

DESIGN (E) 314  
TECHNICAL REPORT

---

# Digital Multimeter and Signal Generator

---

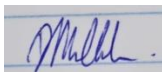
*Author:*  
Daanyaal Mullah

*Student Number:*  
22898220

March 28, 2022

## Plagiaatverklaring / Plagiarism Declaration

1. Plagiaat is die oorneem en gebruik van die idees, materiaal en ander intellektuele eiendom van ander persone asof dit jou eie werk is.  
*Plagiarism is the use of ideas, material and other intellectual property of another's work and to present is as my own.*
2. Ek erken dat die pleeg van plagiaat 'n strafbare oortreding is aangesien dit 'n vorm van diefstal is.  
*I agree that plagiarism is a punishable offence because it constitutes theft.*
3. Ek verstaan ook dat direkte vertalings plagiaat is.  
*I also understand that direct translations are plagiarism.*
4. Dienooreenkomstig is alle aanhalings en bydraes vanuit enige bron (ingesluit die internet) volledig verwys (erken). Ek erken dat die woordelike aanhaal van teks sonder aanhalingstekens (selfs al word die bron volledig erken) plagiaat is.  
*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.*
5. 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.  
*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.*



Handtekening / Signature

D Mullah

Voorletters en van / Initials and surname

22898220

Studentenommer / Student number

May 23, 2022

Datum / Date

### **Abstract**

The STMCube IDE provided an easy to use and good environment to write code for the device. The IDE was sufficient for the development of the device. I did make use of another person's code after demo two due to not being able to implement the ADC fully. The NUCLEO board is fast enough and capable of interacting with all the hardware components needed to build the digital multimeter/signal generator. Implementing the hardware components on the PCB was easy and by reading the datasheets made implementation and configuration straight forward. The I2C was not implemented due to time constraints.

## Contents

<b>1</b>	<b>Introduction</b>	<b>6</b>
<b>2</b>	<b>System description</b>	<b>6</b>
<b>3</b>	<b>Hardware design and implementation</b>	<b>7</b>
3.1	Power supply .....	7
3.2	Buttons .....	8
3.3	LED's.....	9
3.4	ADC .....	10
3.5	DAC .....	11
3.6	LCD .....	12
3.7	Current Sensor .....	13
<b>4</b>	<b>Software design and implementation</b>	<b>14</b>
4.1	Software Overview .....	14
4.2	Button Bounce Handling.....	14
4.3	UART Protocol and Timing .....	15
4.4	ADC Data flow and Processing.....	16
4.5	DAC Data flow and Processing.....	17
4.6	LCD Interface and Timing.....	18
<b>5</b>	<b>Measurements and Results</b>	<b>18</b>
5.1	Power Supply .....	18
5.2	UART Communications.....	19
5.3	Buttons .....	19
5.4	LEDS .....	20
5.5	ADC .....	20
5.6	DAC .....	21
5.7	LCD .....	22
5.8	Complete System.....	23
<b>6</b>	<b>Conclusions</b>	<b>23</b>
6.1	Shortcomings and Non-Compliances.....	23
6.2	Appendix.....	24

## List of Figures

1	Device Flow Diagram .....	6
2	Full Hardware Flow Diagram .....	7
3	5V Regulator Circuit .....	7
4	3V Regulator Circuit .....	8
5	Active Low Button Circuit .....	8
6	Button Layout.....	9
7	LED Circuit Schematic .....	9
8	Op-Amp Buffer Schematic.....	10
9	DC signal.....	10
10	AC Signals.....	10

11	Low Pass Filter Schematic .....	11
12	Op-Amp on PCB .....	12
13	LCD Schematic .....	12
14	LCD Layout on PCB .....	13
15	I2C Schematic .....	13
16	Main Software Flow Diagram .....	14
17	Button Bounce .....	15
18	UART Flow.....	16
19	ADC frequency .....	17
20	DAC Flow .....	17
21	LCD Software Setup .....	18
22	Power Regulator Circuit .....	19
23	Termite UART .....	19
24	Button Bounce .....	20
25	LED Schematic .....	20
26	ADC Test Setup .....	21
27	ADC Calibration Connections.....	21
28	DAC Test Setup .....	22
29	DAC Termite Commands .....	22
30	LCD.....	23
31	Pin Layout.....	24
32	PCB Layout .....	25

## List of Tables

33	Voltage Requirements .....	6
34	Pin Configurations.....	24

## List of Abbreviations

**LED** Light Emitting Diode

**GPIO** General Purpose Input Output

**LCD** Liquid Crystal Display

**I2C** Inter-Integrated Circuit

**UART** Universal Asynchronous Receiver-Transmitter

**ADC** Analogue to Digital Converter

**DAC** Digital to Analogue Convert

**PCB** Printed Circuit Board

**AC** Alternating Current

**DC** Direct Current

**TIC** Test Interface Connector

**IDE** Integrated Development Environment

## List of Symbols

*V* Voltage

*I* Current

*F* Frequency

$\Omega$  Ohms, Unit of measurement for resistance

F Frequency

R Resistance

$\neq$  Not equal to

# 1 Introduction

This technical report contains the explanations, methods, and motivations to design and build a device which functions as a digital multimeter and signal generator. The system is designed to be a multi-use device that will operate as a digital multimeter capable of measuring AC and DC voltage and current with 5% precision. It will also function as a signal generator being capable of generating AC, DC, and pulse signals with variable amplitude, offset, frequency, and duty cycle.

# 2 System description

The user will need a basic understanding of the STM32CubeIDE and basic soldering skills to build the device. The components needed to build the system are provided in table 1 below. The system uses the STM32 NUCLEO F303-RE microcontroller and a PCB to solder the other components on to. The PCB also provides tunnels to route connections to the board and provide power to components. The device should measure DC/AC voltage and current ranging from 0.1V to 2V and 0mA to 9mA respectively in multimeter operation mode. During signal generation, the device should generate DC/AC/pulse signals with variable parameters. The device will communicate by transmitting and receiving via its UART connection. It also has an LCD which will show information to the user about both signal generation and signal measurements. The LCD also provides an interface for the user to navigate through and measure specific parameters or set specific variable parameters for signal generation shown in figure 1. Debug LEDs are soldered on to the board to show specific state changes.

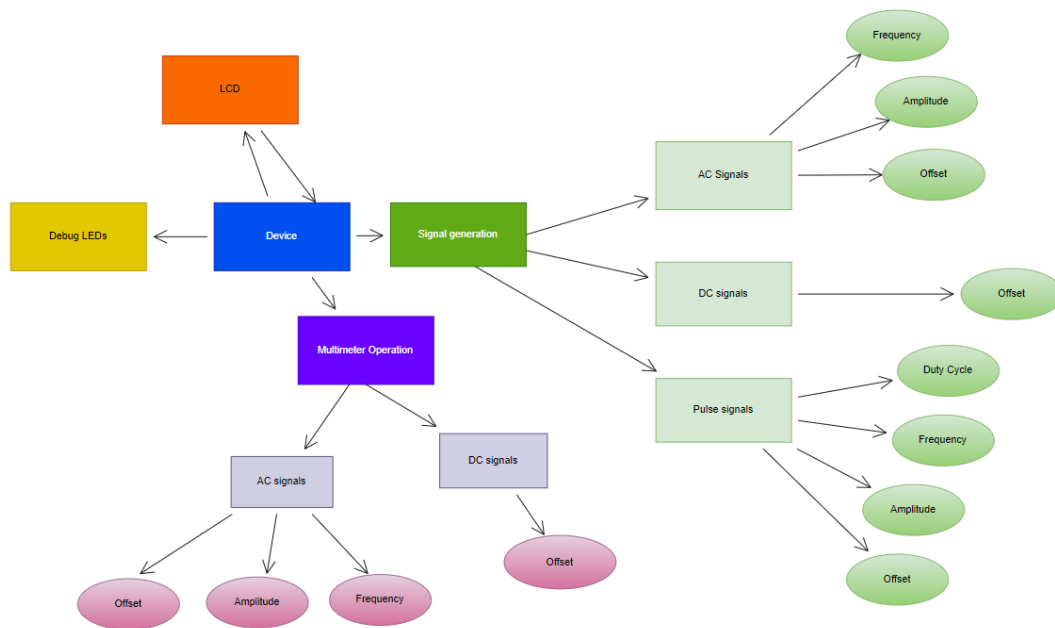


Figure 1: Device Flow Diagram

Table 1: Voltage Requirements

Component	Operating Voltage
STM32 NUCLEO	5 V
UART	3.3 V
LCD	5V
LEDs	3.3V
I2C	3.3V
LM2950	5V
L7805CV	9V
MCP602	3.3V

### 3 Hardware design and implementation

The device uses a 9V external power supply. The power supply can either be a 9V batter or a power bench provided in the labs. The 9V power is then routed into a regulator circuit which regulates the 9V supply into 5V. The 5V is used to power the NUCLEO, the LCD, and the 3.3V regulator circuit. Both the 3.3V and 5V are fed through jumpers to the entire PCB to provide power to the different components used to provide functionality for the device. The NUCLEO provides power to the debug LEDS and the UART. The 3.3V regulator circuit provides power to the active low buttons, the I2C module and the Op-Amp. All these connections are fed to the TIC so that it can verify measurements and send signals to the ADC for measurements.

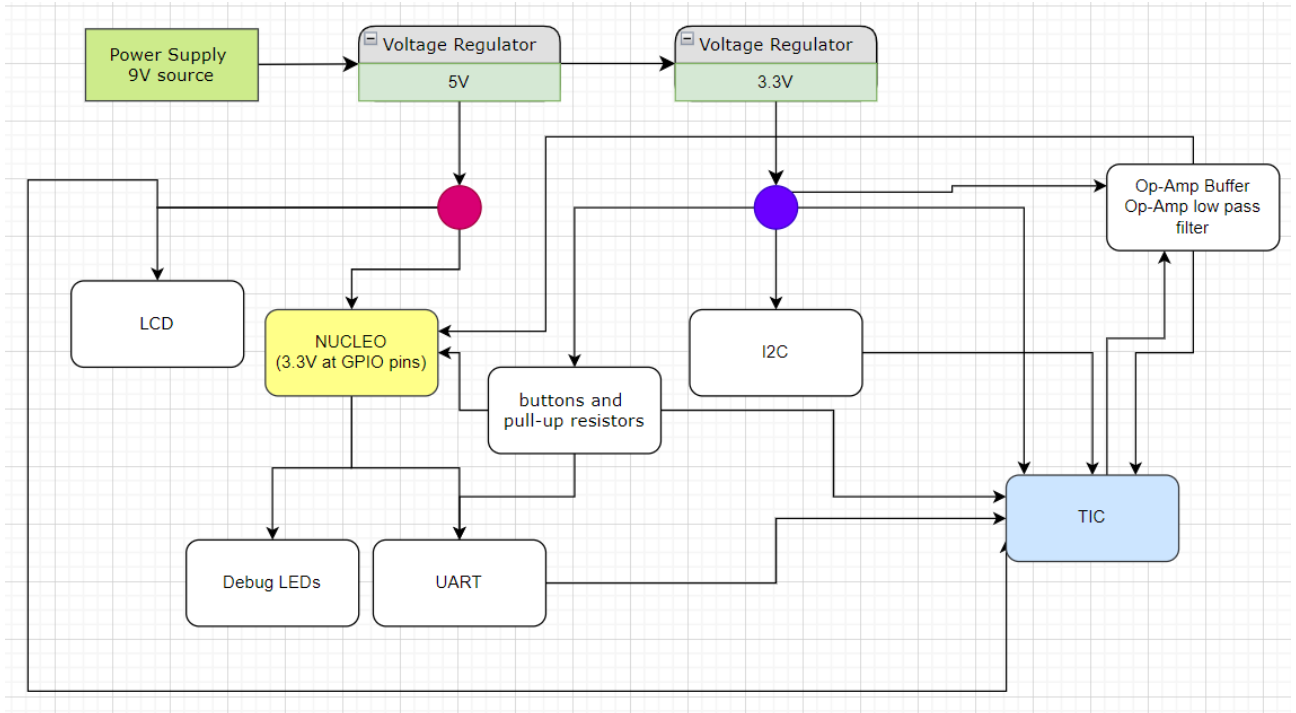


Figure 2: Full Hardware Flow Diagram

#### 3.1 Power supply

The device is powered using a nominal 9V battery source. During development, the device can be powered using the bench power supply. A 5V and 3.3V regulated supply is needed as some components require 5V and others 3.3V to operate. A standard LM7805CT regulator is used to regulate the 9V down to 5V. The 5V power is then routed via three jumpers to provide connections on the PCB to supply power to other different components like the LCD. A LED is connected on the PCB to show that the regulator circuit is working. Input capacitor C1 is used to reduce source impedance. The output capacitors are used to increase stability. The capacitors perform a low pass filter function helping smooth out the high frequency noise.

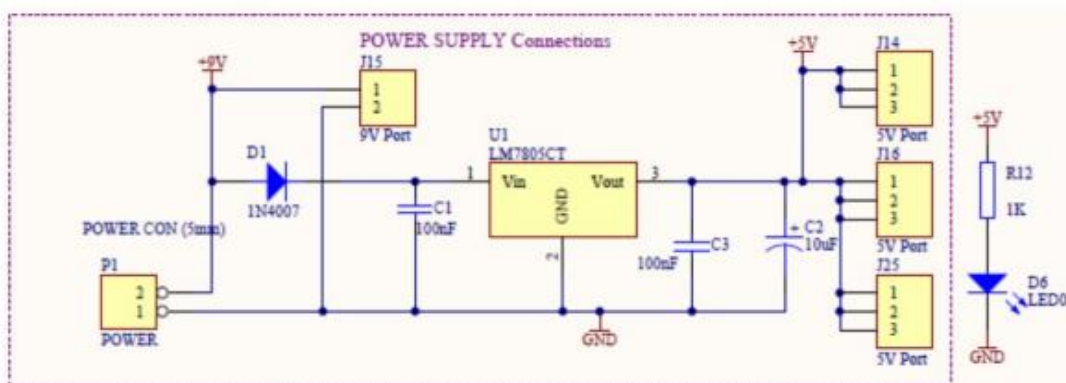


Figure 3: 5V Regulator Circuit



The 5V power is then routed to the 3.3V regulator. A LM2950 regulator is used to implement the 3.3V regulator. The following circuit diagram shows how the 3.3V regulator was implemented.

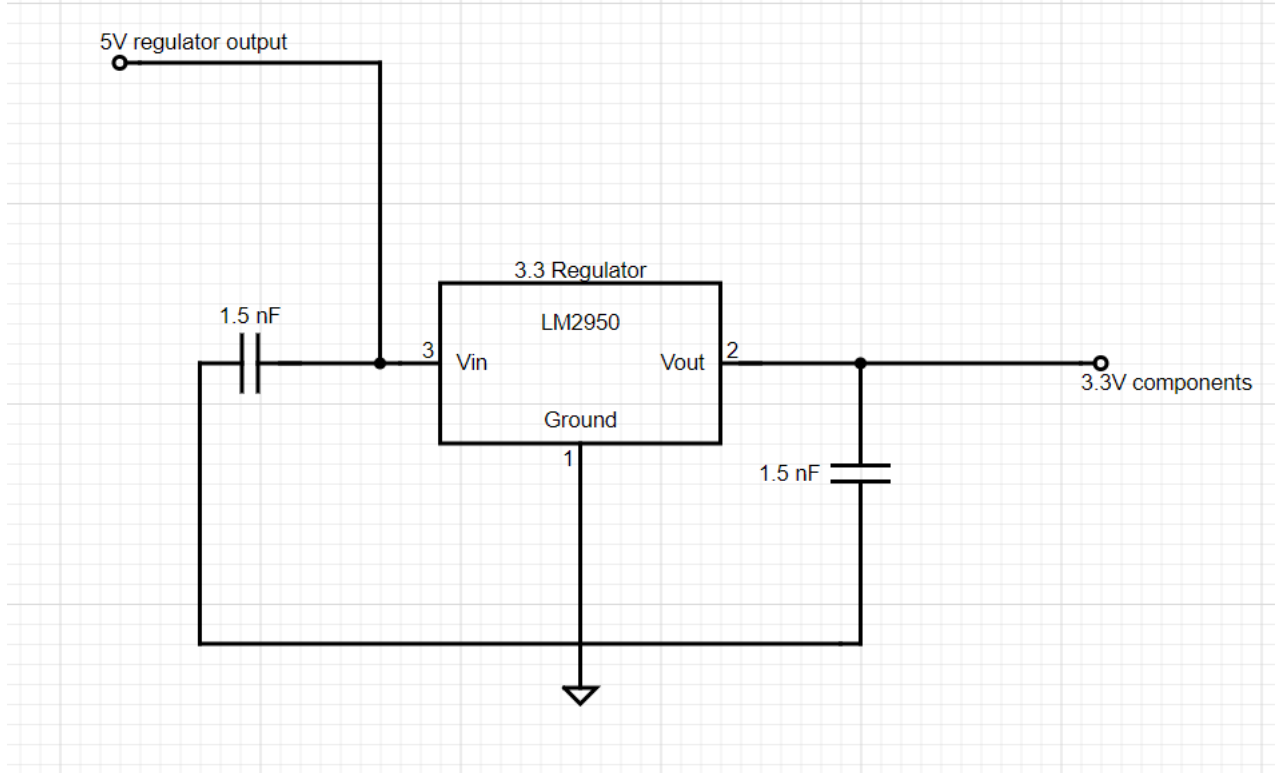


Figure 4: 3V Regulator Circuit

## 3.2 Buttons

Five buttons were implemented to navigate through the menu and display state. The middle button was used to toggle between the menu and display state and enter in values for the DAC and ADC. The down and up buttons were used to traverse through the menu tree and change the value of the DAC and ADC. The left and right buttons were also used to traverse though menu options. The push buttons are implemented in an active low state, meaning when the button is active it drives the GPIO input pin low. A pull up resistor is needed to drain the current as the max current allowed to enter the GPIO pin is 8mA. The resistor can be implemented in hardware or software. The pull-up resistor was implemented in software to save space on the PCB. Implementing in software is also simpler and quicker than building the circuit on the PCB.

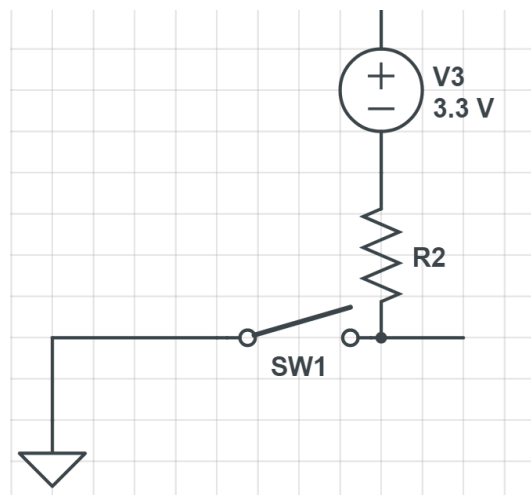


Figure 5: Active Low Button Circuit

The buttons can be implemented using any GPIO pin that can be configured as an interrupt. I chose the following pins for the 5 buttons implemented. Pin A0, A7, B6, B8 and B9.

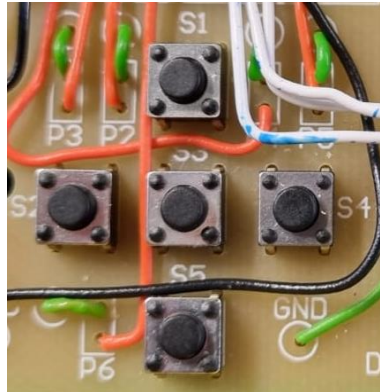


Figure 6: Button Layout

### 3.3 LED's

LEDs were used for debugging and to show what state the system is in. Five LEDs were implemented in the system. The first LED, D1 was used to show that the power regulator circuit was functioning. Each of the four remaining LEDs would go on or off depending on the systems specific state. LED D2 would turn on if the system is in menu state. D3 would be on if the system is in measurement and output state. D4 toggles on if the system is measuring AC or DC current. Finally, D5 would be on if the system was outputting a signal. A basic LED circuit design is implemented using a resistor and LED in series. The LEDs are connected to the GPIO pins of the NUCLEO board, however the current needs to be limited. The GPIO pins have an output of 3,3V. The LEDs display acceptable brightness when the current is between 4-6mA. The LEDs were manually measured to have a turn on voltage of 1.79 V using a multimeter. The voltage drop over the resistor must therefore be  $V_r = V_{gpio} - V_{led} = 3.3 - 1.79 = 1.51$  V. The resistance can then be calculated.

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

$$R = \frac{1.51}{5mA} = 302\Omega$$

Therefore a resistor of 330Ω was used. The diagram shown in figure 1 displays how the LED was set up. I used GPIO pin A8, B4, B5 and B10 for the four LEDs. The pins are close to the LEDs on the PCB and therefor allow for neat and close connections.

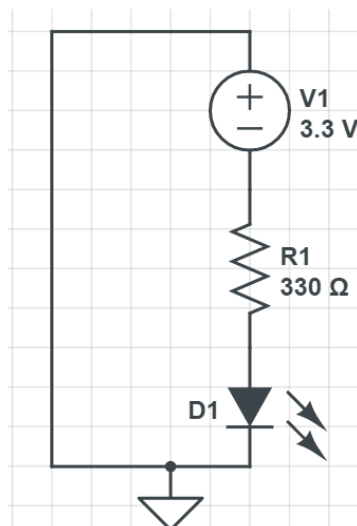


Figure 7: LED Circuit Schematic

### 3.4 ADC

The system can measure the voltage of two different types of signals (DC and sine) applied at the GPIO pin. The voltage range of the ADC input is from 0V to 3.3V but the input voltage range is from 0.1V to 2V. Therefore the system can measure DC and sine signals from 0.1V to 2V. It can measure the offset of DC signals and the offset, frequency, and peak-to-peak voltage of AC signals. An operational amplifier input buffer was used to protect the NUCLEO board from damage in case of a sudden spike in input signal. The Op-Amp was implemented as unity gain and the circuit diagram shows how the Op-Amp was configured. The Op-Amp chosen was a MCP602. The input is connected to the TIC and the output must be connected to any GPIO pin that can be set up as an ADC connection. I chose pin PA01 as it can be configured as an ADC pin connection and was situated close to Op-Amp on the PCB. The system outputs the measurements using the UART and LCD screen.

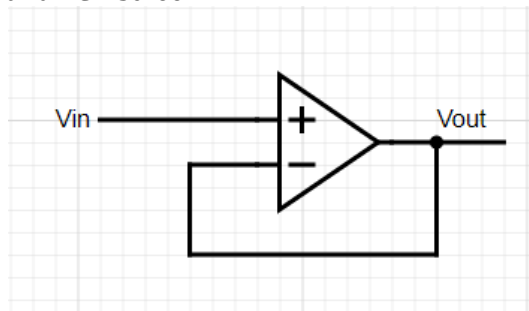


Figure 8: Op-Amp Buffer Schematic

The types of parameters our ADC must measure are shown below figures.

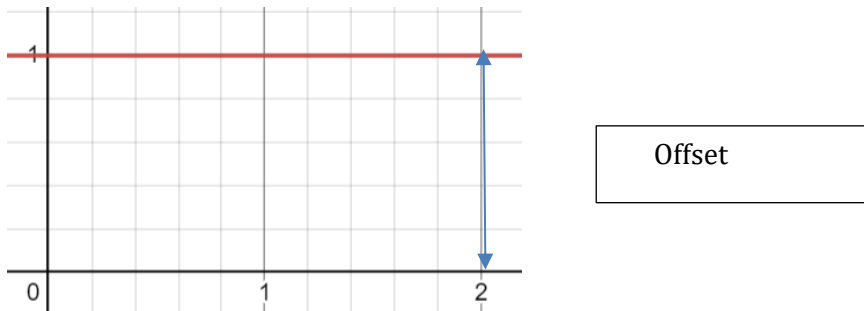


Figure 9: DC signal

DC signals only have an offset

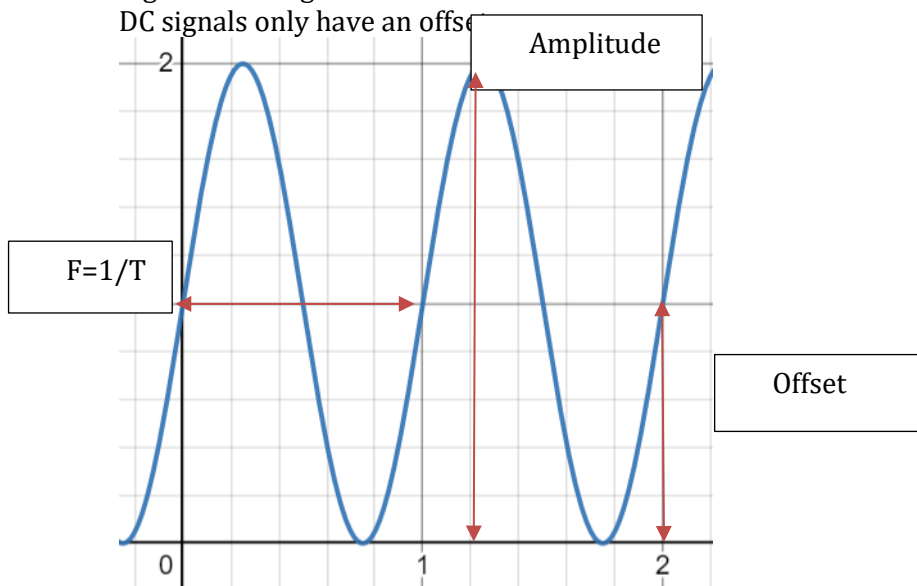


Figure 10: AC Signals

The parameters that the system measures for sinusoidal signals are frequency, amplitude, and offset. Shown in figure 10.

### 3.5 DAC

The system is capable of outputting three types of signals. A DC signal with variable offset that the user can adjust the signal from 0.1V to 3.1V. An AC signal with peak-to-peak amplitude of 0V to 3.1V, offset of 0.1V to 3.1V and a frequency ranging from 10Hz to 5kHz. Finally in signal generation mode the system can generate a pulse signal with offset between 0.1V to 3.1V and duty cycle from 0% to 100%. The frequency of the pulse also ranges from 10Hz to 5kHz. The generated signals are accurate to 5% of the input value in the correct ranges. The values and parameters for the signals that the system will output are taken either from the UART or the LCD where the user uses buttons to change the values/parameters. The DAC was implemented using GPIO pin A4 however any pin which can be configured as a DAC output could be used. The signal was output through a RC filter op-Amp circuit to smooth out the sinusoidal signals being output by the DAC. The same MCP602 Op-Amp was used for the DAC as was used for the ADC. The gain of the circuit was given by the following equation.

$$Gain = 1 + \frac{R_1}{R_2}$$

$$Gain = 1 + \frac{5.6k\Omega}{5.6k\Omega} = 2$$

The resistor and capacitor values are chosen for the low pass filter. A low pass filter is used to filter out high frequencies to smooth out AC signals. This does affect the pulse signals. By selecting a cut-off of 8000Hz and choosing the resistor value to be 10k $\Omega$ .  $\tau$  was calculated to be 20 $\mu$ s. The capacitor value was calculated to be 2nF. The transient response could then be calculated.

$$F = \frac{1}{2\pi\tau}$$

$$\tau = RC = (10000\Omega)(2nF) = 20\mu s$$

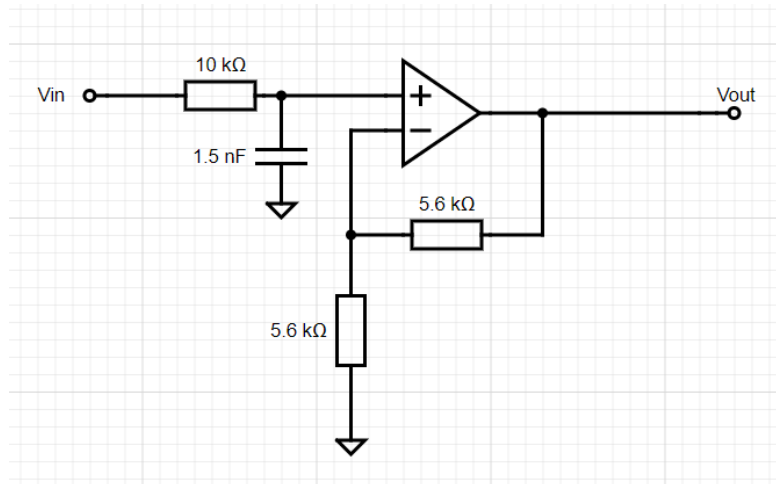


Figure 11: Low Pass Filter Schematic

Figure 12 shows how the circuit was built on the PCB. The Op-Amp needs connections to ground and the 3.3V connections. Its output also connects directly to the TIC.

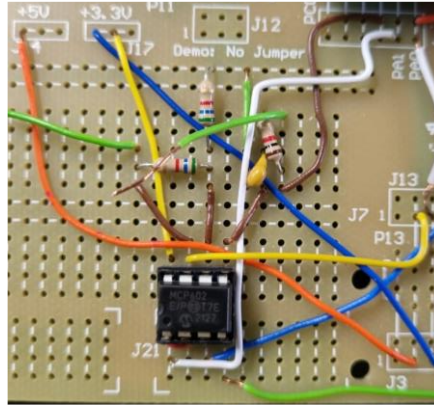


Figure 12: Op-Amp on PCB

### 3.6 LCD

An LCD is implemented on the device to display relevant information to the user. It displays active measurements, the generated signals, values of both the generated signals and measured signals, and finally a menu which allows the user to navigate through all the manual input configurations. The LCD is operated in 4-bit nibble mode. For this device the LCD chosen was the 1602A which is a 16x2 character display which is compact and provides the required space for all the information we need to show. Scrolling can also be implemented on the LCD. It is powered with a connection to 5V jumper. 7 GPIO pins are required for the LCD. The A and K terminals also need to be connected as it is difficult to read the LCD without the backlight turned on. A resistor must be connected in series as to limit the current below 10 mA. The value for the series resistor is chosen using the following equation.

$$R > \frac{V - \Delta V}{I} > \frac{5 - 4.1}{10\text{mA}} > 90\Omega$$

By increasing R, the backlight will be less bright and by decreasing it you can increase the brightness. The minimum value for the resistor which still has a current less than 10mA is 90Ω. However, I found the best resistor value to be 330Ω as the text was readable. It is acceptable to use any resistor value if it is above 90Ω and it's still easy to read the LCD.

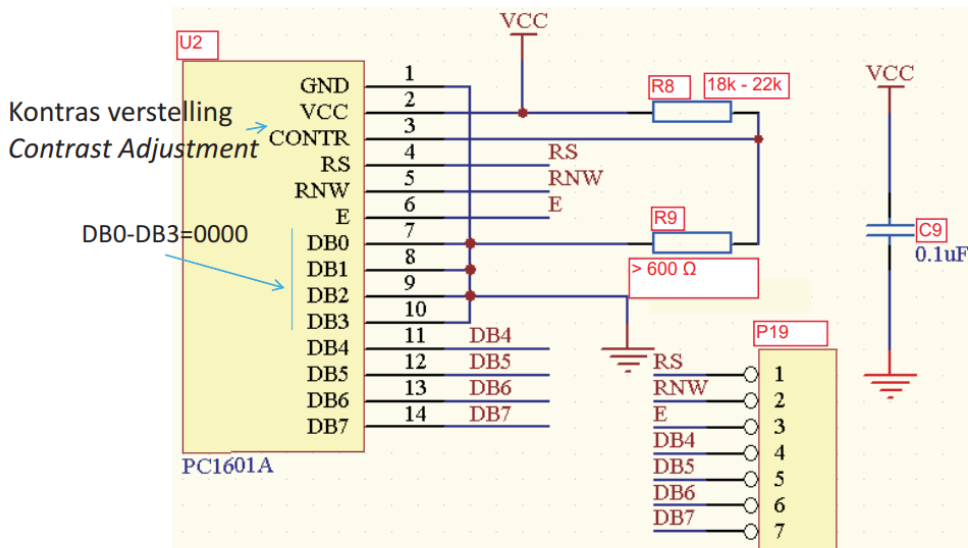


Figure 13: LCD Schematic

The values for R8 and R9 are responsible for the contrast. By adjusting these values, you can make the LCD more readable for the user. For R8 and R9 resistor values of 18 kΩ and 3.3kΩ were chosen respectively. These values make the screen easy to read and corresponded to the design document for the LCD. Capacitor C9's value was assigned as 0.1μF and therefor was used on the device as well. The

GPIO pins chosen were PC8, PC6, PA11, PA12, PB11, PB12 and PB13. All were setup as GPIO-outputs and were chosen due to being situated close to the LCD on the PCB.

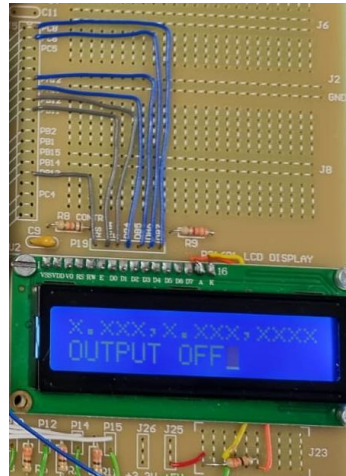


Figure 14: LCD Layout on PCB

### 3.7 Current Sensor

The system uses an INA219B device to measure the current of an input signal. The module is mounted on to the baseboard and requires 3.3V input power supply. As the input voltage of the system is limited to 2V, the current which the system must measure must be limited to below 10mA. Due to the TIC having limited pin connections an external controlled load which can measure the current consumption is not used, Therefore the input voltage signal to drive a fixed value load resistor is soldered on the PCB. The current sensor is connected to the Op-Amp and fed into the ADC so that it can be measured.

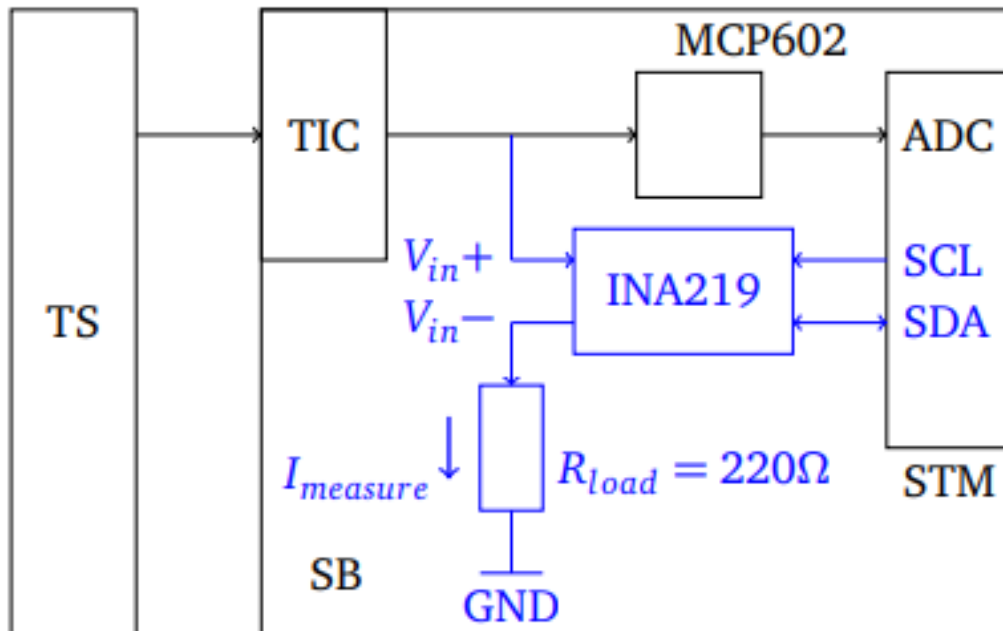


Figure 15: I2C Schematic

## 4 Software design and implementation

### 4.1 Software Overview

My entire program was coded around the main while loop. The while loop is continuously executing and if a flag is generated it quickly recognizes it. The flow of the program can be seen in figure 16. When a specific flag is generated depending on the systems state the while loop will call the appropriate function to execute. For example, if an input is sent from the TIC to output on the DAC, a flag called outputOn is set to 1. Then the while loop checks to see if the outputOn = 1 and if it is, it will call a function which will turn the DAC on and output a signal. The flow diagram below describes how the flags are generated. The entire pin configuration can be seen in table 2.

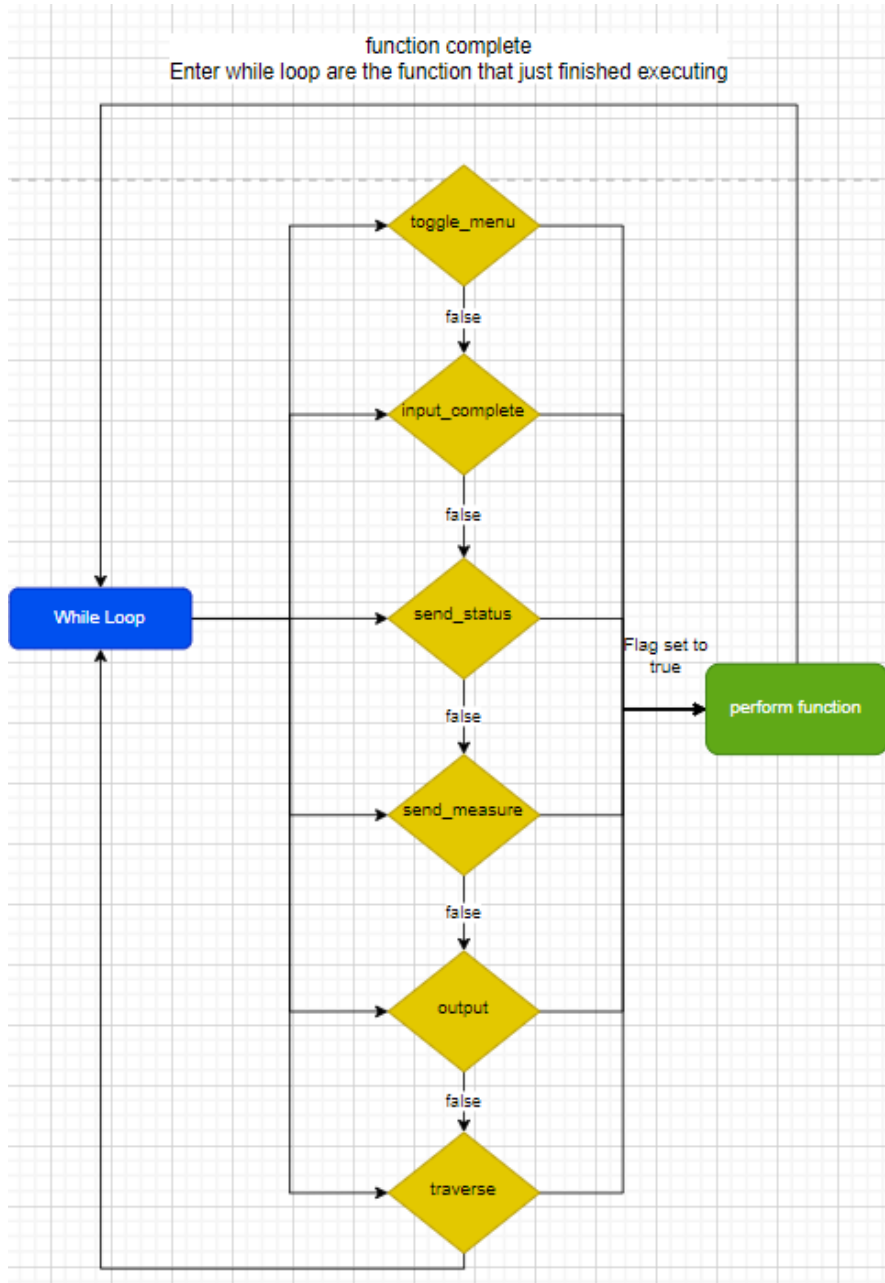
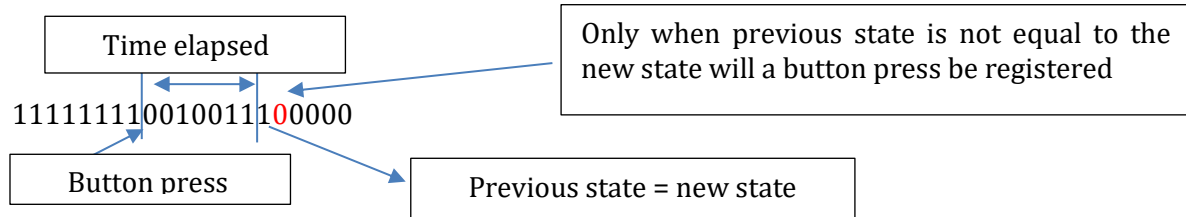


Figure 16: Main Software Flow Diagram

### 4.2 Button Bounce Handling

Button debounce was handled in software. The five physical buttons were set up as GPIO external interrupts in software. When the button is pressed the GPIO will receive a bunch of on and off signals

due to the button bouncing. This bounce lasts for a specific amount of time. The average button bounce was measured to be about 3ms. The button bounce was solved using two different methods. Firstly, by making use of `getTick()` function. When the interrupt is generated, the button records the current system time and waits 25ms before a new interrupt can be generated. The second solution used to ensure that no bounce occurred was comparing the buttons previous state to its current state. Bounce occurs in the following way where 0 represents an off state and 1 represents an on state.



By holding the previous state of the button in a variable right before its pressed and comparing it to final state after the change the code makes sure that a proper change occurs. If the previous state is 0 and then an interrupt is generated the current state is 1, then only do we record the button change. The combination of the two nested if statements prevent two changes to occur when the button was pressed once. Due to being active low the button always resets to its initial state and then goes low when pressed.

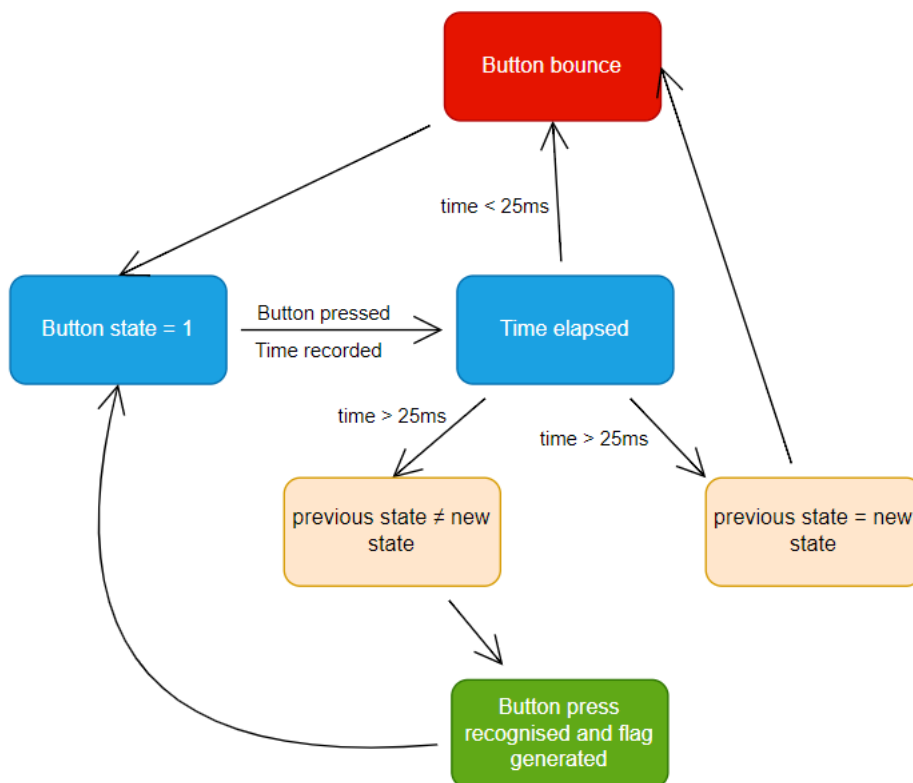


Figure 17: Button Bounce

### 4.3 UART Protocol and Timing

The UART was implemented by populating an array of characters and then checking each character in the array to compare the certain parameters of the received message for the program to decipher what the user is requesting. The UART would also respond by transmitting messages back to the TIC. For example, if the TIC transmitted a request status message. Then the UART would populate an array and respond appropriately by transmitting the status of the system. GPIO pin PA3 was set up as USART\_RX and pin PA2 was set up as USART\_TX, because the messages that were transmitted to the device differed



in length. In a HAL\_UART\_RxCpltCallback function the software would wait until a “\n” character was received as it would then know that the message is fully transmitted. An interrupt is then generated which generates a flag in the main while loop. Once the flag is generated the program enters a function called processInput(). The array of characters is then deciphered. For example, if the third character in the array is “\$” then the system enters measure mode. The basic logic is shown in the flow diagram, under each state there are more checks in the software to deal with specific requirements of the request. For example, if the message wants the system to output a sine wave at a specific frequency, then the software will go into the signal generation system and check for the ‘f’ character and store the frequency value that the UART received.

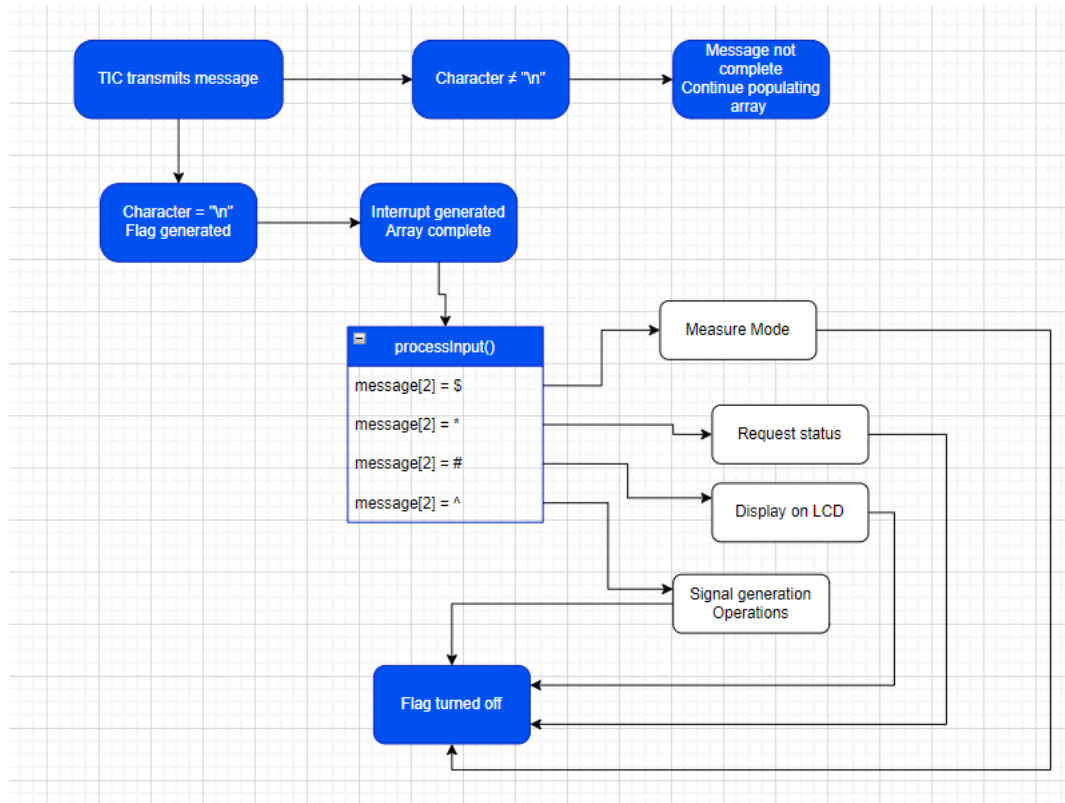


Figure 18: UART Flow

After the message is processed, the flag is set to zero and the system waits for a new message to be transmitted to the TIC.

#### 4.4 ADC Data flow and Processing

The ADC was used to measure sine and DC signals. The ADC was implemented on pin PA1. It is configured as ADC1\_IN2 which means it is started in the main function and assigned a channel. The code shows how the ADC was started and every time a value was needed to be read it had to be polled. This takes the value read by the pin and stores it in a specific address where it can be processed.

```

HAL_ADC_Start(&hadc1);
HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);

```

A timer was set up for the ADC to measure frequencies. Timer 3 was used for the ADC. For DC signals the ADC is sampled one hundred times and stored in an array. The average value of the array is calculated and that is used as the measurement. By sampling multiple times, the system can negate the affect of spikes in the signal and can get a more accurate measurement.

For sine signals the ADC fills a much larger array of 2000 values. For the offset, the average value of all the array is taken. The amplitude is measured by comparing thirty entries of the array. The max and min values that are measured are stored for each thirty values. The average of the max and min values of all two thousand values is taken and by subtracting the max and min values, the software can accurately

find the amplitude. The frequency is measured by comparing the filled ADC array and comparing the values to the measured offset.

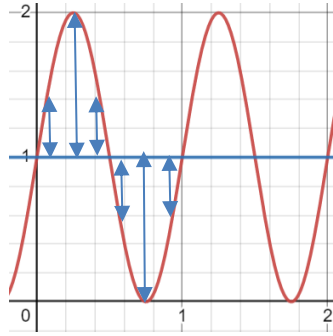


Figure 19: ADC frequency

In figure 19 it is easy to see that before and after the offset three ADC values were stored. As the array is filled on a set timer, TIM3. Then by using the number of samples before the signal goes below the offset for the first part of the wave and above for the second. The software can calculate the period and then convert it to frequency. The average of all the values is taken for each period and that is the final frequency the ADC will output.

#### 4.5 DAC Data flow and Processing

The software for the DAC is used to generate three signals. A DC signal with variable offset, an AC signal with variable frequency, amplitude, and offset. Finally, it can generate a pulse signal with varying duty cycle, offset, amplitude, and frequency. The DAC is set up through GPIO pin PA4 as a DAC1\_OUT1. The DAC was set up in DMA mode so that it could function faster. The DMA request settings is set up in circular mode and accessed memory with a data width of a word. It made use of timer 4 for calculations of values which it needed to output for sine and pulse waves to get the correct frequency. When a signal was needed to be output, the DMA was started and when the system received a request to stop the output the DMA stopped. The software also had flags generated if the signal parameters were changed during an output to change the signal instantly instead of having to manually stop and start the output again. The DAC functioned by populating an array with values of a signal that needed to be output. The standard library math.h was imported to make use of the sin function to calculate values for the AC signal. It also generated an interrupt so that the DAC function would stop the flow of the program when a request is made to output a signal. This did make the code slow if the array that the program had to populate was too big. The array for the sine wave was the largest with a value of 2500 and allowed for smooth operation of the code. The following figure shows the logic of the DAC from when it is turned on and a parameter has been changed. If the parameter is changed before the DAC is turned on, the arrays are filled during normal operation and an interrupt isn't generated.

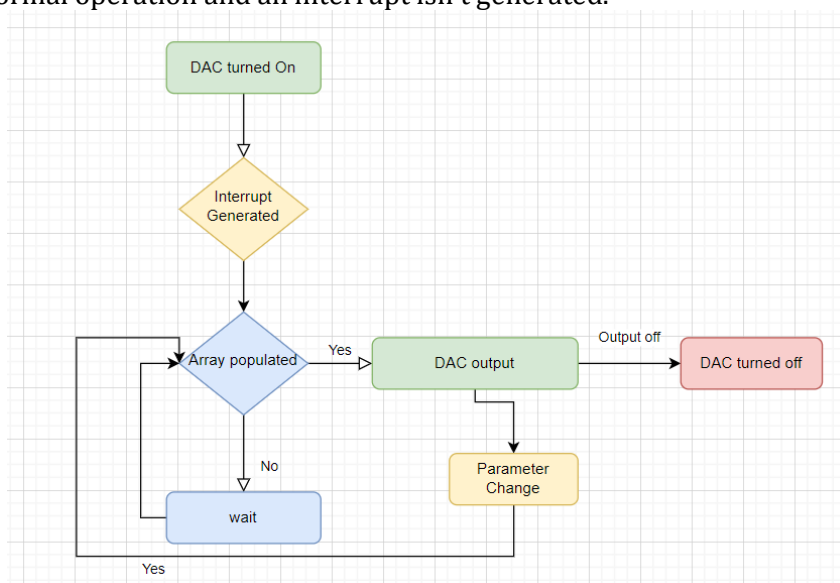


Figure 20: DAC Flow

## 4.6 LCD Interface and Timing

The LCD provided a display interface and a menu where the user could change and view the parameters of the signal generation from the DAC or measurements from the ADC. Therefore the LCD had to be able to recognize state changes. The LCD is connected to 7 GPIO pins which are set up as GPIO outputs. By changing the state of the output, we can write 0's and 1's to the LCD. To make use of the LCD it had to be initialized. This initialization occurred in software. Firstly, the LCD must be set to 4-bit nibble mode operation. An 8-bit command is written to the LCD to change its mode. Timing was very important for initialization.

```
void LCD_init(){
    HAL_Delay(15);
    LCD_DATA(0x00);
    HAL_Delay(31);
    LCD_CMD(0x03);
    HAL_Delay(5);
    LCD_CMD(0x03);
    HAL_Delay(1);
    LCD_CMD(0x03);
    LCD_CMD(0x02);
    LCD_CMD(0x02);
    LCD_CMD(0x08);
    LCD_CMD(0x00);
    LCD_CMD(0x0F);
    LCD_CMD(0x00);
    LCD_CMD(0x06);
}
```

Figure 21: LCD Software Setup

Timing is important for refreshing the LCD however, I did not implement scrolling therefore minimal work with timers was required. I used HAL\_Delay(1) after setting a GPIO pin high or low. HAL\_Delay stops the entire program but because the delay is very small, it did not affect the response of the writing to the LCD. If new information was needed to be displayed a flag was generated which would recognise that the LCD would need to change. When this interrupt is generated in the main while loop a function is called. This function then clears the LCD and display the new information which the LCD should display. This means the LCD isn't continuously updated but only updated when it must display something new. The two main modes are shown in the following state diagram. In display mode the LCD must display the output on the bottom and the measured signal on top. In menu the user can navigate through the menu and change parameters. The LCD starts with a welcome message

## 5 Measurements and Results

### 5.1 Power Supply

Rigorous testing was not done on the power supply regulator circuits. The board powered up correctly. I powered the device using a 9V power supply. After soldering on the 5V regulator circuit I observed LED D1 was turning on. Then I built the 3.3V circuit. I used a multimeter and connected the pins between the 5V and 3.3V jumpers and ground and measured the signal. These measurements were close enough, so I connected the NUCLEO board and powered it up. Figure 3 and 4 shows the regulator circuit. After powering up I just made sure the NUCLEO board turned on correctly in EV5 mode.

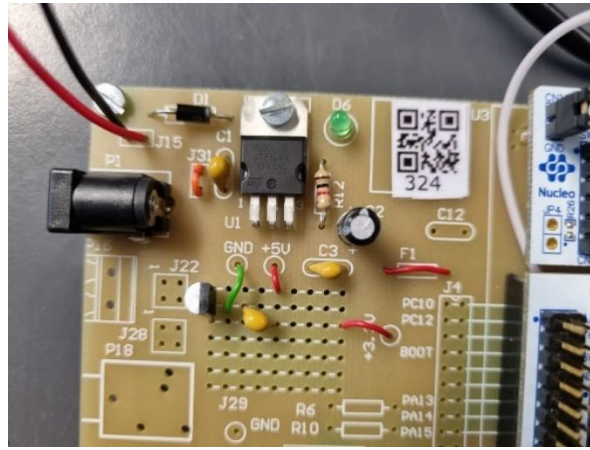


Figure 22: Power Regulator Circuit

## 5.2 UART Communications

Testing for the UART was simple. By making use of termite. A software which functions as an RS232 terminal and connecting the NUCLEO microprocessor to my laptop. I was able to transmit UART messages to the system. I could also view the messages that the device transmitted.



Figure 23: Terminate UART

By transmitting specific commands and requesting the state, I was able to verify that the device was receiving the commands by comparing the expected outcome to the what the device transmitted. I also verified that a signal was being output by connecting a 10x probe to J13, pin 5. I connected the probe to the oscilloscope and set it to trigger mode. Then by resetting the board, it would transmit my student number and the oscilloscope would take a snapshot of the signal being transmitted.

## 5.3 Buttons

Before connecting the buttons, I built the active low circuit shown in figure 5 on a breadboard. Then by connecting the oscilloscope to the output. I could see when the button changed states. I then set up the oscilloscope to take a snapshot when the button changes states. The bounce measurement is shown in figure 24.

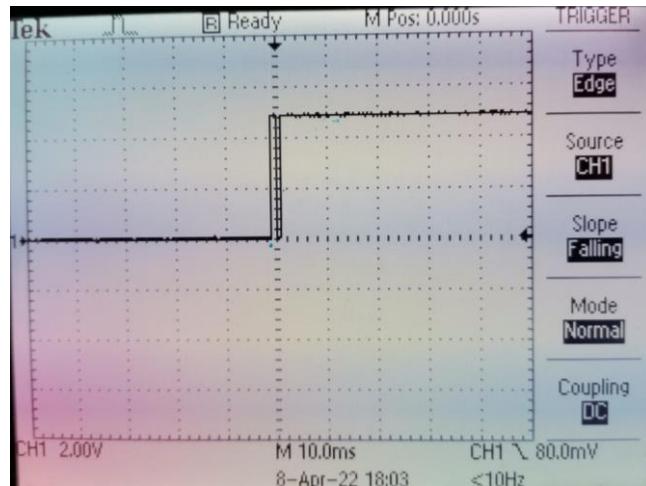


Figure 24: Button Bounce

## 5.4 LEDS

I initially built the circuit shown in figure 25. I used trial and error to find the turn on voltage. By manually increasing the voltage of DC power supply, I eventually found the turn on voltage of the diode to be 1.78V for sufficient brightness. This displayed decent brightness and allowed me to develop the circuit in figure 7.

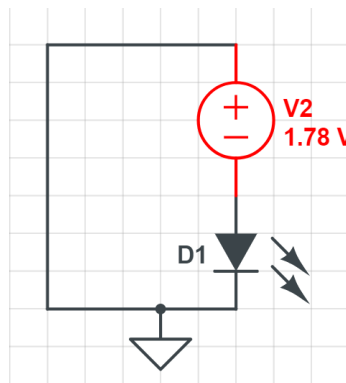


Figure 25: LED Schematic

To test the debug LEDs. I changed the devices state and made sure the debug LEDs turned on and off in the correct states. I send the commands through the UART via termite.

## 5.5 ADC

ADC testing was done using termite, a signal generator, and an oscilloscope. The method used to test was connecting the oscilloscope and signal generator to J13, pin 7 or 8 on the TIC. The reason for connecting an oscilloscope is to verify that the signal generator is outputting the correct signal. This double check was useful as the signal generator was often off by 10%-20%. Then many DC and AC signals were sent to the board. Finally, by using termite I requested measurements of the different signals and verified that the device was measuring the correct signals to 5% tolerance. The offset of DC signals was measured. For AC signals the offset, frequency and amplitude was measured. The ADC had to be calibrated, meaning that for some values I would have to scale up or down depending on their range to get a more accurate reading. For example, the ADC would measure offsets in the range of 800mV to 1200mV 5% higher than it should. I would then scale the measurement down by 5% and get a more accurate reading. I did this for all the signals and parameters. My device however could not measure the frequency of an AC signal below 50 Hz. The full setup is shown below in figure 26.



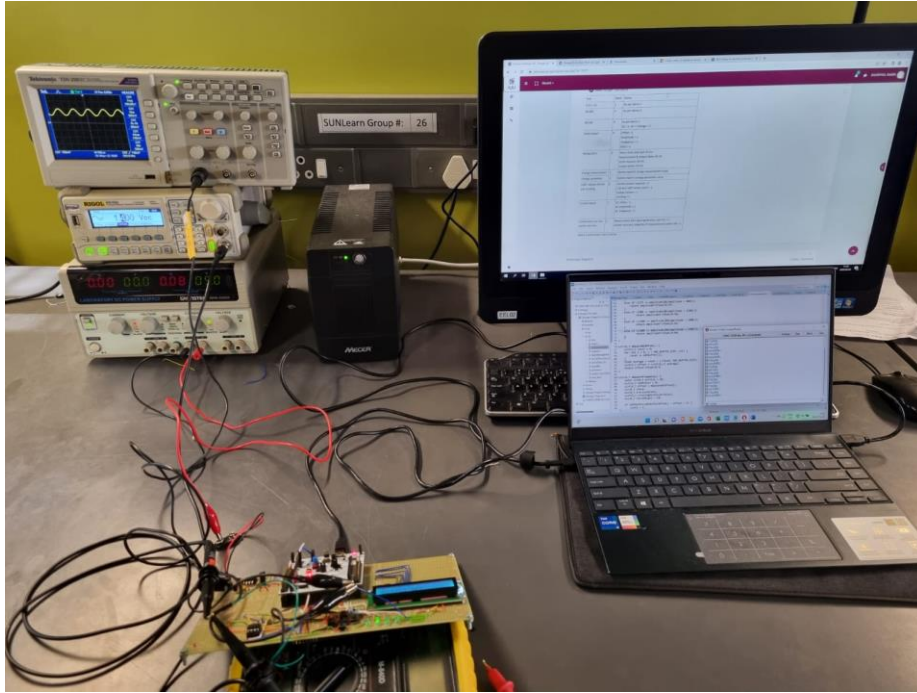


Figure 26: ADC Test Setup

An example of the messages sent in termite to the device can be seen in figure 23.

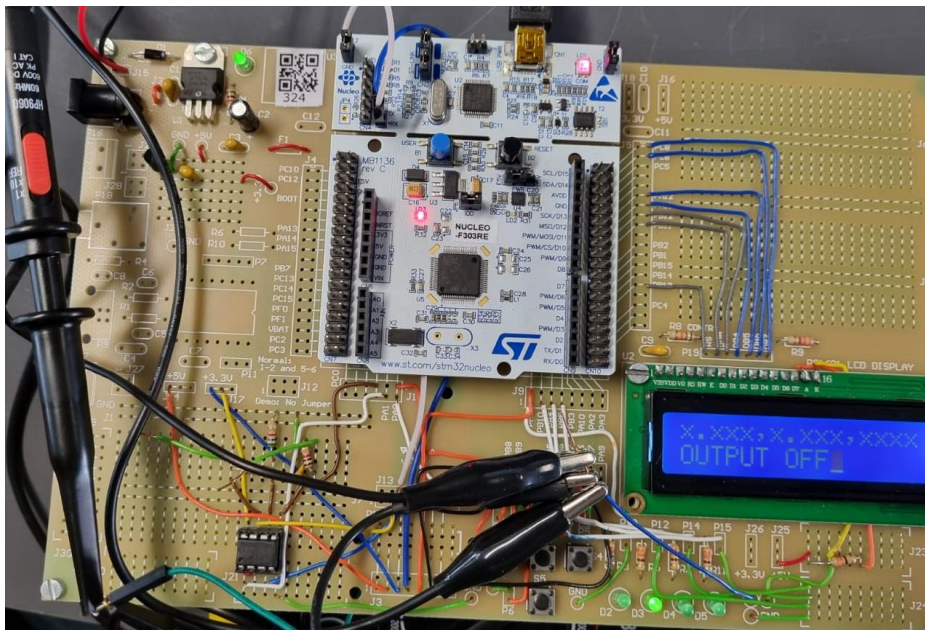


Figure 27: ADC Calibration Connections

## 5.6 DAC

The DAC outputs AC, DC, and pulse signals. I tested the DAC by soldering wires to the input before the Op-Amp filter circuit and measured the signals using an oscilloscope. Afterwards I connected the oscilloscope via a 10x probe directly to the TIC, using connected to J13, pin 3 or 4. Using termite, I changed parameters of the signals and measured the signal which the DAC output. The setup is like the ADC without the signal generator being used. The commands I sent to the device to change on termite can be seen in figure 29. The DAC was also calibrated in the same way as the ADC where I used oscilloscope to compare the actual output to what I was expecting. I scaled the signal up and down depending on whether the value was below or above what it should have been outputting.

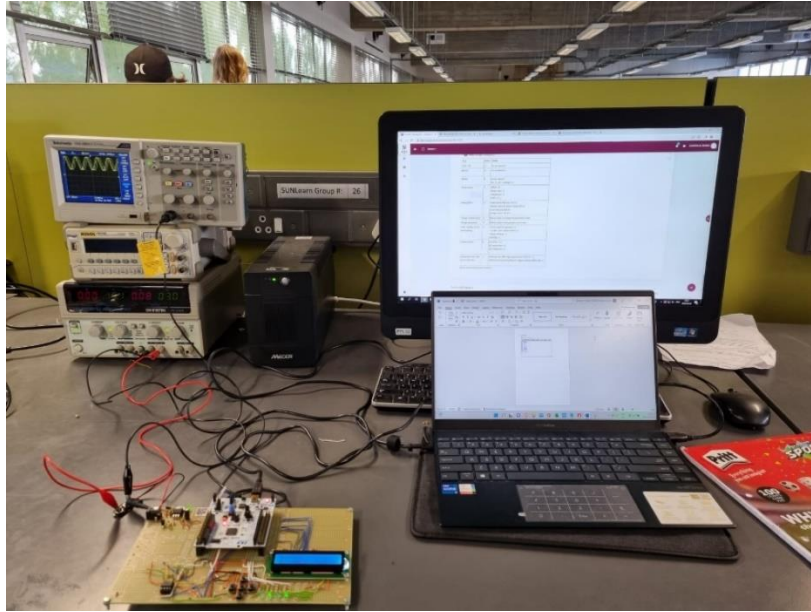


Figure 28: DAC Test Setup

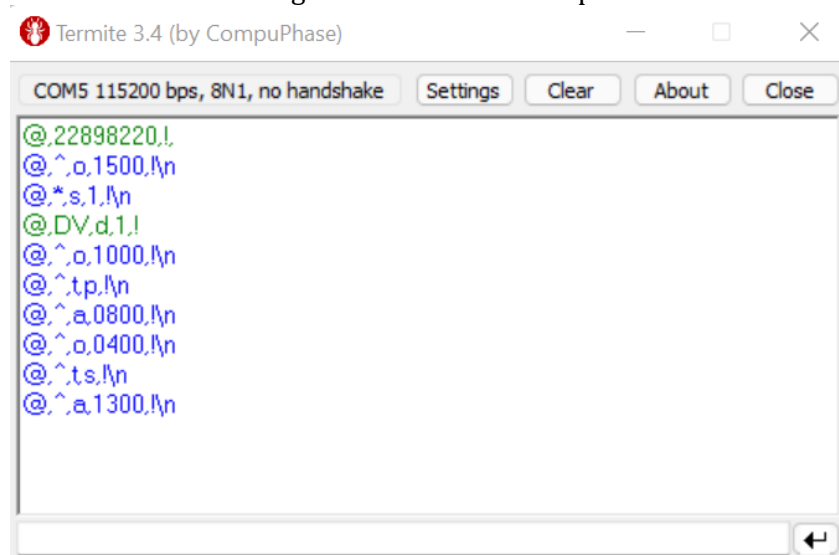


Figure 29: DAC Termitte Commands

## 5.7 LCD

The LCD was tested by soldering the circuit directly to the PCB. I used larger resistor values than what was needed to ensure the safety of the NUCLEO processor. The  $330\Omega$  used for the backlight was chosen as it provided a current of roughly  $0.3\text{mA}$ . The equation is shown below.

$$I = \frac{V - \Delta V}{R} = \frac{5 - 4.1}{330} = 2.73\text{mA}$$

The resistors chosen for R8 and R9 were chosen by trail and error. I connected larger resistance values and slowly decreased the values until the contrast was easy to read. I displayed my student number on the LCD to see how readable the contrast is. This was not the most efficient way to test the LCD as it would be easier to build the circuit on a breadboard, but this method worked due to only having two resistor values and being given the rough values those values should be. Figure 30 shows how the LCDs contrast looked.



Figure 30: LCD

## 5.8 Complete System

The complete system was tested by monitoring termite and connecting the ADC to a signal generator, the DAC to an oscilloscope and the board to my laptop. I ran commands for the ADC and DAC and made sure all the measurements being measured were displayed on the LCD correctly. The signals being generated by the device corresponded to what they were supposed to be. I also navigated through the menu to make sure the system was functioning fast enough while it was outputting and measuring signals. After implementing the menu and doing a test with the ADC, I was clear that the ADC needed to be recalibrated. After recalibration the system responded fast enough, and the ADC was back to being within 5% accuracy.

## 6 Conclusions

The full system accomplished most of the system requirements with slight shortcomings. I did however have to use somebody else's code after the second demo as I initially coded with very little room for adding additional functionality. A shortcoming was that I coded one demo at a time. I should have taken more time to fully understand what the full system requirements were so that I could code in a way where it was easier to add more functionality. I did better with the physical circuits as the PCB was set up in different sections. This could be improved by planning out and drawing flow diagrams before coding. This would have improved efficiency and reduced the time. Besides some poor coding practices, the system responded fast and worked well.

### 6.1 Shortcomings and Non-Compliances

I did not implement the I2C module on the PCB and therefore my system was unable to measure current. The LCD also did not have scrolling as I was unable to implement it in software. This meant that although I could display measurements and the output information on the display but not the units and what the parameters were. My devices DAC could not accurately output sin signals with a frequency less than 50Hz. It could also not change the frequency of a pulse signal.



## 6.2 Appendix

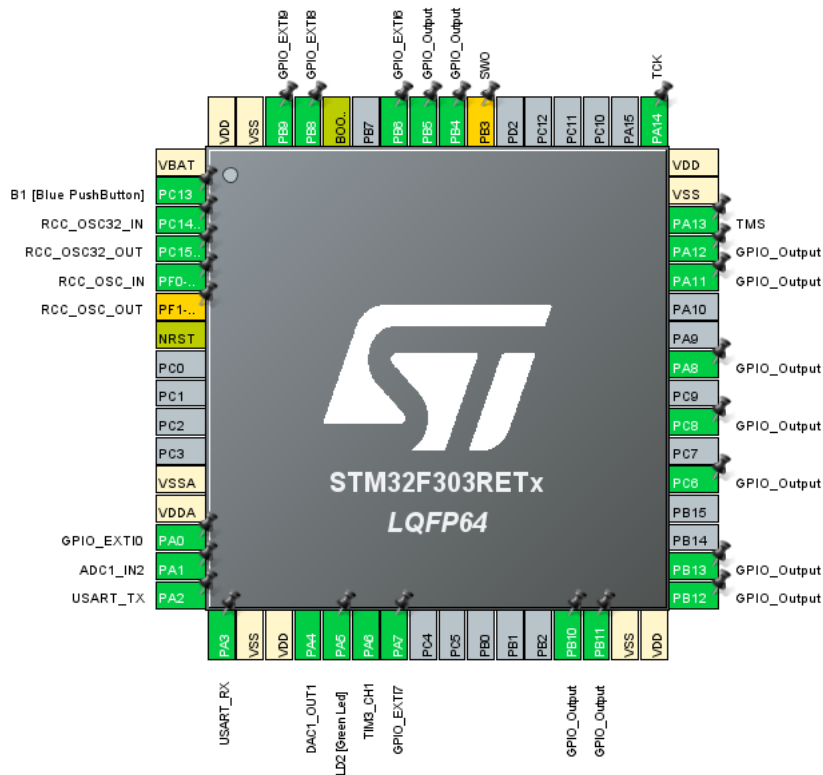


Figure 31: Pin Layout

Table 2: Pin Configurations

GPIO PIN	Configuration
PA0	External Interrupt 0
PA1	ADC1_IN2
PA2	USART_TX
PA3	USART_RX
PA4	DAC1_OUT1
PA6	TIM3_CH1
PA7	External Interrupt 7
PA8	Output
PA11	Output
PA12	Output
PA13	TMS
PA14	TCK
PB4	Output
PB5	Output
PB6	External interrupt 6
PB8	External interrupt 8
PB9	External interrupt 9
PC6	Output
PC8	Output

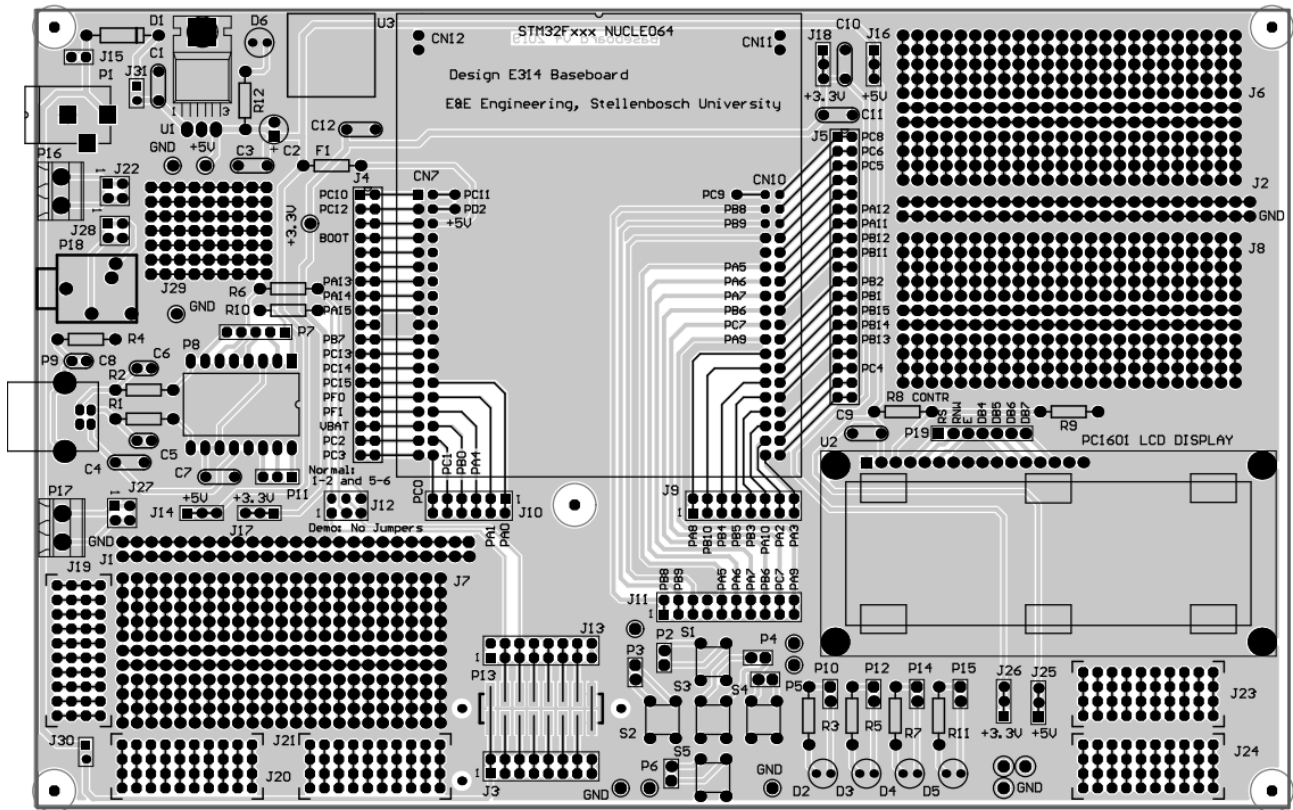


Figure 32: PCB Layout

## References

- [1] G Brown, *Discovering the STM32 Microcontroller*, 2016.
- [2]  $\mu A7800$  SERIES POSITIVE-VOLTAGE REGULATORS, 7805 datasheet, Texas Instruments, Nov. 2003
- [3] FT230X USB to Basic UART IC, FT230X Datasheet, Future Technology Devices International Ltd, 2016.
- [4] STMICROELECTRONICS, STM32F303xD STM32F303xE Datasheet, 2016
- [5] Dr A Barnard, *Lecture7\_LCD.pdf*, 2017
- [6] Brent A. Crosby, *Crystalfontz CFAH1602A-AGB-JP Character Display*, 2005
- [7] Microchip Technology Inc, *2.7V to 5.5V Single Supply CMOS Op Amp*, 2007
- [8] 송윤진, *Lm2950.pdf*, 2013
- [9] Dr A Barnard, L Grootboom, U Louw, and D Klink, *Design (E) 314 Project Definition*, 2022
- [10] Sean Muller, *23575786 demo 2, ADC code*
- [11] Lise Prinsloo, *23726059 demo 2, UART code*