	PD11	Output	GPIO_Output	LCD_DB7
59	PD12	Output	GPIO_Output	LD4 [Green Led]
60	PD13	Output	GPIO_Output	LD3 [Orange Led]
61	PD14	Output	GPIO_Output	LD5 [Red Led]
62	PD15	Output	GPIO_Output	LD6 [Blue Led]
63	PC6	I/O		
64	PC7	I/O	I2S3_MCK	I2S3_MCK [CS43L22_MCLK]
65	PC8	I/O		
66	PC9	I/O		
67	PA8	I/O		
68	PA9	I/O	USB_OTG_FS_VBUS	VBUS_FS
69	PA10	I/O	USB_OTG_FS_ID	OTG_FS_ID
70	PA11	I/O	USB_OTG_FS_DM	OTG_FS_DM
71	PA12	I/O	USB_OTG_FS_DP	OTG_FS_DP
72	PA13	I/O	SYS_JTMS-SWDIO	SWDIO
73	VCAP2	Power		
74	VSS	Power		
75	VDD	Power		
76	PA14	I/O	SYS_JTCK-SWCLK	SWCLK
77	PA15	I/O	USART1_TX	
78	PC10	I/O	I2S3_CK	I2S3_SCK [CS43L22_SCLK]
79	PC11	I/O		
80	PC12	I/O	I2S3_SD	I2S3_SD [CS43L22_SDIN]
81	PD0	I/O		
82	PD1	I/O		
83	PD2	I/O		
84	PD3	I/O		

Pin		Type	Signal	Label
85	PD4	Output	GPIO_Output	Audio_RST [CS43L22_RESET]
86	PD5	Input	GPIO_Input	OTG_FS_OverCurrent
87	PD6	I/O		
88	PD7	I/O		
89	PB3	I/O	SYS_JTDO-SWO	SWO
90	PB4	I/O		
91	PB5	I/O		
92	PB6	I/O	I2C1_SCL	Audio_SCL [CS43L22_SCL]
93	PB7	I/O	USART1_RX	
94	BOOT0	Boot		
95	PB8	I/O		
96	PB9	I/O	I2C1_SDA	Audio_SDA [CS43L22_SDA]
97	PE0	I/O		
98	PE1	I/O	GPIO_EXTI1	MEMS_INT2 [L3GD20_INT2]
99	VSS	Power		
100	VDD	Power		

Table F.1: Complete STM32F411VE Pinout