ELECTRICAL & ELECTRONIC ENGINEERING
STELLENBOSCH UNIVERSITY

DESIGN (E) 314
TECHNICAL REPORT

Digital Multimeter and Signal Generator

Author:
Dean Jeggels

Student Number:
23676787

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 woordelikse 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.

DJeggels	23676787
Handtekening / *Signature*	Studentenommer / *Student number*
DE JEGGELS	20/05/2022
Voorletters en van / *Initials and surname*	Datum / *Date*

Abstract

This project tests the designing, building and testing of a multimeter and signal generator using the NUCLEO-F303RE STM32 board.

Contents

List of Abbreviations

TIC - Test-interface Connector
PDD - Project Definition Document
I/O - Input/Output
PCB – Printed circuit board
GND – ground
Op amp – Operational amplifier
GPIO – General purpose input output
LCD – Liquid crystal display
ADC- Analogue-to-digital conversion
DAC- Digital-to-analogue conversion
LED – Light-emitting diode
n_s = number of samples

1 Introduction

This project consists of designing, building and testing of a multimeter and signal generator. A digital multimeter capable of measuring AC/DC voltage and current. A signal generator capable of creating a signal with adjustable waveform, frequency, offset and amplitude. A baseboard and NUCLEO-F303RE has been provided along with necessary components. The NUCLEO-F303RE will be used for inputs and outputs as well as do the computational work. The system will generate a regulated 5V and 3.3V supply voltage from a 9-12V power supply. The device will be able to measure AC/DC voltage in the range of 0.1V to 2.0V as well as AC/DC current from 0mA to 9mA. The device will generate DC/AC/Pulse signals with adjustable offset(0.1V-3.2V), Amplitude(0V-3.1V), Frequency(0Hz-5kHz) and Duty cycle(0%-100%). The device will use UART in receive and transmit modes to report on the system state , parameters of active measurement ,the type of signal generated and output state. The connected LCD display will show the active measurement parameters, active generated signal and its parameters as well as a menu which will allow the user to set different modes and parameters using the push buttons. The device will use for LEDs to indicate the system state.

2 System description

The system will be designed according to user requirements given in the PDD [5].

2.1 Baseboard

The baseboard for this project is provided by the department of E&E Engineering for Design E314 2022. All connection pins in the report are based on the design in Figure 2.1.

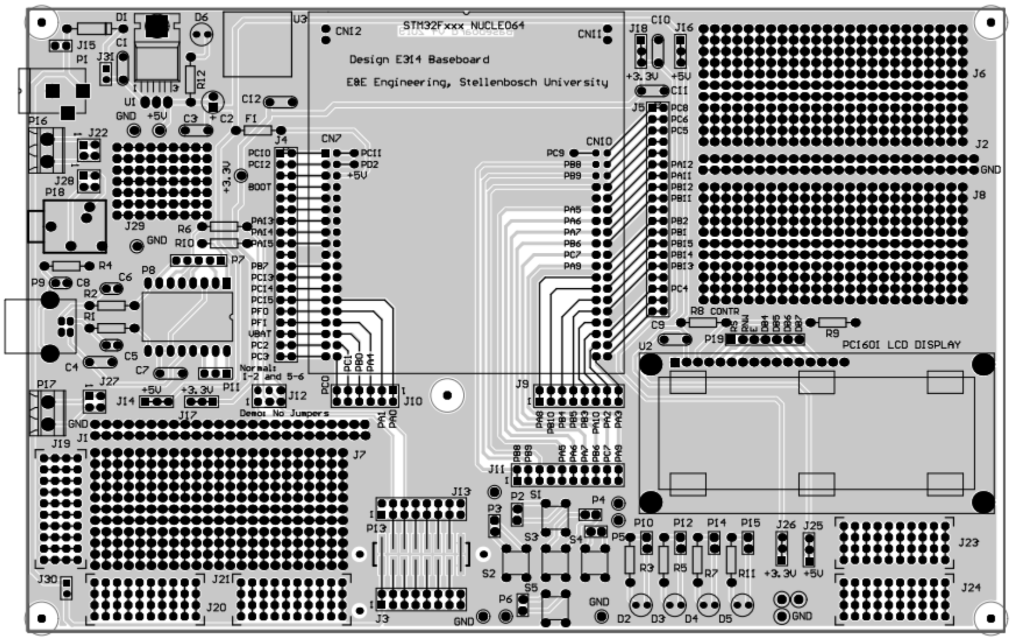


Figure 2.1.1: Baseboard

2.2 Development board

The development board we will be using is the NUCLEO-STM32F303RE[1]. It has a pre-defined layout on the baseboard with the digital and analogue ground pins of the microcontroller hard-wired to the main board ground. Main power from the 5V regulator via the E5V pin (CN7-6) and F1 (fuse or link).

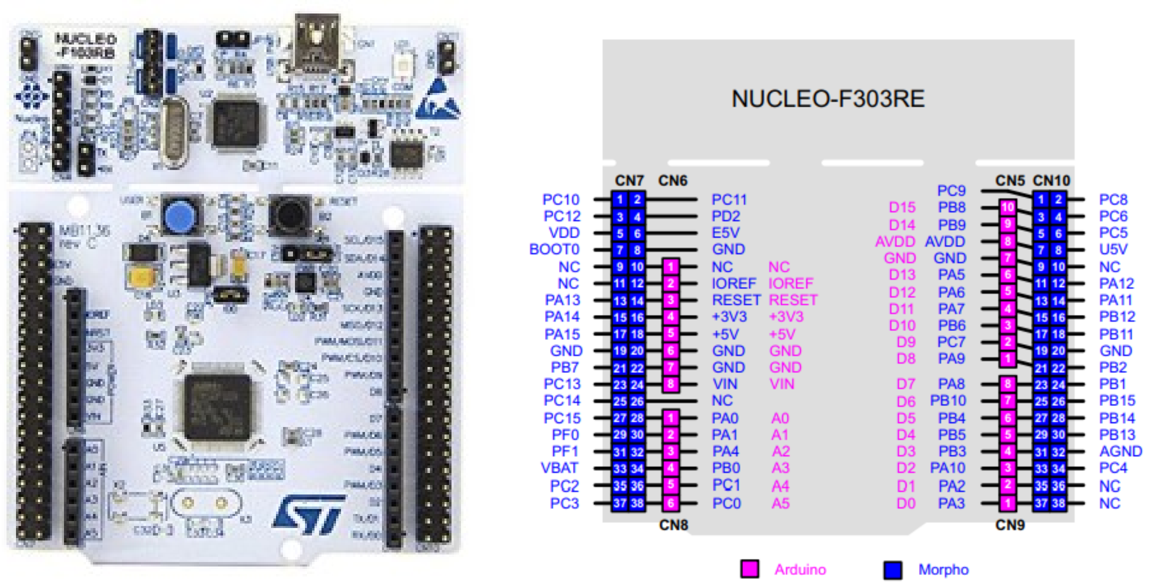


Figure 2.2.1: NUCLEO-F303RE layout

2.3 Power supply

2.3.1 9V power supply

A standard external 9V DC power supply is used as per user requirements, which is then fed into our 5V regulator using the provided barrel jack and socket.

2.3.2 5V regulator

The 5V regulator uses the standard 7805 regulator in a TO220 package to regulate the input from the 9V power supply down to 5V. This supplies the NUCLEO-F303RE[1] with 5V through the fuse F1.

2.3.3 3.3V Regulator

The 3.3V regulator uses LM2950[2] regulator, provided in a TO-92 package to regulate the 5V input to 3.3V as it cannot be fed with the 9V power supply. 9V exceeds the specifications of the LM2950 regulator The 3.3V regulator is connected to jumpers (J17, J18 and J26) to be used across the baseboard.

2.4 Inputs

2.4.1 Buttons

5 Standard push buttons are used for navigation of the menu on the LCD. The menu state depends on which button is pressed. The buttons change the menu state upon release.

2.4.2 ADC

The ADC input is sent from the TIC. The TIC sends a sinusoidal wave which requires no amplification or attenuation. However to protect the microcontroller an op-amp buffer circuit is implemented using the MCP602[3] device provided. The input buffer circuit is a voltage follower with unity gain implemented through one of the op-amps in the dual op-amp device.

2.5 Outputs

2.5.1 LCD

a 16x2 Character LCD display (1602A)[4] is used in this project as a display device. A pre-defined layout has been provided on the baseboard. The LCD will be used to display the different states of our multimeter as well as measurements

2.5.2 Current sensor

The system must measure the current of a signal applied to the input port of the system. From UR2 (Table 1)[5] the system must be able to measure, using the INA219[6], [7], both a DC and an AC current.

2.5.3 LEDs

Standard LEDs are used to display system states.

3 Hardware design and implementation

The hardware design process is explained in detail in the following section.

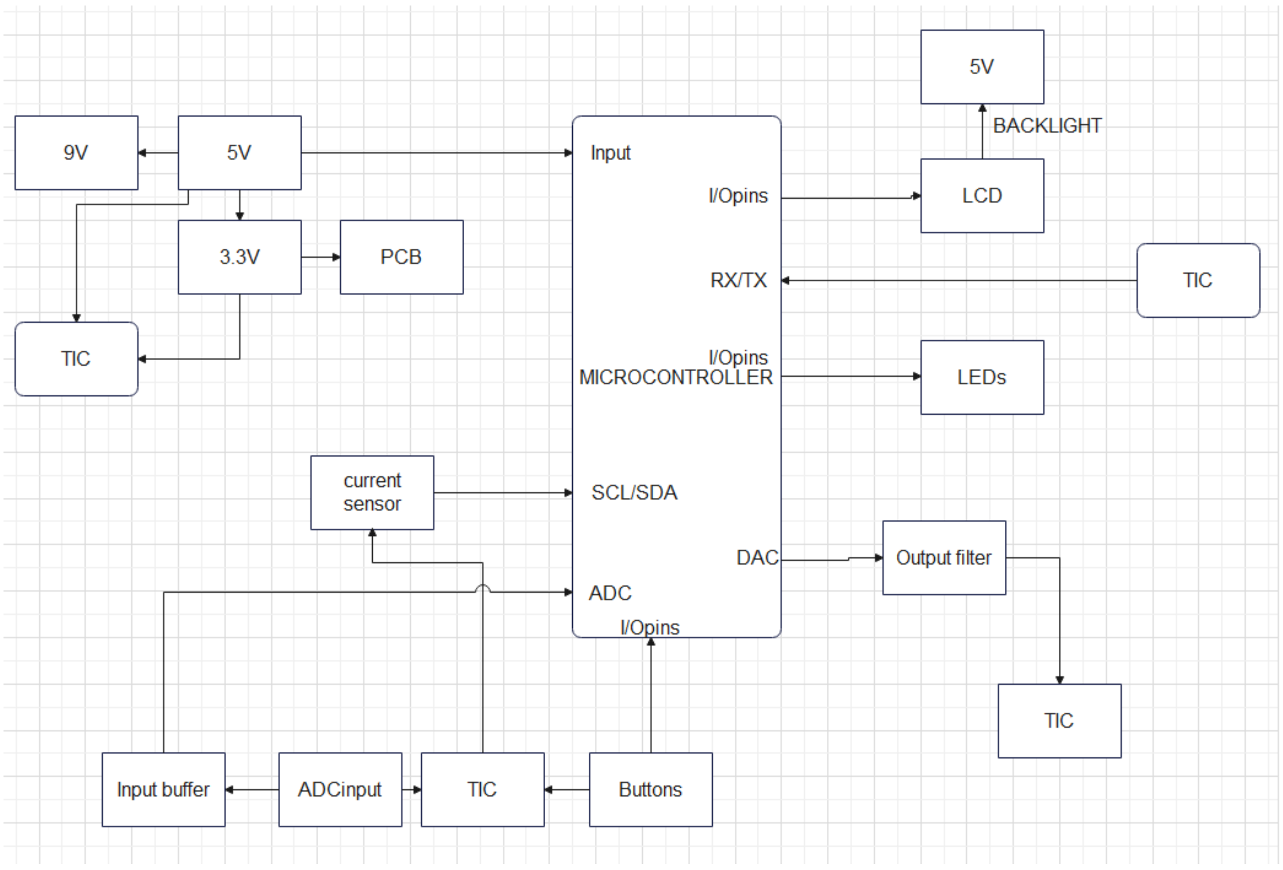


Figure 3.1: Hardware block diagram

3.1 Power supply

3.1.1 5 V regulator

The 5V regulator has a pre-defined layout position on the PCB with connections as shown in figure 2.2 which was given in the PDD[5]. Also given is the LED (D6) connections which will serve as a debug LED.

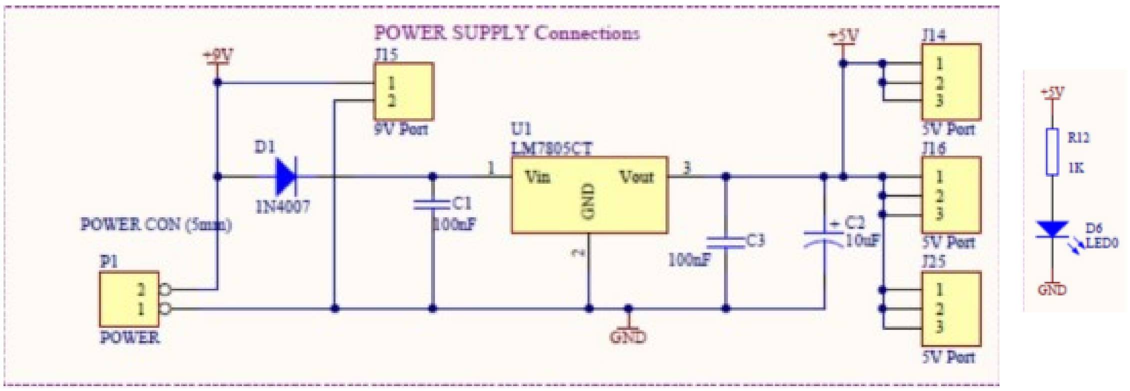


Figure 3.1.1: 5V regulator schematic and pin connections

After these connections have been made the LED (D6) should light up indicating that there is 5V present on the 5V line.

3.1.2 3.3V Regulator

The following schematic for the 3.3V regulator is given in the datasheet for the LM2950[2] as well as the pin layout

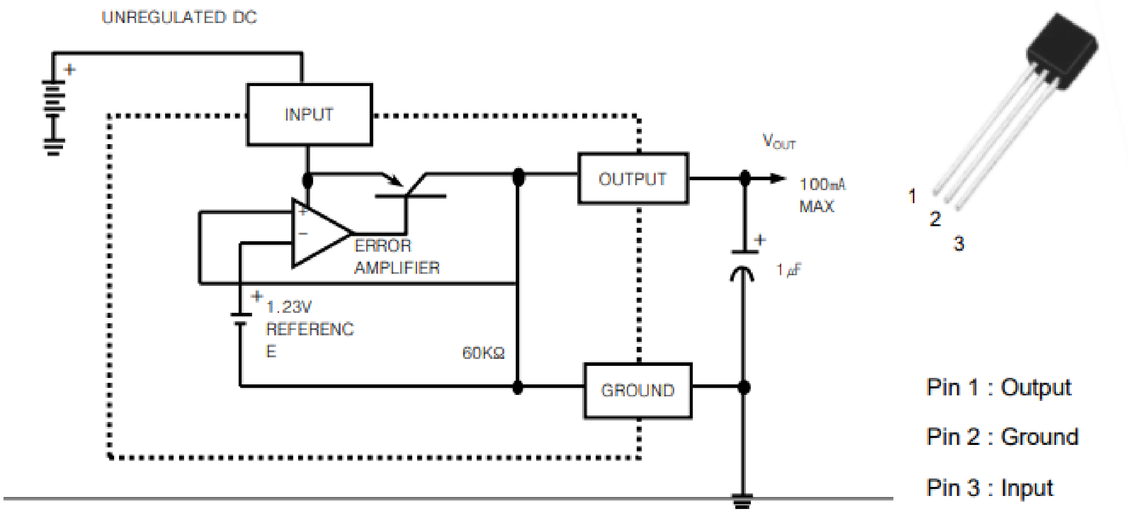


Figure 3.1.2: 3.3V regulator schematic and pin connections

There is no pre-defined layout to build the regulator and we thus use the J29 connections to build our 3.3V regulator and bridge it to the 3.3V line.

3.2 Inputs

3.2.1 Buttons

The buttons all have pre-defined connections on the PCB namely S1-S5 which are logically arranged to present up, down, left, right and middle. The I/O pins for the buttons have been selected according to the following table to allow us to use them as different external interrupts.

Button	I/O pin
UP	PC2
DOWN	PC0
LEFT	PC3
RIGHT	PC1
MIDDLE	PA6

Table 3.2.1: Button to pin connections

We need to however put a current limiting resistor in series with the button otherwise we can damage our regulator circuit. This resistor can be designed in hardware but a software approach has been used to save time and space on the PCB. The push buttons are designed in software to be active low and will be explained in the software design section.

3.2.2 ADC

We make use of the MCP602[3] as an input buffer for our ADC in hardware. We use a voltage follower with unity gain configuration according to the schematic below. GPIO pin PA0 was selected as it was the default pin set by the configuration for ADC1.

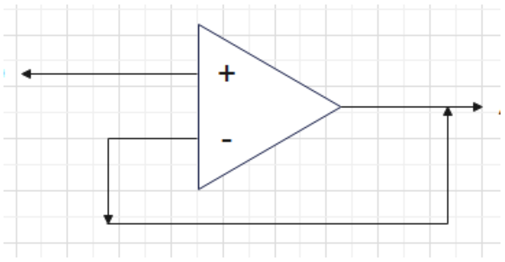


Figure 3.2.1: Voltage follower design

The voltage follower is needed for its high input impedance, so that the signal can draw very little current.

3.3 Outputs

3.3.1 DAC

We make use of the MCP602[3] as an output filter for our DAC in hardware. The following specifications for our signal generation has been set out in the PDD[5].

- DC / AC / Pulse offset: 0.1 V - 3.2 V
- AC / Pulse peak-to-peak amplitude: 0 V - 3.1 V
- AC / Pulse frequency: 0 Hz or 10 Hz - 5 kHz
- Pulse Duty cycle: 0%-100%
- Signal shape: DC, sinusoidal(AC), or pulse

To make sure we stay within these specifications we need to design an active low-pass filter with a DC gain of 2 according to the following schematic. This will allow all the high frequencies to be attenuated and only pass frequencies from 0 Hz to the cut-off.

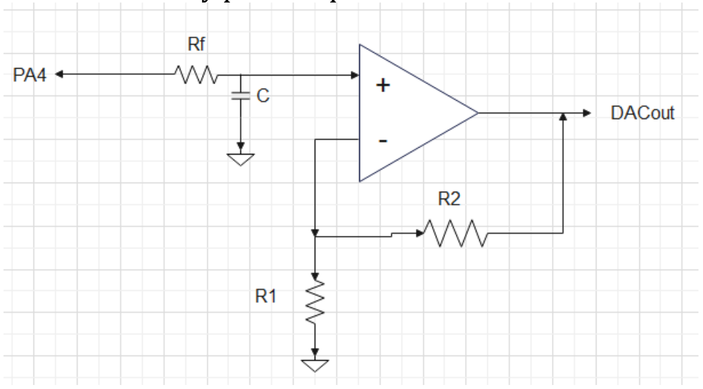


Figure 3.2.2: Low-pass filter design

First start with R1 and R2.

To achieve a DC gain of 2 we design R2 and R1 as follows:

$$R1 := 10\ \mathrm{k\Omega}$$
$$R2 := 10\ \mathrm{k\Omega}$$
$$DCgain := 1 + \frac{R2}{R1} = 2$$

Our cut-off frequency should be 5kHz according to specification but we design for 7.5kHz to leave space for error margins. We choose Rf

$$Rf := 10\ \mathrm{k\Omega}$$
$$fc := 7,5\ \mathrm{kHz}$$
$$C := \frac{1}{2 \cdot \pi \cdot Rf \cdot fc} = 2,1221 \cdot 10^{-9}\ \mathrm{F}$$

According to the DC characteristics of our op amps given on page 2 of the datasheet[3].

$V_{OUT} \approx V_{DD}/2$,

The DC gain of should cancel out the 2 in this equation and give us a default DC output of VDD which is 3.3V.

We find our voltage swing of the output in the datasheet on page 2[3].

Linear Output Voltage Swing	V_{OUT}	$V_{SS} + 100$	—	$V_{DD} - 100$	mV
	V_{OUT}	$V_{SS} + 100$	—	$V_{DD} - 100$	mV

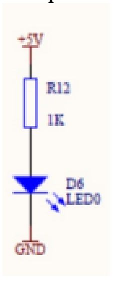
This means our voltage will clip at 0.1V and 3.2V.

3.3.2 Current sensor

Was not implemented although a tutorial was given online. [8]

3.3.3 LEDs

To design LED D2-D5, the voltage drop and operating current of the LED is required. These specifications can be required by using LED D6 (Figure 2.1.1) as a reference by measuring the voltage drop over the LED and the voltage drop over the resistor R12 as shown in Figure 2.1.2.



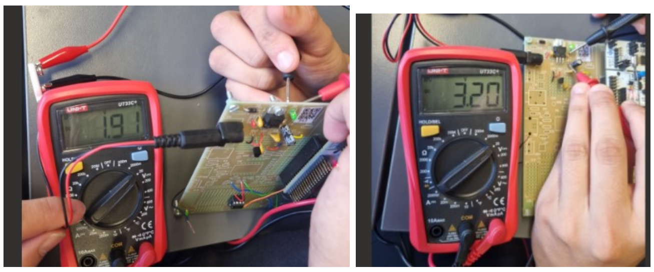


Figure 3.3.1: Voltage measurements for LED

The voltage drop of the LED is the 1.91V. To get the current through the diode we apply Ohms law.

$$I = \frac{v_{R_{12}}}{R_{12}}$$

$$I = \frac{3.20\ V}{1000\ \Omega}$$

$$I = 3.20\text{mA}$$

We can now design LED D2 according to these specifications of LED D6 using Ohms law to obtain the required series resistor to limit the current to an acceptable level. The GPIO pins have a specified output current of 80mA found from the datasheet. The LED D2 is chosen to be driven by GPIO pin PB6 which has a voltage output of 3.3V according to datasheet. Applying Ohms law yields.

$$R = \frac{v_0 - v_D}{I_D}$$

$$R = 434\Omega$$

Using standard resistor of 470Ω in the lab stock, it gives us a visible brightness for the LED as well as the appropriate current required to not damage the output pins. Giving us a final design for the LEDs as shown in Figure 2.1.3

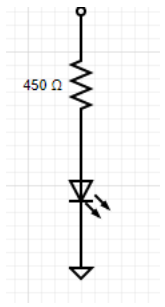


Figure 3.3.2: LED D2-D5 design

LEDs	I/O pins
D2	PB6
D3	PC7
D4	PA9
D5	PA10

Table 3.3.1: LED I/O pins

3.4 LCD

The Lcd has a pre-defined layout on the PCB. It is recommended that a male header strip is used to solder the LCD onto the PCB. We will use the LCD in 4-bit interface mode.

3.4.1 Pins

The following I/O pins were chosen for the LCD.

LCD pins	I/O pins
RS	PC4
RW	PB1
E	PB2
D4	PB11
D5	PB12
D6	PB13
D7	PB14
A	+5V
K	GND

Table 3.4.1: LCD I/O pins

Pins D0-D3 are hard wired to GND on the PCB

3.4.2 Power

The required VCC(5V) and GND pins of the LCD are already pre-connected on the PCB. We do however need to connect power to the A and K pins as we need this to power the backlight. We will use the 5V line and connect it through a resistor on J8 through to the A pin and ground the K pin.

3.4.3 Backlight

The backlight LED can be switched on by wiring pins A to 5V and K to GND. We cant connect these pins directly to 5V and GND. We need a current limiting series resistor between 5V and pin A. This limits the current to our backlight and prevents the LCD getting damaged from too high of a current. We need to keep the current below 10mA

We first measure the voltage drop across pins A and K and then use ohms law to get our minimum resistor value.

$$Vdrop := 4\ \mathrm{V}$$

$$\frac{V\_5V - Vdrop}{I} = 100\ \Omega$$

Based off of this we choose the Resistor as 330 Ω to make room for error margins. This should regulate the current to lower than 10mA

3.4.4 Contrast

The following diagram for our LCD was given.

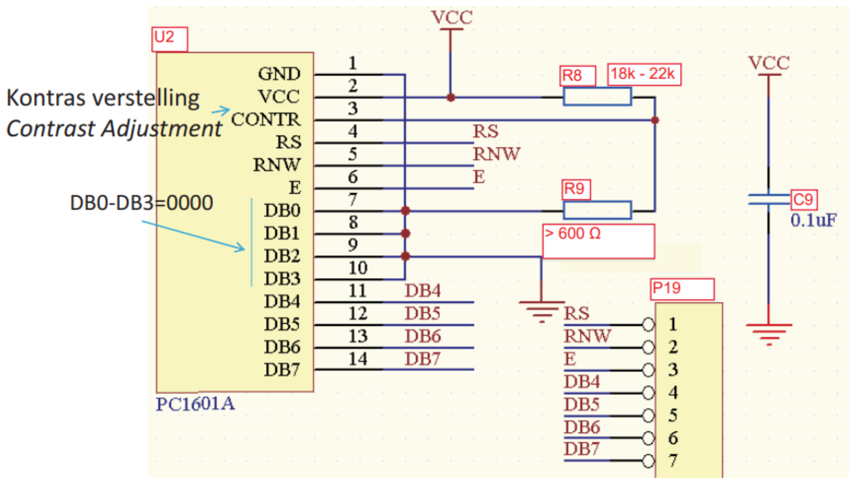


Figure 3.4.4.1: LCD connections

For the contrast we need to adjust resistors R8 and R9. It is specified to keep the values between 18kΩ and 22kΩ for R8 and bigger than 600 Ω for R9. R8 is chosen as 18k and R9 is changed with a variable resistor based on this choice until the contrast is good. R9 = 3.3kΩ gives us a good contrast.

4 Software design and implementation

Discuss top-level software design and implementation, using design tools, like flow diagrams and timing diagram, where needed.

4.1 Timers

First we start by the setup of our timers. We will use 3 timers namely TIM1,TIM2, and TIM3.

TIM1

Timer 1 will be used as a general timer for our delay functions. We use the internal clock as our clock source which runs at 72MHz. We pre-scale the frequency by 72 which will scale it down to 1MHz for our timer. We set the ARR value to the maximum 16 bit value which is 0xFFFF. And enable auto reload preload.

We have now created a timer which counts in a frequency of 1MHz or 1µs until 65 535µs and then restarts. This will be used in our delay functions.

TIM2

Timer 2 will be used to create a sample rate for our ADC. We will use it as an interrupt which will trigger an interrupt at a rate of 20kHz. The clock source is the internal clock, we pre-scale it by 72. And use an ARR value of 50.

From the user manual of stm32 we get the following equations to calculate the rate.

$$T := \frac{(ARR)}{ftimer} = 5 \cdot 10^{-5}\ s$$

$$f := \frac{1}{T} = 20000\ Hz$$

……………………………………………Equation 4.1.1

TIM3

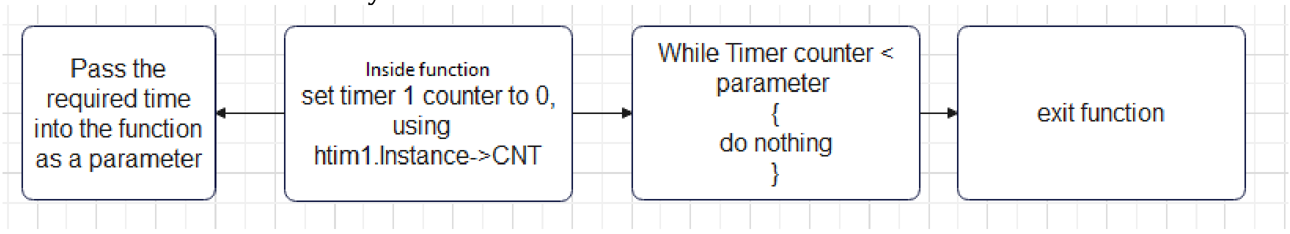
Timer 3 will be used to create a sample rate for our DAC. We once again pre-scale it by 72 and use an ARR value of 100 which will give us 10kHz according to equation 4.1.1. When the timer reaches the ARR value we trigger an update event which will trigger our DAC.

4.2 Delay functions

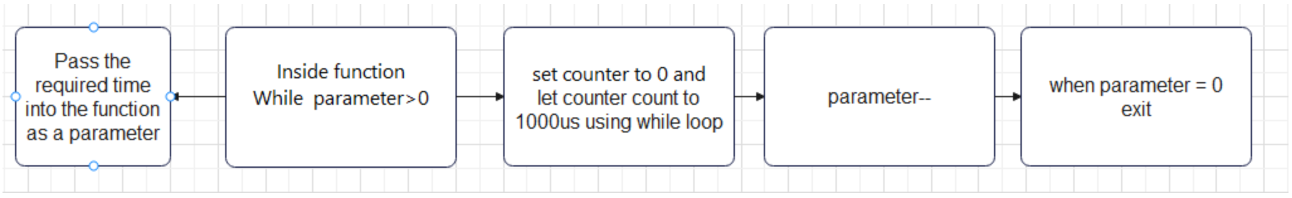
Throughout our project we need to use delays for certain things. It is best to create our own delay functions as it is suggested to not use HAL_Delay() since it has low priority and wont increment inside an interrupt.

It is best to understand these functions through a flow diagram.

To create a microsecond delay:


Pass the required time into the function as a parameter
Inside function
set timer 1 counter to 0, using htim1.Instance->CNT
While Timer counter < parameter
{
do nothing
}
exit function

We already setup the timer to count in µs but to create a ms delay we create a separate function
To create a millisecond delay:


Pass the required time into the function as a parameter
Inside function While parameter>0
set counter to 0 and let counter count to 1000us using while loop
parameter--
when parameter = 0 exit

This essentially has the same effect as multiplying µs by a 1000.

4.3 Buttons

4.3.1 Pins

All pins were configured as external interrupts. This pin setup allows us to use different interrupt handler functions for each button.
It is important to use an internal pull-up resistor for the buttons as this limits the current. A Pull up resistor is used to create an active low setup for our buttons as specified by the PDD[5].

4.3.2 Debounce

Bounce is an inherent property of mechanical buttons that introduce electrical noise when they are pushed. This can cause an input signal to be read incorrectly and not give us our desired response. It is best to fix this hardware issue in software using the method of debouncing.

To find the estimated time we need program our debounce, we simply set up a circuit with the button and connect the oscilloscope probes to either end of the button which should us a visual view of the button bounce noise as in Figure 3.1.1.

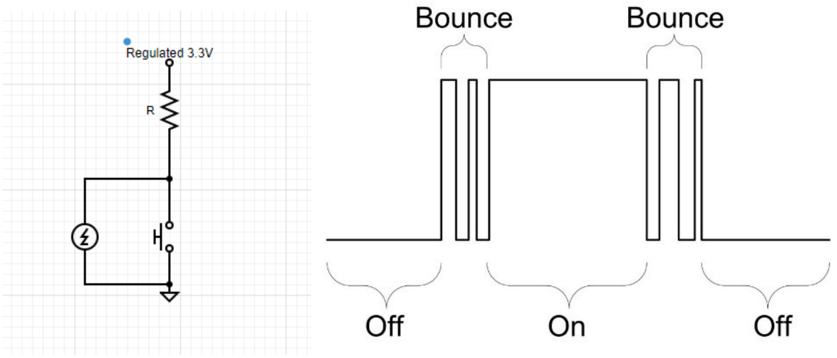


Figure 4.1.1: Visual button bounce and circuit diagram.

We need to 'pause' the microcontroller while we wait for the bouncing to finish. A simple skeleton flow diagram is shown in figure 4.1.2.

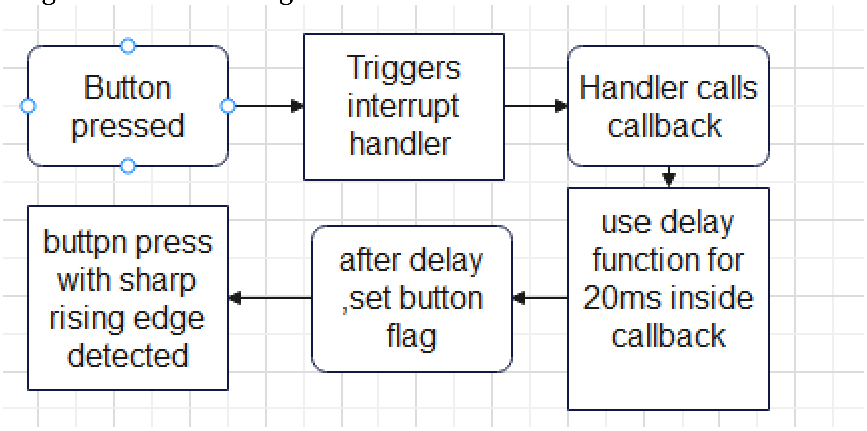

Button pressed
Triggers interrupt handler
Handler calls callback
use delay function for 20ms inside callback
after delay ,set button flag
buttpn press with sharp rising edge detected

Figure 4.1.2: Button debounce flow

4.4 UART

The UART will operate at a baudrate of 115200bps, with 8 data bits, 1 stop bit and no parity bit using TTL signal levels (8N1). This will give us a period of 86.5 µs per byte.
The UART protocol and message formats were given in the PDD[5]

We need to receive and send, configuration and command messages from and to the TIC. The USART 2 global interrupt was enabled which allows us to trigger interrupts using UART. The interrupt is triggered whenever a UART message is detected in the serial line. We store each character it receives in a command buffer. We keep storing the characters until a new line character or \n is detected. This allows us to receive commands of variable sizes. We then need to evaluate our received command and verify that it is a valid command and then proceed with what the command wants us to do.

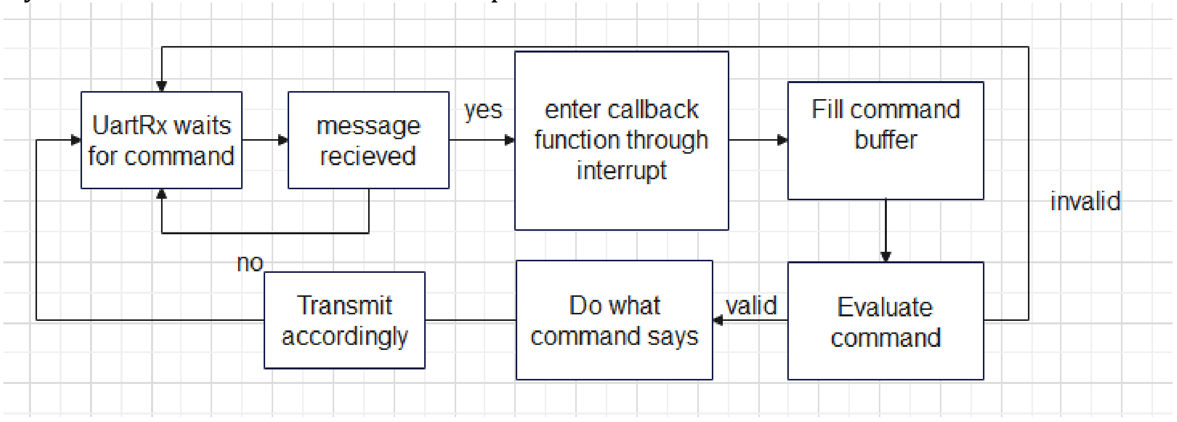


Figure 4.4.2: UART flow diagram

4.5 ADC

Timer 2 is used as described above. On every interrupt of our timer we set a flag that will allow the program to record a sample. We use this flag in a Measure() function which is constantly running in our main while loop. It will only enter the function calculations if the ADC flag is set to 1.

4.5.1 Conversion

We will use ADC conversion by polling. When the flag is on we start the ADC peripheral and start conversions wait for the ADC conversion completion, retrieve the conversion results, store them in an array and stop the conversion and ADC peripheral. Doing this in a for loop will allow us to create sample points. We will use a 1000 sample points in our calculations as this will give us enough time before the interrupt is triggered again.

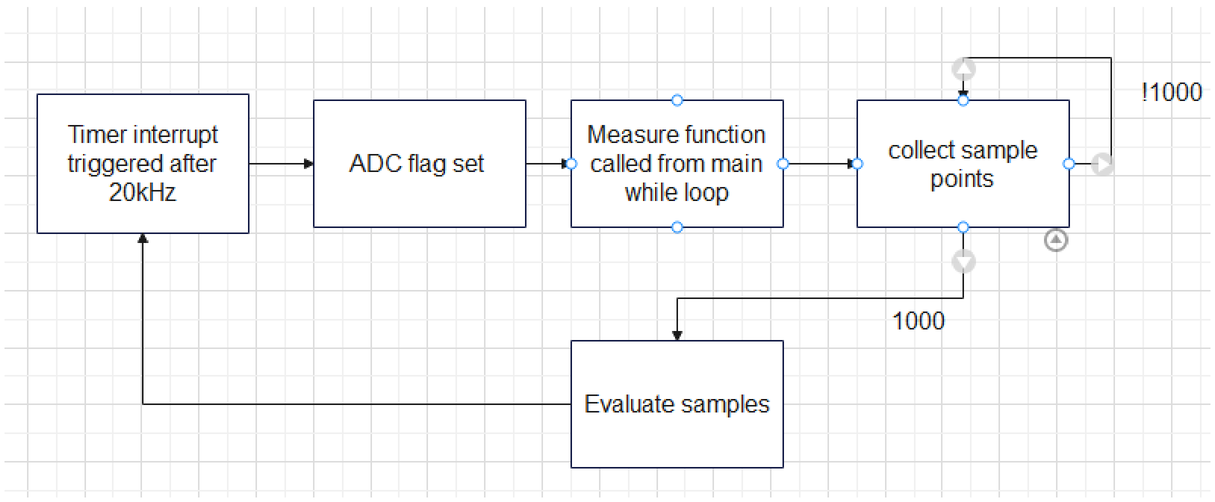

Timer interrupt triggered after 20kHz
ADC flag set
Measure function called from main while loop
collect sample points
!1000
1000
Evaluate samples

Figure 4.5.1: ADC flow diagram

4.5.2 Calculations

We need to take our array of sample points and evaluate it to give us measurement details about our signal. It is important to convert our analogue signal to digital by using a conversion factor. Our Analogue values range from 0-4096, the maximum 12 bit number from our register. We multiply our received analogue value by the maximum voltage we want to measure which is 3300mV and divide by our max analogue value.

$$ConversionFactor = \frac{MAXvoltage}{MAXanalogue}$$

The signal definitions will be used as specified in the PDD[5].

We can start by calculating the offset using the average of our samples.

$$Offset = \frac{Total\ of\ samples}{amount\ of\ samples} ConversionFactor$$

The amplitude is specified as the peak to peak value of our signal. We can calculate the amplitude by comparing the sample points and getting the max and min value. We use a basic max and min function which compares the previous array entry to the next one and stores the higher or lower value into a variable. Subtracting the max and min values will give us the peak to peak amplitude.

$$Amplitude = (Max - Min)ConversionFactor$$

Frequency

The frequency can be calculated by detecting the time it takes for the signal to cross the offset 3 times. A counter will be triggered when the signal first crosses the offset. Once the 3rd crossover is detected we multiply the current counter value by the period of the timer which in our case is 50us. The inverse of the period will give us the frequency of our signal.

$$Period = (counter * periodOfTimer)$$
$$frequency = \frac{1}{period}$$

4.6 DAC

A tutorial has closely been followed by Controllerstech[9]. We need to generate a signal according to parameters set by either UART or menu. This signal can either be DC, sinusoidal or pulse. Table 1 , UR3 of the PDD[5] gives us the user requirements for our signal. We will use DAC in DMA mode IO operation.

4.6.1 Configuration

Selecting DAC1 enables pin PA4. We need to also enable DMA and set it up for sine wave generation. Set the DAC trigger as Timer 3 trigger out event. Timer 3 has already been setup as mentioned above.

4.6.2 Calculations

We convert our digital parameters into analogue signals using a conversion factor to be converted by our DAC.

$$ConversionFactor = \frac{MAXanalogue}{MAXvoltage}/2$$

<u>DC:</u>

We start by initializing an array of 5 entries which we will populate by our received parameter. For DC we will receive an offset parameter change. The following equation was used.

$$DC[i] = offset * conversionfactor/2$$

<u>AC:</u>

We are going to use an array of 100 entries for our sample points.
The following equation was found in a DAC document provided by ST

$$y_{\mathrm{SineDigital}}(x) = \left(\sin\left(x \cdot \frac{2\pi}{n_s}\right) + 1\right)(\mathrm{ConversionFactor})$$

This formula assumes an offset of 0 and an amplitude of 1. We can edit this formula to account for variable offset and amplitude value.

$$y_{\mathrm{SineDigital}}(x) = \left(\mathrm{AMPLITUDE}/2 * \left(\sin\left(x \cdot \frac{2\pi}{n_s}\right) + 1\right) + \mathrm{OFFSET}\right)(\mathrm{ConversionFactor})$$

Where n_s=100 and x is our array index. We can then get a 100 sample points from a for loop. This should give us a clean sine wave. With the correct offset and amplitude.

To get the correct frequency output we need a separate calculation. Our frequency depends on our ARR value of our timer. We are working with a fixed amount of samples with a variable frequency. It is thus best to use the ARR value to change the frequency. We can change the ARR value depending on our frequency according to the following equation.

$$TIM3 \rightarrow ARR = \frac{finternalclock}{ns * frequency} - 1$$

It is important to change our timer counter back to 0 when we want to change our frequency as this could causes issues when the ARR value is set lower than our current timer count value. The timer counter will continue counting until it overflows and will cause discrepancies.

Pulse:

We need create a pulse signal which is high and low depending on our duty cycle. The minimum/low value of our pulse signal will be determined by the offset and the high/ max value will be determined by our amplitude.

We create an array of 100 samples which will cover a full duty cycle period of our pulse signal.

We fill our first entries with our high/amplitude values in a loop up until the required duty cycle.

We then fill the remaining entries up until 100 by our min value which is simply the offset value we receive.

0-Dutycycle:

$$PULSE[i] = (Amplitude * 2 + offset)(conversionfactor)$$

4.6.3 Block diagram

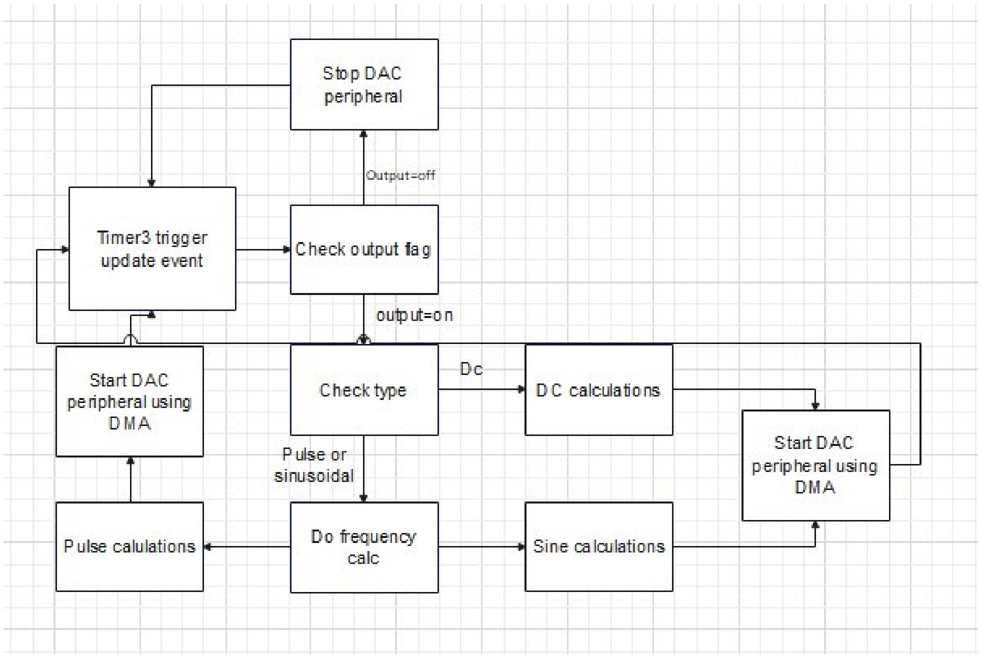


Figure 4.6.3: DAC flow diagram

4.7 LCD

4.7.1 LCD interface

We need to write instructions to our LCD according to the (Orise Tech SPLC780D1 Controller Datasheet) page 8[4]. To help us write data to our LCD a tutorial and LCD library was used from deepbluembedded[10]. This library contains all the necessary functions for our LCD interface.

Functions:

LCD_DATA(`unsigned char Data`)

This function sends 4 bit data to the LCD. Because we are using our LCD in 4-bit nibble mode we are only allowed to write to pins DB4-DB7 as pins DB0-DB3 are hardwired to ground. This function takes in the data we want to send and writes either high or low to the pins based off of the data received.

`LCD_CMD(unsigned char a_CMD)`

This function sends commands to the LCD. It is necessary to send commands for the initialization of the LCD. The function writes 0 to the RS pin to indicate that a command is being sent. It then sends the command using the data function above. Toggles the EN pin on and off which acts as a clocks signal to indicate the data is ready.

`LCD_Init()`

The command function will be used to initialize our LCD for a 4-bit interface according to the initialization process on page 12 of the datasheet[4]. The delay functions described above on page 14 has been used to create delays in between our commands. The initialization function will be called in our main before the while loop to set up our LCD for use.

`LCD_Write_Char(char Data)`

The write character function will be used to write and display characters to our LCD display according to page 14 of the datasheet[4]. It first separates the high and low nibbles of data by using the & operand. &ing with 0x0F will give us the low nibble and 0xF0 will give us the high nibble. It then writes 1 to the RS pin to indicate that the LCD needs to write or read data. It then uses the data function to write the high nibble to the pins , sends the EN clock signal then writes the low nibble to the pins and sends the EN clock signal again.

MENU:

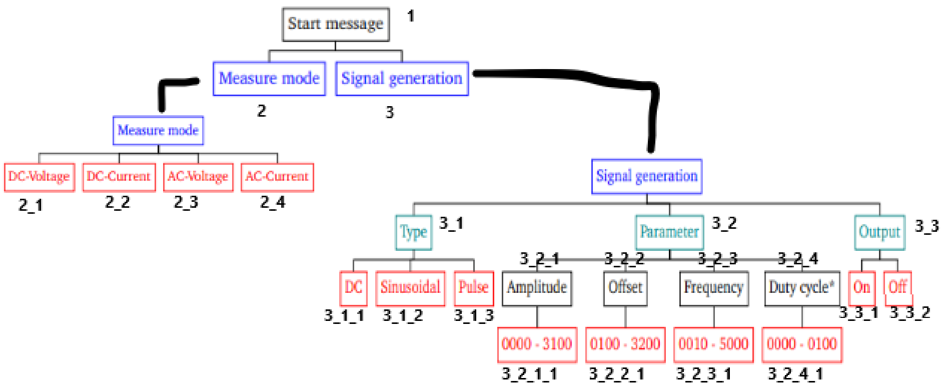


Figure 4.7.1: Menu state diagram

A menu will be displayed on the LCD. The above state machine has been given in the PDD. The menu systems works with the push buttons and should traverse through the tree using the up and down buttons. The far left state should be shown first when traversing up or down and then the left and right buttons should be able to traverse left and right. The red colored states should be set by pressing the middle button and the system should change accordingly.

The menu was implemented with nested if else statements , with the current state and button pressed variables as parameters. The next state was set according to the current state and button press.

Scrolling could not be implemented thus the “no-scrolling” display option was used as specified in the PDD.

5 Measurements and Results

5.1 Power supply

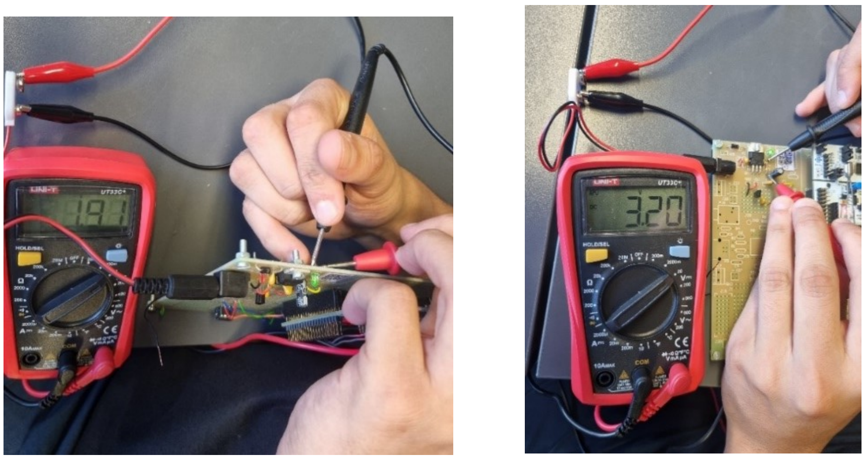
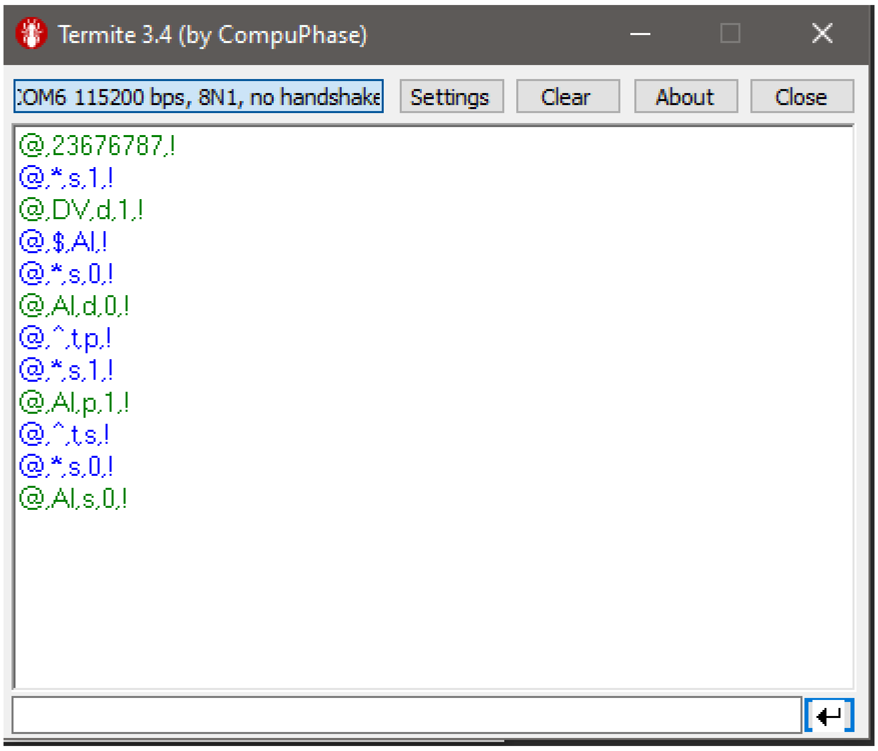


Figure 5.1.1:Voltage across LED D2 and resistor

Here we confirm our voltage across the LED. Adding up the voltages from our LED and resistor gives us 5.1V which means our 5V regulator successfully regulates the 9V voltage down to 5V. The LED is visibly bright which indicates that there is voltage flowing through the 5V line.

5.2 UART



:OM6 115200 bps, 8N1, no handshake

```
@,23676787,!
@,*,s,1,!
@,DV,d,1,!
@,$,Al,!
@,*,s,0,!
@,Al,d,0,!
@,^,t,p,!
@,*,s,1,!
@,Al,p,1,!
@,^,t,s,!
@,*,s,0,!
@,Al,s,0,!
```

Figure 5.2.1: UART verifications

Termite[11] was used as a terminal program to send serial communications on to our UART. Commands were sent according to the format in Figure 4.4.2. The system responds with the valid response with each command as seen above.

5.3 Buttons

An oscilloscope was hooked up to the button.

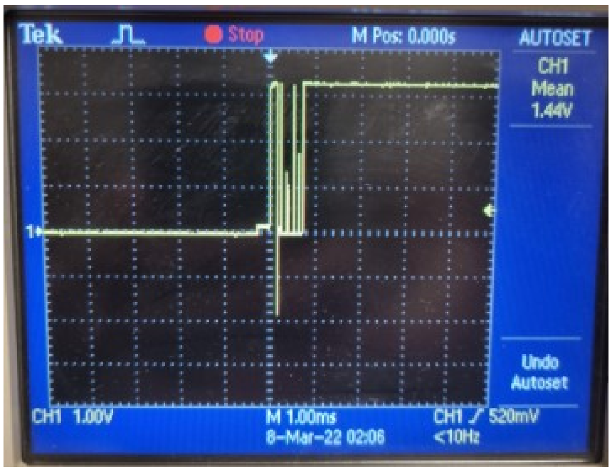


Figure 5.3.1: Button bounce

The button bounce was measured before the debouncing has been implemented and the results were obtained as expected. A bounce of 20ms was accounted for and designed in software.
The result after software implementation is shown below.

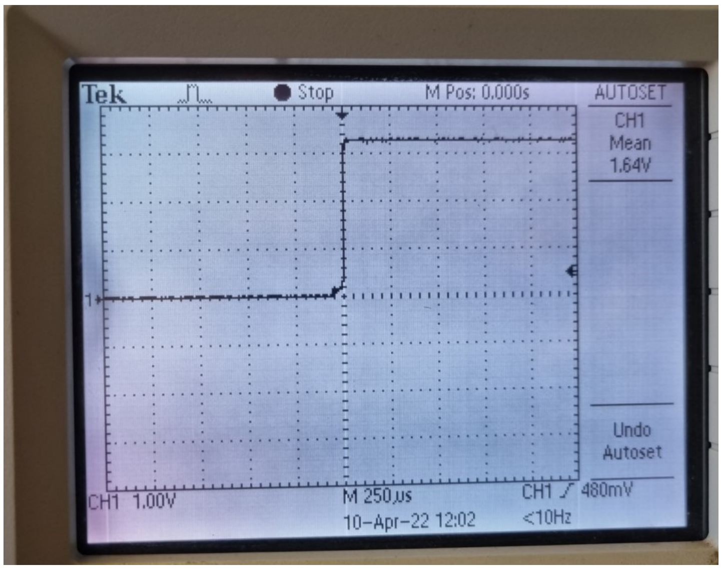


Figure 5.3.2: Debounced button

The button was successfully debounced. A clean rising edge as expected. The active low setup is also confirmed, a button that goes low when pressed and high when released.

5.4 LEDs

Our LEDs should be visibly bright but not too bright.

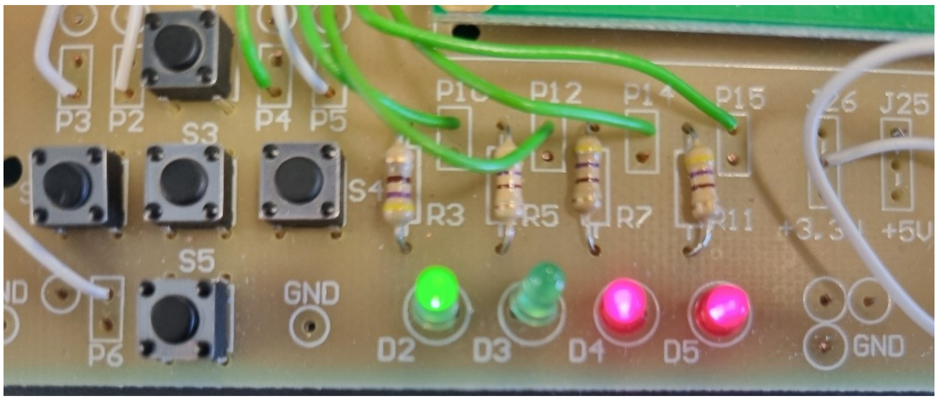


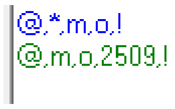
Figure 5.4.1: Debug LEDs brightness

The LEDs have a visible brightness which means the current has been successfully limited to give us a good brightness.

5.5 ADC

To test our ADC a DC voltage of 2500 was fed into voltage follower circuit which connects to the ADC pin PA0.

The only way to confirm our ADC was to check through the measurement request command from UART.



```
@,*,m,o,!
@,m,o,2509,!
```

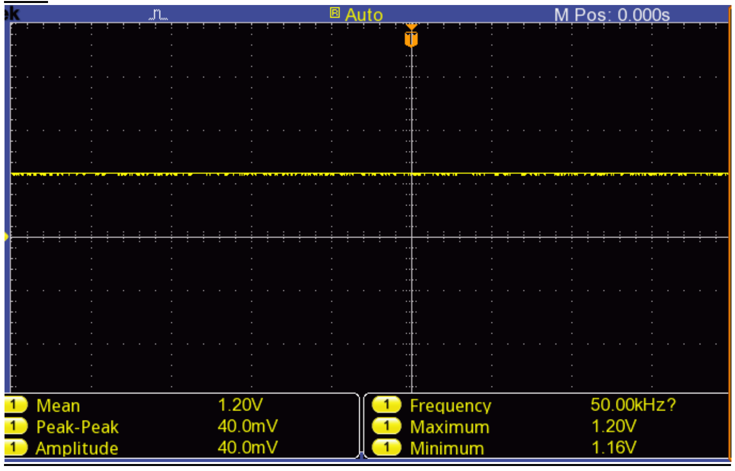
The ADC successfully measured a DC voltage value. A few more similar tests were run with varying voltage and yielded the same results.

The AC measurements weren't accurate. The measurements would sometimes be way off and will sometimes be perfect. This could possibly be because of a timing error. This could not be fixed in time.

5.6 DAC

The DAC was first tested with default values as given in the PDD[5] table 1 UR3.

DC:

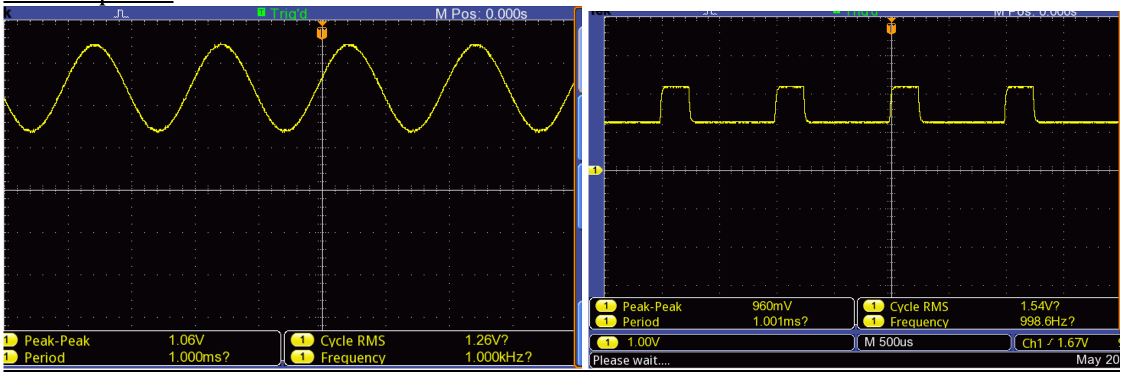


```
1 Mean        1.20V     1 Frequency  50.00kHz?
1 Peak-Peak   40.0mV    1 Maximum    1.20V
1 Amplitude   40.0mV    1 Minimum    1.16V
```

Figure 5.6.1: Default DC generation

A 1.2V offset DC signal was generated and the results for DC were deemed successful.

AC and pulse:



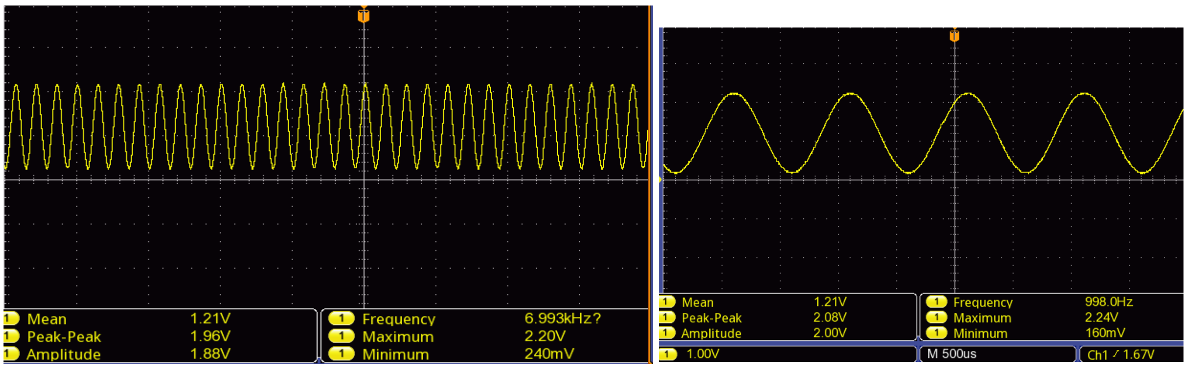
```
Trig'd          M Pos: 0.000s
1 Peak-Peak   1.06V        1 Cycle RMS   1.26V?
1 Period      1.000ms?     1 Frequency   1.000kHz?

1 Peak-Peak   960mV        1 Cycle RMS   1.54V?
1 Period      1.001ms?     1 Frequency   998.6Hz?
1 1.00V       M 500us      Ch1 ⁄ 1.67V
Please wait....            May 20
```

Figure 5.6.2: Default sine and pulse wave generation

A 1V pk-pk 1.2 offset and 1kHz sine wave was generated and thus the AC signal generation was deemed successful.
A 1V pk-pk 1.2 offset 25% duty cycle pulse signal was generated and the Pulse signal generation was deemed successful.

A test for variable values was done by changing the values in UART using the set parameter command.

```
@,^,a,2000,! | @,^,f,6988,!
```

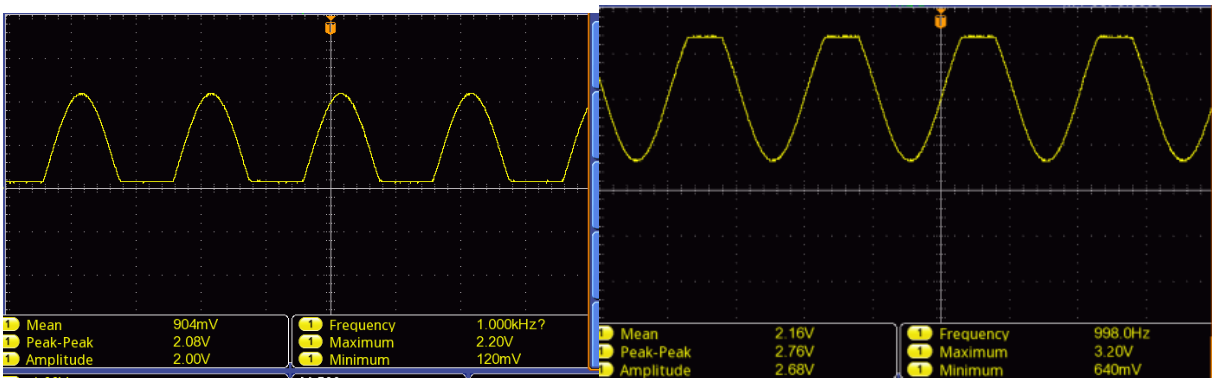


```
1 Mean        1.21V        1 Frequency   6.993kHz?
1 Peak-Peak   1.96V        1 Maximum     2.20V
1 Amplitude   1.88V        1 Minimum     240mV

1 Mean        1.21V        1 Frequency   998.0Hz
1 Peak-Peak   2.08V        1 Maximum     2.24V
1 Amplitude   2.00V        1 Minimum     160mV
1 1.00V       M 500us      Ch1 ⁄ 1.67V
```

Figure 5.6.3: Variable parameter values

The parameters were successfully changed and the signal generation corresponds with the change in parameter.

We also tested if the output clips at 3.2V and 0.1V as required by the PDD[5].



```
1 Mean        904mV        1 Frequency   1.000kHz?
1 Peak-Peak   2.08V        1 Maximum     2.20V
1 Amplitude   2.00V        1 Minimum     120mV

1 Mean        2.16V        1 Frequency   998.0Hz
1 Peak-Peak   2.76V        1 Maximum     3.20V
1 Amplitude   2.68V        1 Minimum     640mV
```

Figure 5.6.4: Output clipping

The output clipped at 3.2V and 0,120V according to Figure 5.6.4. The output is meant to clip at 3.2V and 0.1V. The output buffer circuit was thus successfully designed.

5.7 LCD

To test the LCD we used our character write function to display characters.

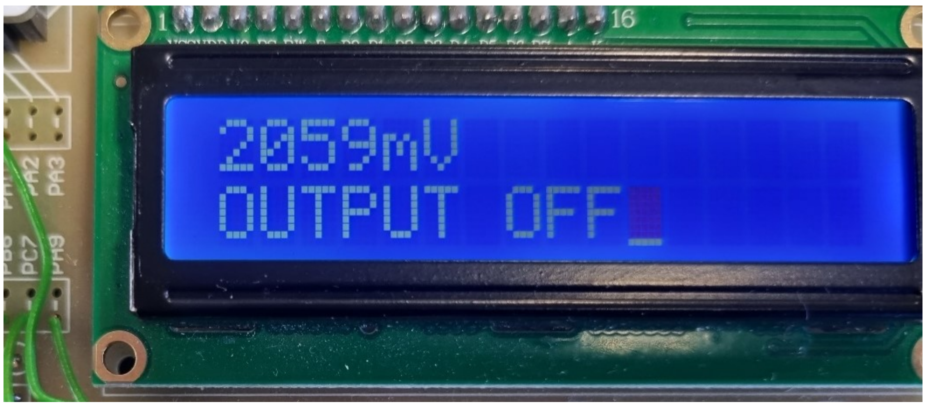


Figure 5.7.1: LCD Display 1

The LCD was successfully initialized and displayed characters with good visible contrast.

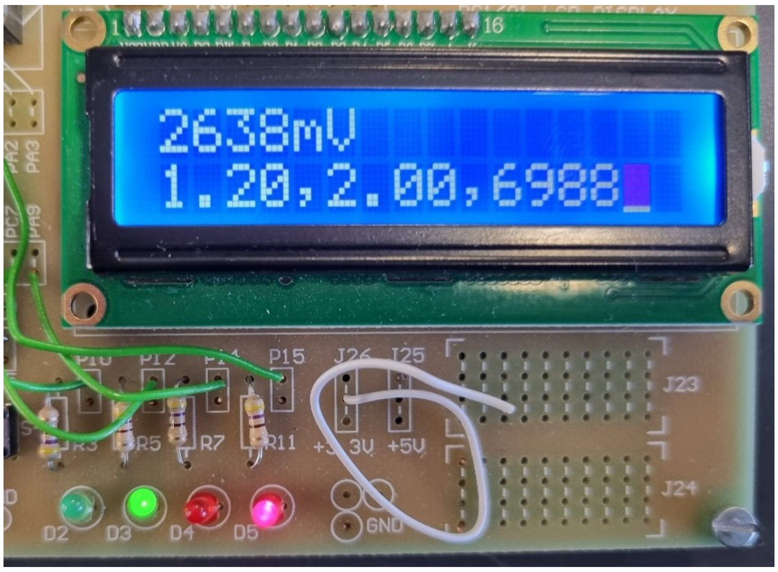


Figure 5.7.2: LCD Display 2

The LCD updated the display correctly depending on the UART or set parameter commands. The same parameters as shown in figure 5.6.4 reflected on the LCD.

5.8 Current sensor

Was not implemented

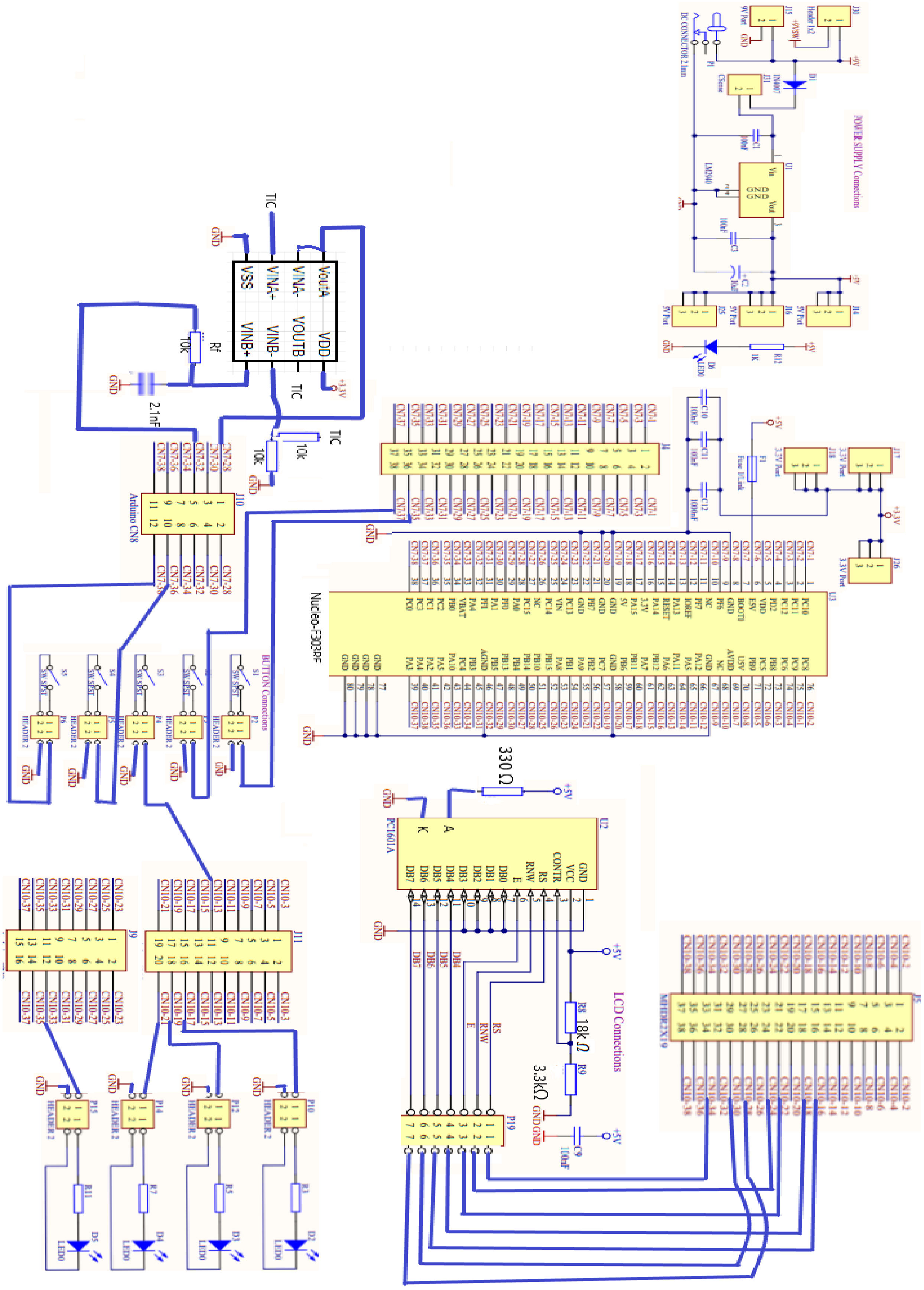
5.9 Complete system

The complete system was tested by changing to different states using UART and checking if the LCD displayed correct info. If the signal being generated on the oscilloscope from the DAC was correct according to the display. If the signal input into the ADC was measured and displayed correctly. All of the figures from 5.6.1-5.7.2 were done in one test to confirm the functionality of the whole system. The system worked well together although the ADC AC measurements were wrong the LCD and UART matched states. The DAC worked flawlessly with the whole system matching with the UART and LCD. The menu system was successful in changing the different states.

6 Conclusions

The system that was designed met user requirements . The voltage regulation was fully functional. The buttons were fully functional and successfully debounced. Along with the buttons the menu system was fully functional, the different states were set and the overall system(LEDs, UART ,signal generation, measurements) responded accordingly. The DAC was fully functional and created signals within 5% range of the adjustable parameters and within the required signal ranges. The ADC wasn’t fully functional. The DC measurements worked but the AC measurements gave varying results that were sometimes correct and sometimes not. This was due to a timing error. The ADC interrupted the process too fast before the measurement values were ready. This could be fixed by reducing the amount of instructions are used per function especially for loops that could last longer than needed. The current sensor was not implemented due to a shortage of time. Scrolling was not implemented. The scrolling caused timing issues with the DAC were the DAC would first wait for the scrolling to finish before changing the output signal. This can however be fixed using flags and variables but due to a shortage of time couldn’t be fixed. The current sensor was not implemented but a good tutorial was given that could be followed.[8]

7 Appendix



Complete schematic

PINS	Configuration
PA0	ADC1_IN1
PA1	TIM2_CH2
PA2	USART_TX
PA3	USART_RX
PA4	DAC1_OUT1
PA6	GPIO_EXTI6
PC4	GPIO_OUTPUT
PB1	GPIO_OUTPUT
PB2	GPIO_OUTPUT
PB11-PB14	GPIO_OUTPUT
PC7	GPIO_OUTPUT
PA9/10	GPIO_OUTPUT
PA15	TIM2_CH1
PB6	GPIO_OUTPUT
PC0-3	GPIO_EXTI0-GPIO_EXTI3

STM32 pins and their configuration

All other pins are default setup when selecting the NUCLEO-F303RE board.

8 References

[1] "This is information on a product in full production. STM32F303xD STM32F303xE," 2016, Accessed: May 24, 2022. [Online]. Available: www.st.com

[2] "00mA LOW DROPOUT VOLTAGE REGULATORS LM2950/1 FEATURES".

[3] "MCP601/1R/2/3/4 Features," 2007.

[4] O. Technology Co http, "Orise Tech SPLC780D1 Controller Datasheet".

[5] "PDD." https://learn.sun.ac.za/pluginfile.php/3452176/mod_resource/content/0/DesignE314_2022_PDD_v0.7.pdf (accessed May 24, 2022).

[6] "Zerø-Drift, Bi-Directional CURRENT/POWER MONITOR with I 2 C™ Interface Check for Samples: INA219 1FEATURES," 2008, Accessed: May 24, 2022. [Online]. Available: www.ti.com.

[7] "INA219," 2008, Accessed: May 24, 2022. [Online]. Available: www.ti.com

[8] "Overview | Adafruit INA219 Current Sensor Breakout | Adafruit Learning System." https://learn.adafruit.com/adafruit-ina219-current-sensor-breakout?view=all (accessed May 24, 2022).

[9] "DAC in STM32 » ControllersTech." https://controllerstech.com/dac-in-stm32/ (accessed May 24, 2022).

[10] "STM32 LCD 16x2 Tutorial & Library | Alphanumeric LCD 16x2 Interfacing." https://deepbluembedded.com/stm32-lcd-16x2-tutorial-library-alphanumeric-lcd-16x2-interfacing/ (accessed May 24, 2022).

[11] "Termite: a simple RS232 terminal." https://www.compuphase.com/software_termite.htm (accessed May 24, 2022).