Design (E) 314
Technical Report

Dot-Matrix Arcade Game Console

Author:
Altus Cilliers

Student Number:
23252162

21 June, 2021

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.

Handtekening / *Signature*	23252162 Studentenommer / *Student number*
AA Cilliers Voorletters en van / *Initials and surname*	June 21, 2021 Datum / *Date*

Abstract

This report covers the design and implementation needed in order to make an arcade style game console, in which two games can be played, a tennis game and a maze game. This project makes use of a 8x8 LED Dot Matrix as a screen in order to visualize the games being played. The user can make use of various inputs such as: buttons, an ADC Slider and an IMU in order to interact with the system. The hardware elements and implementation of each is also discussed and reasons for specific design choices are motivated. The software structure and design is also discussed, including details on timing, button debounce handling, how the LED Matrix is updated, control logic for the menu and which peripherals were used and how they are configured. Various measurements were taken and together with the results obtained are used to verify that each element in the system meets the requirements and works as intended.

Contents

List of Figures

List of Tables

List of Abbreviations

ADC Analog-to-Digital Converter

CS Chip Select

FSM Finite State Machine

GPIO General-Purpose Input/Output

I2C Inter-Integrated Circuit

IMU Inertial Measurement Unit

LED Light Emitting Diode

LSB least significant bit

MCU Microcontroller Unit

MEMS Micro Electro-Mechanical Systems

MOSFET Metal-Oxide Semiconductor Field-Effect Transistor

ms milliseconds

ns nanoseconds

PDD Project Definition Document

SCL Serial Clock Line

SDA Serial Data Line

TIC Test Interface Connector

UART Universal Asynchronous Receiver/Transmitter

List of Symbols

Ω Ohm

Hz Hertz

mA milli-Amp

V Volt

1 Introduction

The main goal of this project is to act as an arcade style game console with a 8x8 Light Emitting Diode (LED) Dot Matrix as a screen where two games can be played, namely a maze game and a tennis game. In the maze game the user can navigate the ball through the maze with the goal of completing the maze. The aim of the tennis game is to keep bouncing the ball against the back wall and not let the ball pass the left-most column. The system will be controlled by a STM32F103RB Microcontroller Unit (MCU), which will take user inputs, do computations etc. and be used to play the two games. The system should make use of Universal Asynchronous Receiver/Transmitter (UART) in order to transmit certain message packages that contain information regarding system information and measurements (for example x and y positions of the ball during a game) to the Test Interface Connector (TIC) for verification and testing purposes, these UART packages can be seen in table 2. Input requirements are that a user must be able to control the system with push buttons, Analog-to-Digital Converter (ADC) slider and an accelerometer. The system has 4 main states; Menu, Maze Selection, Maze Game and Tennis Game; and makes use of a finite state machine in order to control the menu logic for switching between these states. All the requirements and specifications are obtained from the [1] Project Definition Document (PDD).

This report will include a system diagram with all the components of the system displayed, the various required hardware elements and how they were setup. This is followed by the software section which gives a high-level overview of the program, explains the menu control logic, how button debounce was handled, Inertial Measurement Unit (IMU) interfacing and details which peripherals were used and how they were setup and initialized.
The report also includes a section on which measurements and tests were done in order to verify that all the system requirements were met and adhered to as described by the [1] PDD.
Finally the report will contain the findings and conclusions detailing which requirements were met/not met, identify shortcomings in the design and include some recommendations for future improvements.

In the appendices the complete circuit diagram(see figure 21, appendix A) and pinout for the STM32F103RB MCU(see table 4, appendix B), along with a photo of the fully built physical project (see figure 22, appendix C) are included such that any person with some understanding of electrical circuits can build the circuit themselves.

2 System description

The system block diagram is shown in Figure 1.

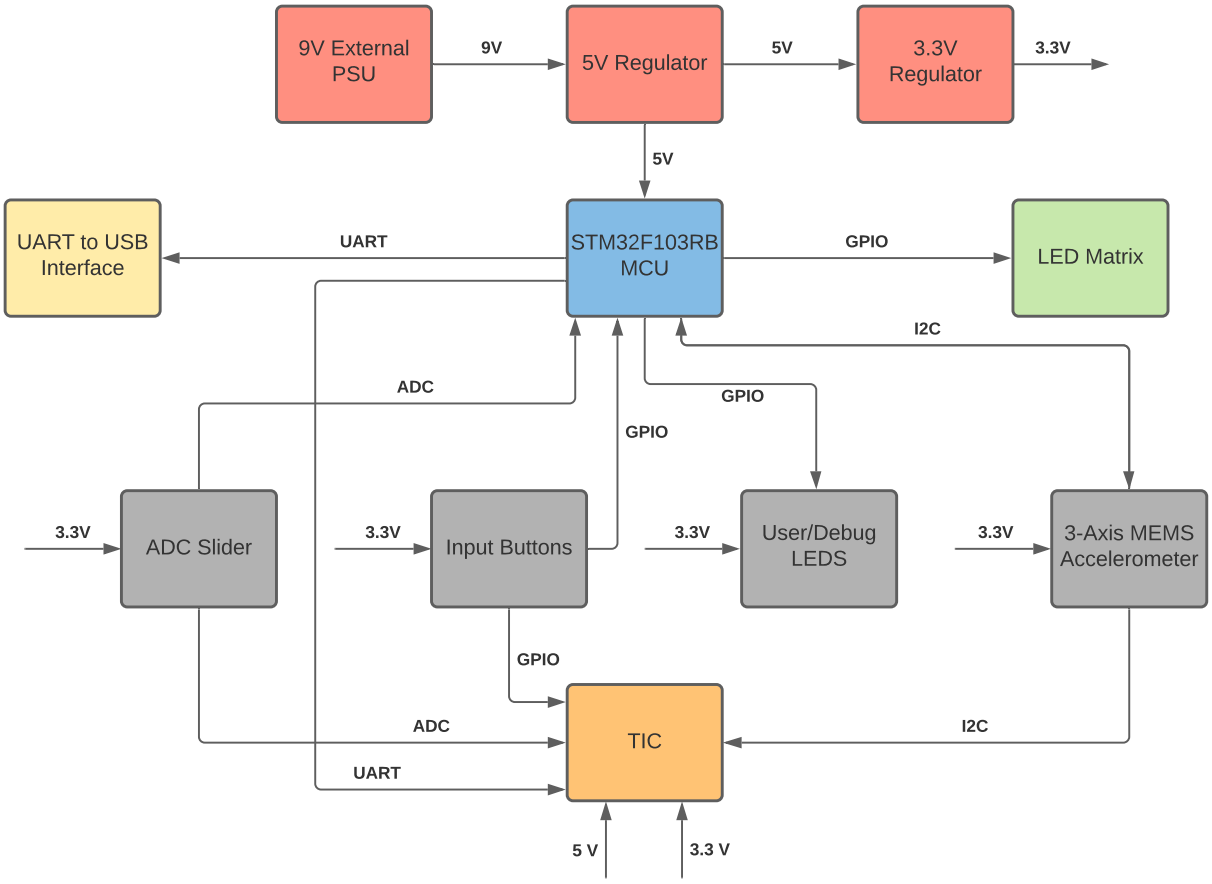


Figure 1: System Block Diagram

Table 1 shows a list of components and their operating voltages in this project.

Table 1: Your table caption

Component	Operating Voltage
5V LM7805 regulator	9 V
3.3V MCP1700 regulator	5 V
STM32F103RB MCU	5 V
LED Matrix	3.3 V
ADC Slider	3.3 V
3-axis IMU Accelerometer	3.3 V
User/Debug LEDs	3.3 V
Input Buttons	3.3 V

Power Supply:
An external 9 Volt (V) power supply is supplied to a 5V LM7805 regulator which is used to power the STM32F103RB MCU board. This regulated 5V is then further regulated using a 3.3V MCP1700 regulator, this regulated 3.3V is used to power most of the external components.

STM32F103RB MCU:
The STM32F103RB MCU board is used to communicated with all the other components used in the

project. General-Purpose Input/Output (GPIO) is used to control the LED matrix and the debug LEDS, UART is used to send system information to the TIC in order to validate the state and measurements of the system. It is also used to interpret data from ADC or Inter-Integrated Circuit (I2C).

LED Matrix:
The LED matrix consists of 64 LEDs in a 8x8 grid, each row/column connected to a GPIO pin. The STM32F103RB MCU drives these LEDs using GPIO pins.

ADC Slider:
The ADC slider gets powered by 3.3V from the MCP1700 regulator, it is connected via ADC to the STM32F103RB MCU, it is also connected to the TIC. The ADC slider acts as a potentiometer which provides a analog voltage level depending on the position of the slider, the logic of converting this analog signal to useful digital values is handled by the STM32F103RB MCU.

3-axis IMU Accelerometer:
The IMU slider gets powered by 3.3V from the MCP1700 regulator and it is connected via I2C to the STM32F103RB MCU, it is also connected to the TIC.

User/Debug LEDs:
The User/Debug LEDs are driven by GPIO pins and are used for debugging purposes and also to display the current maze being selected in the maze game.

Input Buttons:
5 Buttons will be used as inputs in order to navigate the "menu" of the games. The buttons are wired up as active low buttons and they are connected to the TIC and the STM32F103RB MCU as GPIO inputs.

UART Interface:
UART is used to send system information to the TIC in order to validate the state and measurements of the system.

TIC:
The TIC receives various inputs and measurements from components in order to check and validate the measurements and state of the system for auto-marking purposes.

3 Hardware design and implementation

3.1 Power supply

A 9V external power supply gives power to the entire system, it is stepped down to $5V$ using the LM7805 voltage regulator as seen in figure 2. The LM7805 regulator was built according to the specifications in the 7805 datasheet [2]. It was then further stepped down to $3.3V$ using the MCP1700 voltage regulator as seen in figure 3 which was built according to typical application specifications found in the MCP1700 datasheet [3]. The $5V$ output is used to give power to the MCU and the $3.3V$ provides power to various other components. In order to verify that the power supply circuit is functioning and that there is power on the board, a green LED is built into the power supply circuit, which lights up when powered.

The design of the LM7805 $5.5V$ regulator can be seen in figure 2. A Diode D1 is used for reverse current protection. The $5V$ regulator has a ceramic input capacitor which filters out high frequency noise from the source, there is also a second $100nF$ ceramic capacitor at the output which filters additional high frequency noise to ensure a clean regulated output signal. There is also a 100nF electrolytic capacitor in parallel with the output, this capacitor protects the circuit and components from voltage spikes and sudden power loss.

The MCP1700 $3.3V$ regulator as seen in figure 3 has input and output $100nF$ capacitors to filter out high frequency noise. As per the MCP1700 datasheet [3] typical application circuit we can provide an input voltage of 5V and the output voltage will be 3.3V.

In both regulators the ceramic capacitors should be placed as close to the regulators are practically possible.

Refer to figure 21 in appendix A for the complete power supply diagram.

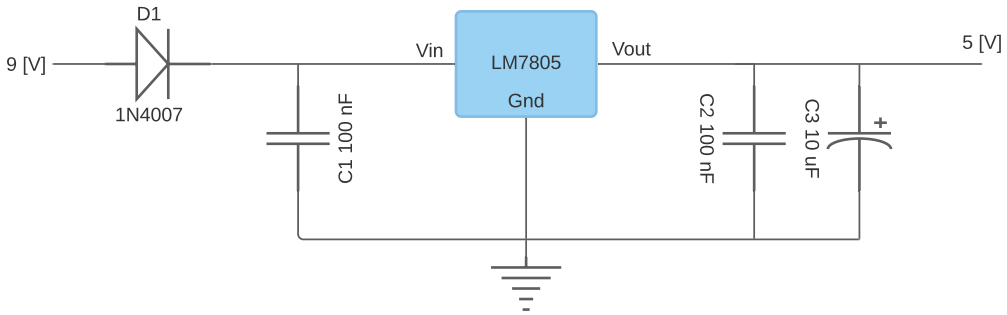


Figure 2: 5 [V] Regulator Diagram

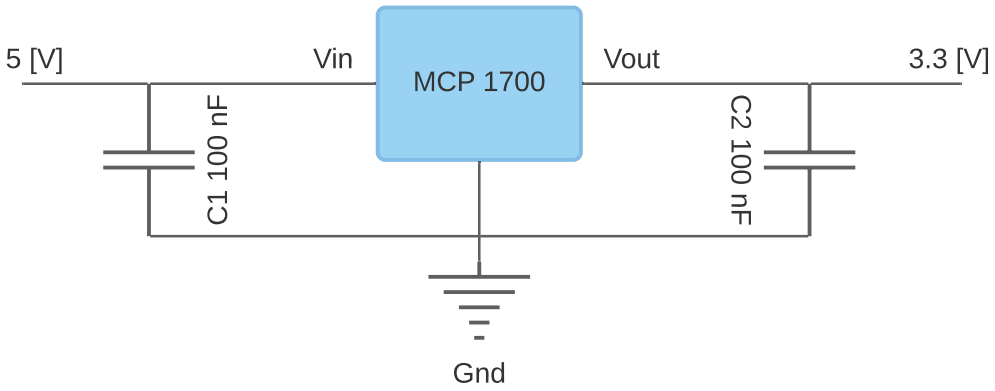


Figure 3: 3.3 [V] Regulator Diagram

3.2 UART communications

The project uses UART in one-directional transmit mode only in order to communicate with the TIC to send system information in order to validate the state and measurements of the system. The protocol is set up in a 8N1 configuration (8 data bits, no parity bit with one stop bit), at a baud rate of 115200 bits per second, refer to figure 4 for a visual representation(figure obtained from ECE353 [4]). The default UART2 channel was used on the STM32F103RB MCU nucleo board, which is connected to the ST-Link chip on the nucleo board, thus when plugging in the nucleo board to a PC, a virtual COM port will be available and thus the UART output can be viewed with an external program such as Teraterm. In order for the UART to communicate with the TIC a physical connection has to be made from the UART2 pin to the test station.
UART messages will make use of ASCII characters and consist of a fixed length of 10 characters. See table 2 to see the layout of the UART message fields.

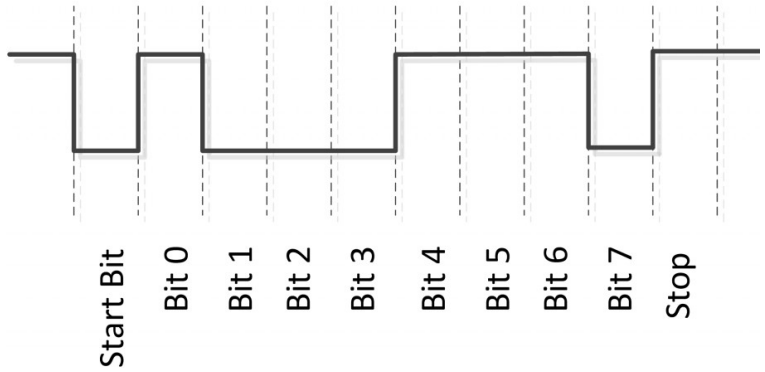


Figure 4: 8N1 UART Transmission example

MESSAGE	0	1	2	3	4	5	6	7	8	9
StdNum	Student Number, 8 bytes									'\n'
Calibration	'$'	'1'	Column	'_'	'_'	'_'	'_'	'_'	'_'	'\n'
Tennis	'$'	'2'	X pos ball	Y pos ball	'Velocity'	'Direction'	'X pos bat'	'Y pos bat'	'IMU'	'\n'
Maze	'$'	'3'	X pos	'Y pos'	'Visible ball'	'Visible goal'	'IMU'	'_'	'_'	'\n'

Table 2: UART Message fields

3.3 Buttons

5 Push Buttons will be used as inputs to play the various games and to navigate the "menu" of the games. The buttons are wired up as active low buttons with a pull-up resistor in order to prevent a floating(undefined) logic level. The pull-up resistor pulls the input pin high when the button is not pressed and when the button gets pressed the GPIO input pin is connected directly to ground and current flows through the resistor to ground. Thus when the button is pressed it reads a low state hence the name active-low. The pull-up resistor also limits the amount of current flowing into the GPIO pin because without the pull-up resistor if the button is pressed it would connect V_{CC} directly to ground causing a short. According to sparkfun [5] a typical pull-up resistor value is $10k\Omega$, which is what was used in this project for the design of the button.

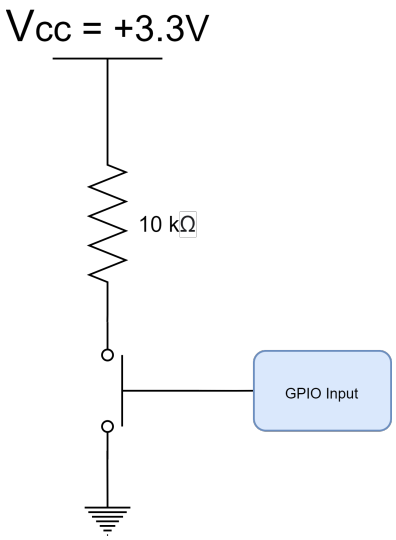


Figure 5: Active Low Button Diagram

3.4 Debug LEDs

Four debug\user LEDs are used for debugging purposes and to display which maze is currently selected. A resistor is wired in series with the LED in order to limit the amount of current that can flow through

the LED. The maximum current that can be sourced by a GPIO output pin is 20 milli-Amp (mA), thus the minimum resistance value for the debug LED can be calculated as follows:

$$R = \frac{V}{I}$$
$$R = \frac{3.3}{20 \times 10^{-3}}$$
$$R = 165\ \Omega$$

3.5 LED Dot Matrix

An 8x8 LED Matrix (Refer to figure 6) was chosen for this project to be used as a screen in order to visually represent the games being played. Resistors are used in series with the LEDs in order to limit the current flowing through them in order to protect the LEDs and also to limit the current being sunk/sourced by the GPIO pins.

Metal-Oxide Semiconductor Field-Effect Transistor (MOSFET)S are used to ensure that all the LEDs stay at a constant brightness. The MOSFETS require a small voltage to turn on, but they can drive large amounts of current, which we need because we want to power 8 LEDs in series per row and our GPIO pins cannot sink the required amount of current, thus a MOSFET is used. The MOSFETS also allow for an easy way to switch a whole row to high/low by sending a high/low to the gate of the MOSFET.

A design choice was made to solder the 8x8 LED matrix a Veroboard which was provided to us. A second design choice was made to solder the Veroboard to the PCB using header pins. These two design choices make it more modular and easier to access all the components if any changes or repairs need to be done in the future.

Resistors R1-R8 as seen in figure 6 for the LEDs are needed in order to limit the amount of current that passes through them to prevent them from burning out. They also limit the total amount of current that is sunk from the GPIO pins. According to the STM32F103RB datasheet [6], the GPIO pins can sink a maximum of 8 mA when more than 8 pins are sunk simultaneously. The output voltage of GPIO pins is $3.3V$, and the turn on voltage of a typical red LED is $1.8V$ (Vd) and from the datasheet the lowest high-level voltage is guaranteed as $Vdd - 0.4$, thus the value of the resistors are calculated as follows:

$$R_{(min)} = \frac{V_{DD} - V_D - 0.4}{I_{max}}$$
$$R_{(min)} = \frac{3.3 - 1.8 - 0.4}{8 \times 10^{-3}}$$
$$R_{(min)} = 137.5\ \Omega$$

Resistor values of 330 Ohm (Ω) were chosen because they make the LEDs bright enough while being above the 137.5 Ω limit.
GPIO and LED Current:
The current sunk by 1 GPIO pin is calculated as follows:

$$I = \frac{V_{ref} - V_D - 0.4}{R}$$
$$I = \frac{3.3 - 1.8 - 0.4}{200}$$
$$I = 5.5\,\text{mA}$$

Therefore, the total current sunk is 8 (columns) x 5.5 mA, which equals 44mA. The LED current (current that flows through 1 LED) is 5.5 mA.

The duty cycle of each LED is:

$$\frac{time\ on\ (ms)}{Period} \times 100$$

It takes 8ms to update the whole LED matrix so the period is 8s, therefore each LED is on for 1ms. We can then calculate the duty cycle as follows:

$$DutyCycle = \frac{1}{8}$$
$$DutyCycle = 12.5\%$$

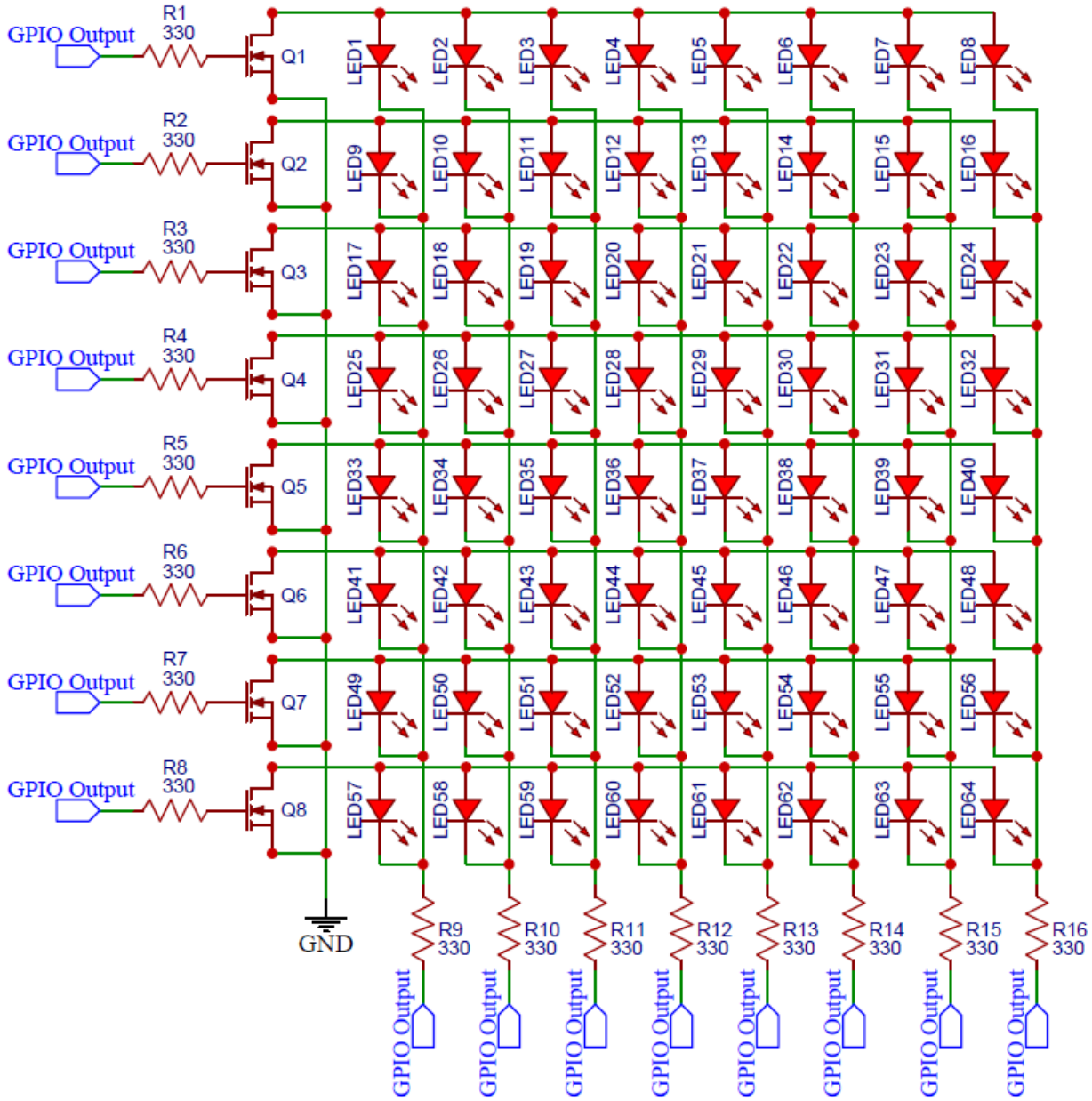


Figure 6: Schematic of 8x8 LED Matrix

3.6 ADC Slider

An ADC slider was chosen to be used as an input for the tennis game to move the bat up and down. This is achieved by using a potentiometer which provides a analog voltage level depending on the

position of the slider, which we then scale and convert to a digital voltage using software. According to the ADC Slider datahseet [7] we need to connect the ADC slider pins as shown in figure 7.

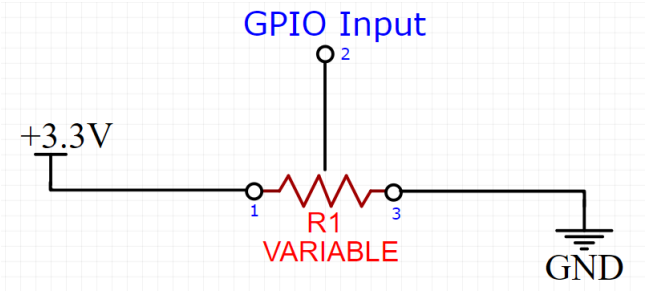


Figure 7: ADC Slider Circuit Diagram

3.7 3-axis IMU Accelerometer

A 3-axis Micro Electro-Mechanical Systems (MEMS) accelerometer was chosen to be used as an input method in order to move the ball through the maze in the maze game and to move the tennis bat in the tennis game.
Figure 8 provides a circuit diagram of how the IMU should be connected. According to the LIS3DH datasheet [8] the Chip Select (CS) pin is pulled high in order to set the IMU into I2C mode. The SDO/SA0 pin is used to set the least significant bit (LSB) of the device's slave address, pulling the pin high sets the LSB to '1'. According to the LIS3DH datasheet [8] pull-up resistors R1 & R2 must be connected to the Serial Data Line (SDA) and Serial Clock Line (SCL) pins. These pull-up resistors prevent floating logic levels by forcing the SDA and SCL to the supply voltage (high) when no data is being transmitted.

The value of these resistors can be calculated according to the formula in the I2C Bus Pull-up Resistor Calculation datasheet [9].

$$R_{p_{min}} = \frac{V_{CC} - V_{OL_{(max)}}}{I_{OL}}$$
$$R_{p_{min}} = \frac{3.3 - 0.1(3.3)}{8 \times 10^{-3}}$$
$$R_{p_{min}} = 371.25\Omega$$

A resistor value of 560 Ω was chosen as this was a standard, readily available resistor above the minimum required resistance.

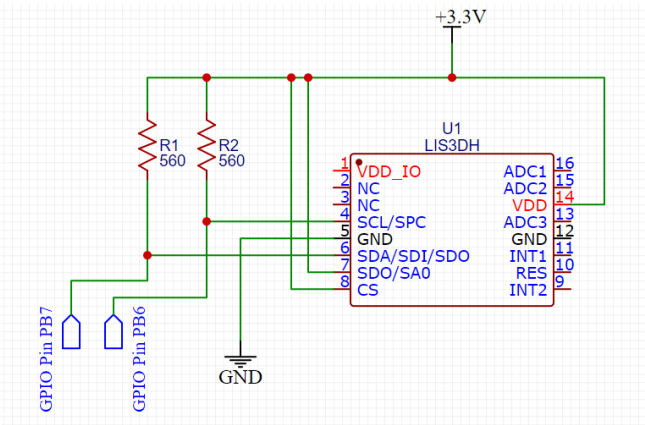


Figure 8: IMU Accelerometer Circuit Diagram

4 Software design and implementation

4.1 High-level Description of Program

The goal of the project is to act as an arcade style game console where two games can be played, namely a maze game and a tennis game. The system will be controlled by a STM32F103RB MCU, which will take user inputs, do computations etc. and be used to play the two games. The system has 4 main states: Menu, Maze Selection, Maze Game and Tennis Game.

The STMCubeIDE v1.3 Integrated Development Environment will be used for this project. The main structure of the code is using a timer-based schedule(making use of the SysTick Interrupt Handler) inside the main while loop, with checks that will react to flags if certain conditions are met like if a button is pressed etc. and call the appropriate functions/interrupt handlers.

Upon power-up the system will begin automatically with a calibration sequence and after it is completed go to the menu/waiting stage.
The first game(maze game) can be entered by pressing the left button to first go to the maze selection state, where pressing the up and down buttons cycle through the different maze options(maze1-4), upon pressing the middle button the current maze is selected and the maze game state is entered. In the maze game the user can navigate the ball through the maze by making use of the push buttons or the *IMU as input devices. The user can exit the maze game and go to the main menu by either completing the maze or by pressing the middle button.

The second game(tennis game) can be entered by pressing the left button(while in the menu state). The aim of this game is to keep bouncing the ball against the back wall and not let the ball pass the left-most column. The user can move the bat around by making use of the push buttons or the *IMU as input devices. The user can exit the tennis game and go back to the main menu by either pressing the middle button or by losing the game.

The entire LED matrix is updated using a technique called row scanning. This entails setting a row high and then setting each column high, then setting the next row high and repeating the process. The entire LED matrix is updated every 8 milliseconds (ms), hence each row is activated for 1ms before moving on to the next row.

*In the student's case he was not able to correctly implement the IMU in software, however it was mentioned in the overview as an example of how it would be used.

4.2 Control Logic

Figure 9 shows the a Finite State Machine (FSM) diagram to illustrate the control logic of how the system changes between various states like the maze/tennis game etc.
This control logic is placed inside the infinite main while loop and if certain buttons are pressed or certain conditions are met a flag is set and then a function is called to change the state of the system.

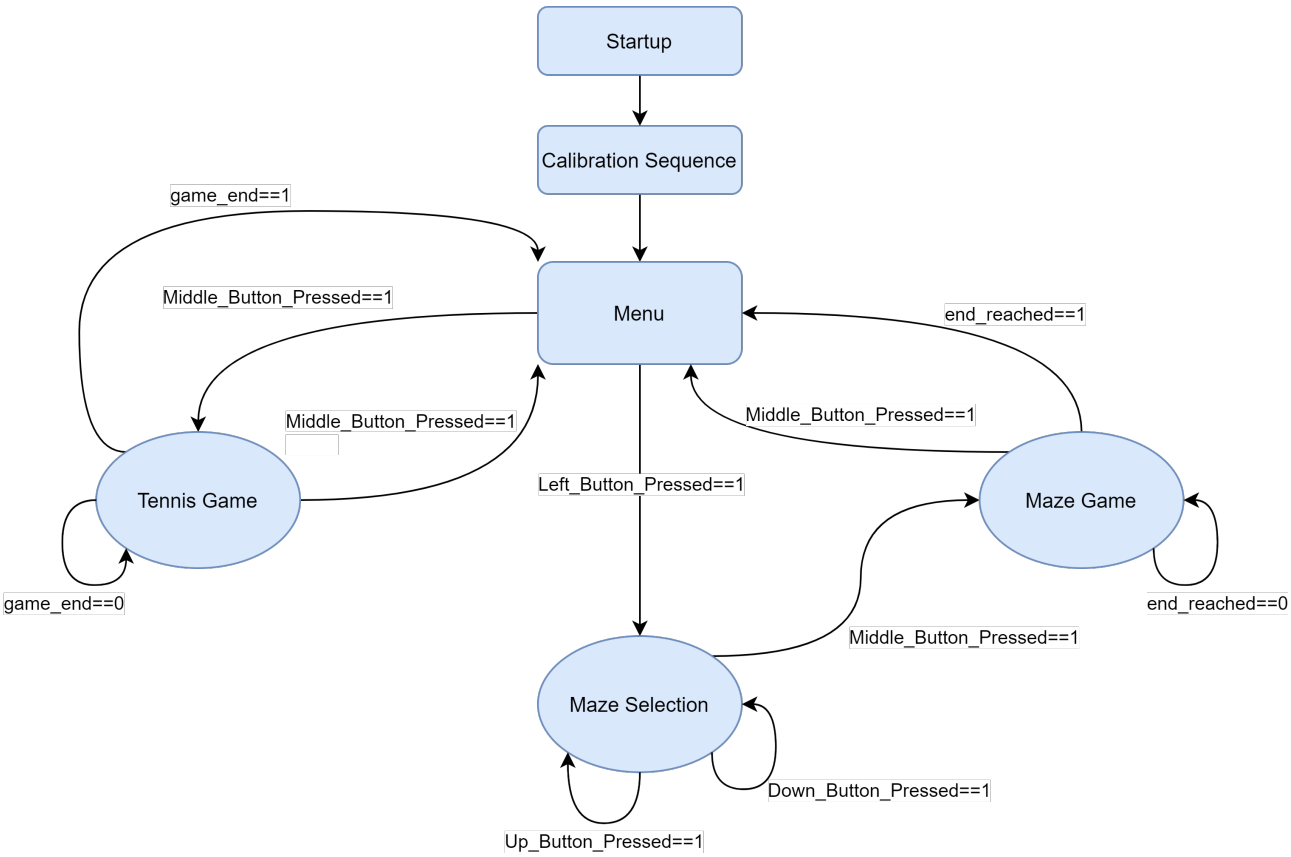


Figure 9: Control Logic FSM

On power-up the system automatically starts with the calibration sequence, afterwards it goes to the menu state(four corner LEDs light up). In order to exit the menu one needs to press either the left or the middle button. Pressing the left button changes the state of the system to the maze selection state. Inside the maze selection state pressing the up or down buttons cycles between the different mazes 1-4, in order to select a maze the middle button must be pressed. After the middle button has been pressed to select a maze the system goes into the maze game state, here you can play the maze game and the system can exit back to the menu state by either pressing the middle button or by the player finishing the maze.

When the system is in the menu state the tennis game state can be entered by pressing the middle button. Inside the tennis game state the player can play the tennis game and the system can exit back to the menu state by either pressing the middle button or by the player losing the tennis game.

4.3 Button Bounce Handling

Button debouncing was implemented via software, figure 10 shows the logic of how button debounce was implemented in this project. A design choice to debounce for 50 ms was made even though the physical button only bounces for a few micro-seconds, in order to provide a safe buffer margin. The code displayed in the flow diagram is placed inside the systick timer and continuously checks (every 1ms) if a button was pressed. Upon start-up the button pressed flag and button bounce counters are set to 0. If a button is pressed the system checks if 50ms has passed and if true it registers a button press and sets a button pressed flag = 1 and resets the button debounce counter. If the condition is false the system increments the button bounce counter and goes back to checking if a button was pressed. If a button pressed flag was set to 1, code in the tennis or maze game detects that flag = 1 and performs the required button action and resets the button pressed flag=0;

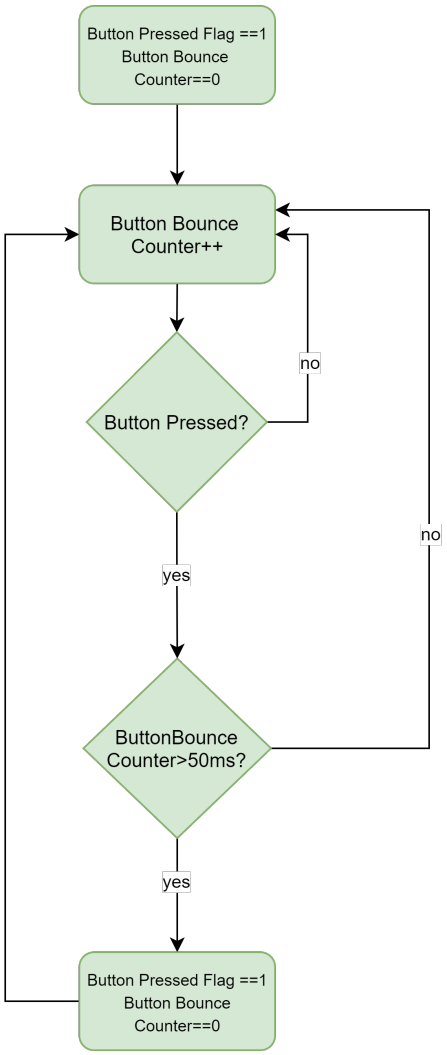


Figure 10: Button Debounce Flow Diagram

4.4 Data Flow and Processing

For menu/control logic of how to switch between different states please refer to section 4.2.
For simplicity sake the UART flags and timing has been left out of the flow diagrams.
Figure 11 shows a flow diagram of the logic used in the coding of the maze game. Every 300ms the maze UART package(see table 2) should be transmitted.
Figure 12 shows a flow diagram of the logic used in the coding of the tennis game, in this case the tennis UART package(see table 2) should be transmitted every 100ms.

Data flow and processing regarding the ADC is as follows:
The ADC samples a analog voltage between 0 and $(2^{12}-1)$, this value is then sent to the MCU via a GPIO Input Pin. In software the analog voltage is scaled and converted to a digital value between 0 and 7 using $\frac{sampled\_value*7}{(2^{12}-1)}$. With regard to the data processing of the IMU, no comment can be made because the student was unable to correctly implement the IMU in software.

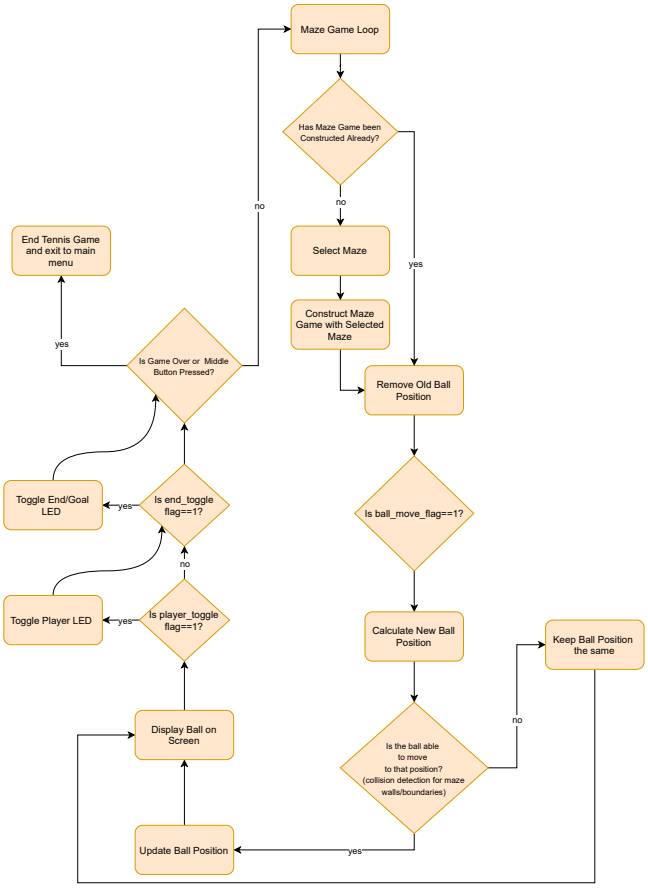


Figure 11: Maze Game Flow Diagram

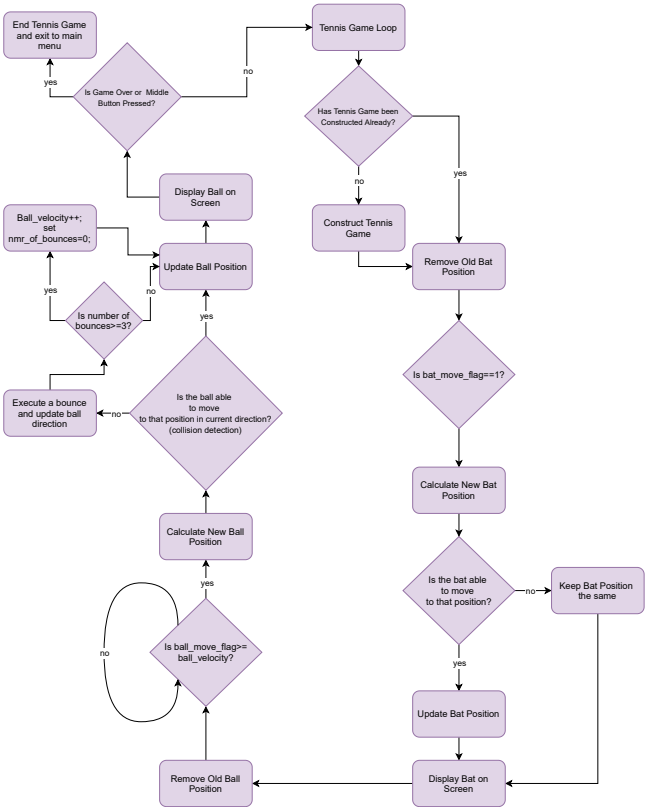


Figure 12: Tennis Game Flow Diagram

4.5 IMU interface

In order to setup the IMU, the application note [10] for the LIS3DH IMU needs to be consulted. Firstly Control Register 1 needs to be configured as follows:

- Setting the address to 0x20(control register 1's address)
- Writing 0x57(this value is to select normal mode at 100 Hertz (Hz)) into control register 1's address.

Then Control Register 4 needs to be initialized.

- Setting the address to 0x23(control register 4's address)
- Writing 0x0(this value is to select default values) into control register 4's address.

In order to read data from the IMU the following is done:

- Setting the address to 0xA8(The address for the OUT_X_L acceleration data is 0x28 and setting the most sigificant bit of this address=1 enables auto increment mode, hence giving the value of 0xA8)
- Then doing a HAL receive command and storing the received data into variables to be used in order to calculate angles.

4.6 Peripheral Setup

ADC
For the ADC setup the mode was select as "IN0" in the IOC file. Setup setings are:

- 12 bit resolution
- Single channel conversion using ADC_CHANNEL_0
- Sampling time: 1.5 x ADC clock cycle = 1.5 x 4Mhz = 116.67 nanoseconds (ns)

UART
USART2 was selected for uart, it is set up in 8N1 mode (8 data bits, no parity bit with one stop bit). Setup Settings:

- Baud Rate: 115200 bits per second
- Mode: Asynchronous
- Word Length: 8 Bits
- Transmit Only

Systick Timer
The systick timer is automatically setup to provide a tick every 1 ms, but the clock settings that were used in order to achieve this is as follows:

$$Systick\ Timer\ Frequency = \frac{CortextSystemTimerClockFrequency}{8000}$$
$$Systick\ Timer\ Frequency = \frac{8MHz}{8000}$$
$$Systick\ Timer\ Frequency = 1000Hz$$
$$\therefore Systick\ Timer\ Tick\ Time = \frac{1}{1000Hz} = 1ms$$

GPIO
The GPIO pins for the buttons were setup as GPIO Inputs and the GPIO pins for the rows and columns of the LED Dot Matrix along with the four Debug LEDs were setup as GPIO Outputs.
I2C

- Mode: Standard Mode
- Clock Speed: 100kHz

5 Measurements and Results

In this section any specifications or requirements mentioned are obtained from the PDD [1].

5.1 Power Supply

The Power supply specifications were that an external 9V supply would be supplied and needs to be stepped down using LM7805 5[V] and MCP1700 3.3[V] voltage regulators to within 5% tolerance.
The power supply circuit was connected to a external power supply source in the Laboratory and various input voltages were supplied and the voltage across both the $5V$ and $3.3V$ lines were measured. Table 3 depicts these results, as we can see from an input voltage of $7V$ both voltage regulators are providing the required outputs. The $5V$ voltage regulator output tolerance is 2.4% and the $3.3V$ voltage regulator output tolerance is 0.606%, thus both are within the required 5% margin and are working correctly as intended.

Input Voltage[V]	4.5	5	5.5	6	6.5	7	7.5	8	8.5	9
5[V] Regulator	2.57	2.83	3.34	3.84	4.28	5.12	5.12	5.12	5.12	5.12
3.3[V] Regulator	2.57	2.83	3.32	3.32	3.32	3.32	3.32	3.32	8.5	3.32

Table 3: Power Supply Measurements

5.2 UART Communications

Verifying that the UART Transmissions are working were done in multiple ways.
First the STM32F103RB MCU board was connected via a usb cable to a PC and a external program called Teraterm was used to create a virtual serial communications port. The baud rate was selected as 115200 bits per second and then the output could be viewed and verified that it is working. Figure 13 shows a UART Transmission of the Maze Message.

Secondly the UART Pin was connected to an oscilloscope and the output was measured and recorded, figure 14 shows a UART Transmission of the student's student number and the first column of the calibration sequence.

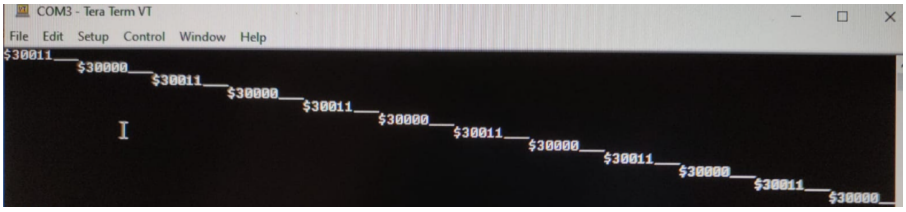


Figure 13: UART Maze Message Transmission

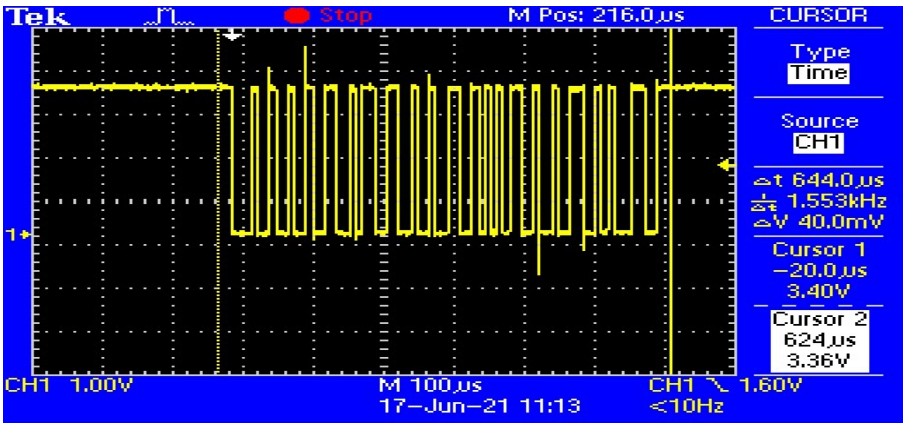


Figure 14: UART Output Upon Startup

5.3 Buttons

The 5 input buttons were required to be wired up as active low and required to have no button bounce. The buttons were tested by connecting oscilloscope probes to either side of the button and measuring what happens when a button is pressed. Figure 15 shows that the voltage is a high when the button is not pressed and when the button is pressed it pulls the pin low and a 0 voltage is measured. Thus we can see that the button is working as intended as an active low button.
Figure 16 shows an example of button bounce happening, a design choice was made to handle button debounce in software, for more detail on how this was achieved please see section 4.3.

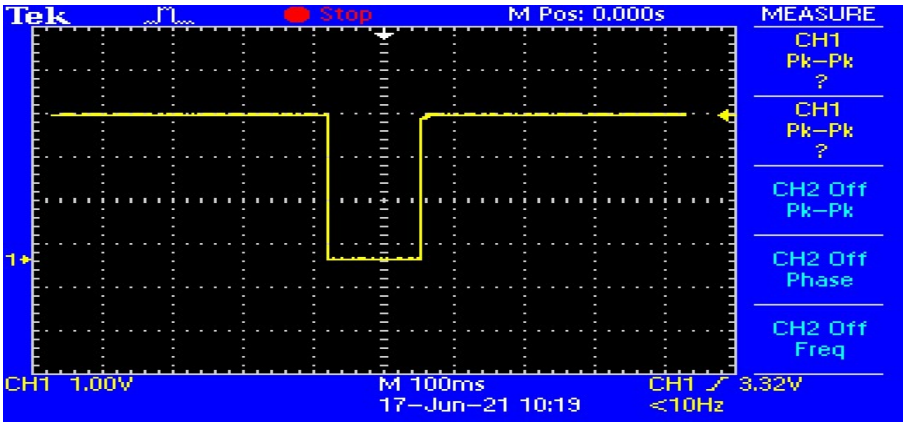


Figure 15: Button Active Low Press

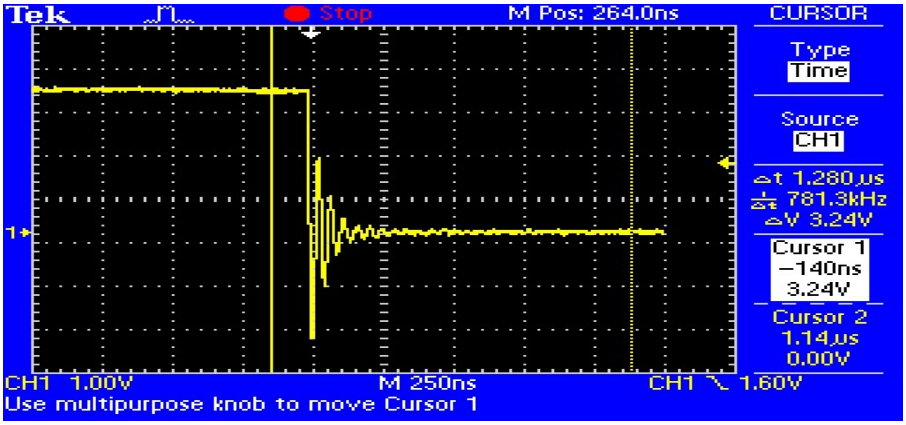


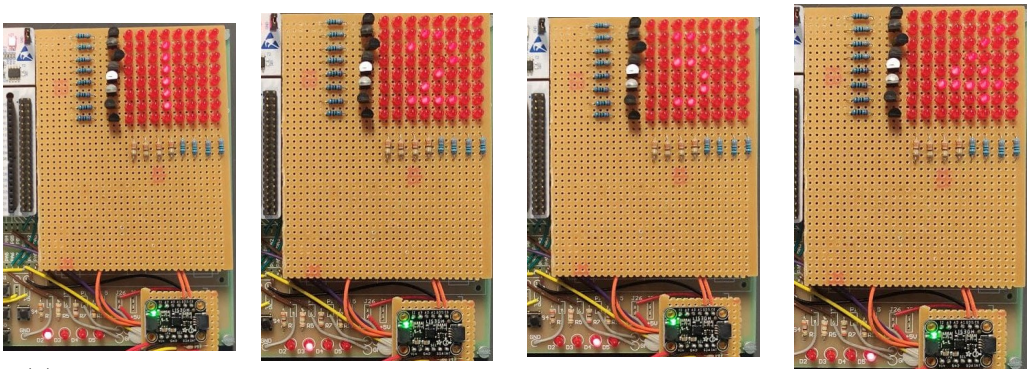
Figure 16: Button Bounce when Pressed

5.4 Debug LEDs

The debug LEDs were required to light up to show which maze was currently being selected, they also had to have a series resistor in order to limit the amount of current flowing through them. As seen in figure 18 the voltage drop over one Debug LED was measured with an oscilloscope to be $1.32V$, thus the current that flows through one Debug LED can be calculated as follows:

$$I = \frac{V}{R}$$
$$I = \frac{1.32}{390\ \Omega}$$
$$I = 3.38\,\text{mA}$$

We can then verify that the debug LEDs work by visually inspecting them as seen in figure 17.



(a) Debug LED1 (b) Debug LED2 (c) Debug LED3 (d) Debug LED4

Figure 17: Debug LEDs

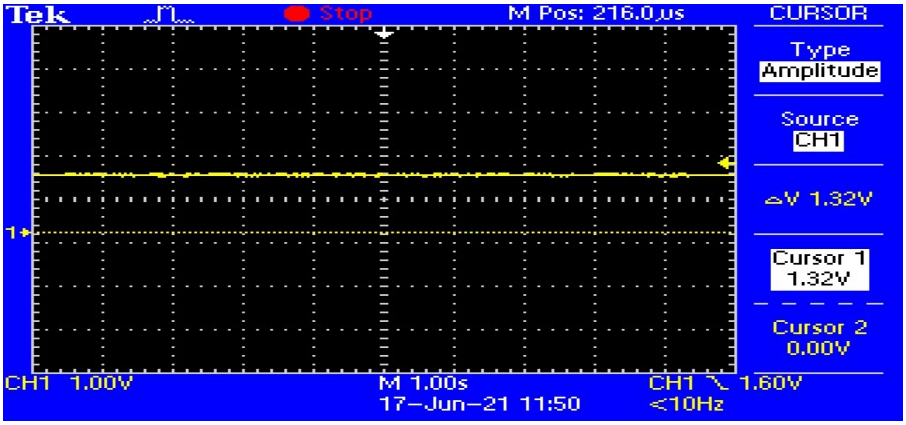


Figure 18: Debug LED Voltage

5.5 LED DOT Matrix

The LED DOT Matrix needed to have a consistent brightness and update each row with a technique called row scanning. The entire matrix should be updated once every 8ms.
In order to test if the LED matrix is working properly we can first do so by visual inspection. Software code was written by making use of the HAL Library functions to set certain rows and columns high to turn the corresponding LED on. After verifying that the expected LED to turns on, the calibration sequence (each column turns on separately) and was run, thus verifying that all the LEDs work and turn on when they are supposed to.

In order to test the timing requirements of the LED DOT Matrix, the first row(the student implemented row scanning, as his MOSFETs were connected in his rows) of the matrix was connected to an oscilloscope and then time it took for the row to turn on twice was measured. As seen in figure 19 it takes exactly 8ms between the first row turning on once and then turning on again, thus proving the entire LED matrix gets updated once every 8ms.

The amount of current that flows through one column when all 8 LEDs were on was also measured as $3.94mA$, thus being below the designed $5.5mA$ and complying with the specifications.

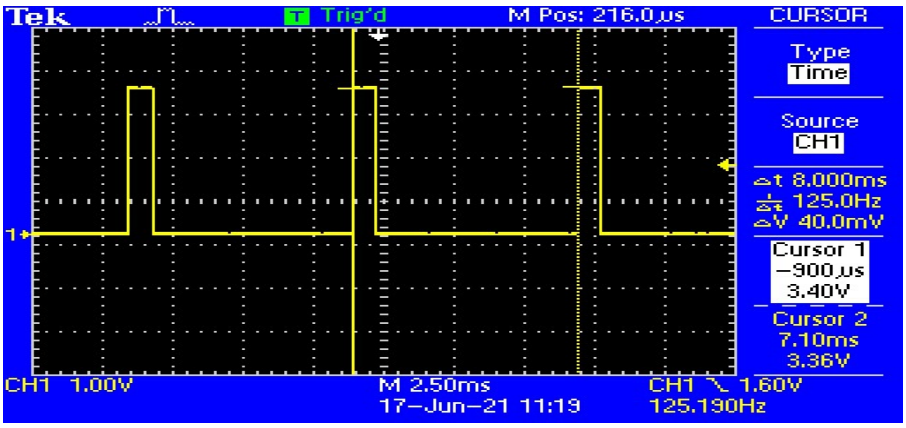


Figure 19: LED Matrix Timing

5.6 ADC Slider

The ADC Slider is required to output a voltage from 0 to 3.3V depending on the resistance of the potentiometer, which is determined by the position of the slider. In order to test if the ADC Slider was working correctly an oscilloscope was connected to pin 2 and 3 (see figure 7) and the output voltage was recorded as the slider was moved from the buttom to the top position. As seen in figure 20 the voltage on pin 2 varied from 0 to 3.32V, thus verifying that the ADC Slider is working as intended.

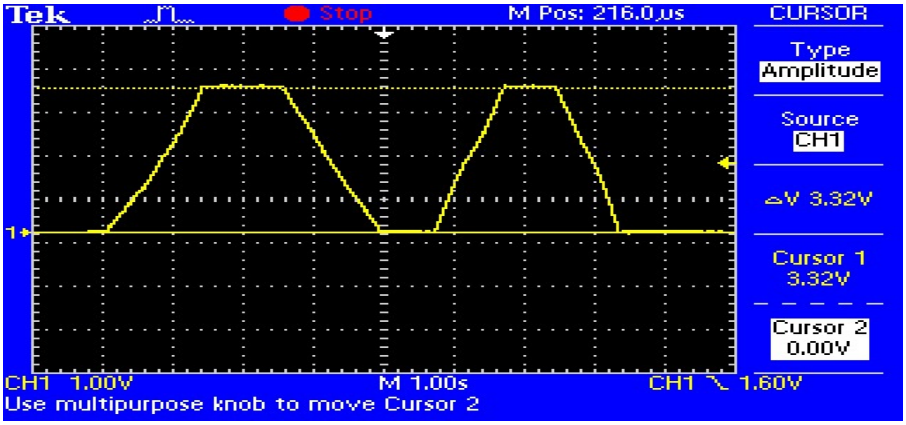


Figure 20: ADC Slider Output Voltage

5.7 IMU

The IMU is required to output acceleration values and upon going beyond a 30 degree angle a button press should be registered in that direction. Although the IMU was implemented in hardware, the student was unable to correctly implement the IMU in software and certain bugs were occurring, thus a choice was made to remove the code and leave it out of the project. Thus, the design cannot achieve the required results for the IMU as it was not working as intended.

6 Conclusions

As seen in this report, this system design and hardware implementation was sufficient in order to achieve the main goal of designing a system that acts as an arcade-style game console (see figure 22 for a photo of the fully built physical project). All of the requirements and specifications, except for the IMU implementation were met, as detailed in section 5. Some notable requirements that were met include the power supply voltage regulator outputs that were within 5% tolerances, the timing/updating of the LED Matrix was correctly shown to update once every 8ms and the total current draw of all the pins in the LED Matrix were below the maximum total current draw specification listed in the STM32F103RB MCU datasheet. Overall the project was a success and the student greatly enjoyed building and implementing each element, then coding the software and seeing the whole system come together.

6.1 Non-compliance's

This system did not meet the requirement to have an IMU implemented to act as an input device with which the user could play the games with. Although the IMU was implented in hardware, when attempting the software implementation certain bugs were occuring that the student was unable to fix, hence a choice was made to take out the IMU functionality in order for the system to still function as intended.

6.2 Design Shortcomings

Overall the wires could have been routed a bit more neatly for a cleaner look. The IMU wires could also be shortened slightly to prevent them from catching/hooking on any surface and breaking wires off.

6.3 Recommendations and Possible Future Improvements

One improvement that could be added is to correctly implement the IMU in order to completely meet the requirements and to add another input method with which the user can control the games.
A second improvement would be to use a higher resolution LED matrix such that more complicated and more detailed games can be implemented in the future.
Another possible feature would be to build in an internal rechargeable 9V lithium-ion battery so that the project can be powered when access to a external power supply is not possible.

References

[1] *Project Definition Document*, PDD v0.6, Dr. Arno Barnard, Dr. Callen Fisher, May 2021.

[2] *μA7800 SERIES POSITIVE-VOLTAGE REGULATORS*, 7805 datasheet, Texas Instruments, May 2003.

[3] *MCP1700 Low Quiescent Current LDO*, mcp1700 datasheet, Microchip Technology, 2013.

[4] E. I. to Microprocessor Systems, "Uart basics," https://ece353.engr.wisc.edu/serial-interfaces/uart-basics/, Oct. 2019.

[5] Sparkfun, "Pull-up resistors," https://learn.sparkfun.com/tutorials/pull-up-resistors/all, Feb. 2013.

[6] *STM32F103xB Medium-density performance line ARM®-based 32-bit MCU*, STM32F103RB datasheet, STMicroelectronics, 2015.

[7] *PTA Series - Low Profile Slide Potentiometer*, ADC Slider datasheet, Bourns, 2013.

[8] *MEMS digital output motion sensor ultra-low-power high-performance 3-axis nano accelerometer*, LIS3DH datasheet, STMicroelectronics, 2016.

[9] *I2C Bus Pullup Resistor Calculation*, Application Report, Texas Instruments, 2015.

[10] *MEMS digital output motion sensor ultra-low-power high-performance 3-axis nano accelerometer*, Application note LIS3DH:, STMicroelectronics, 2011.

7 Appendices

7.1 Appendix A: Complete Circuit Diagram

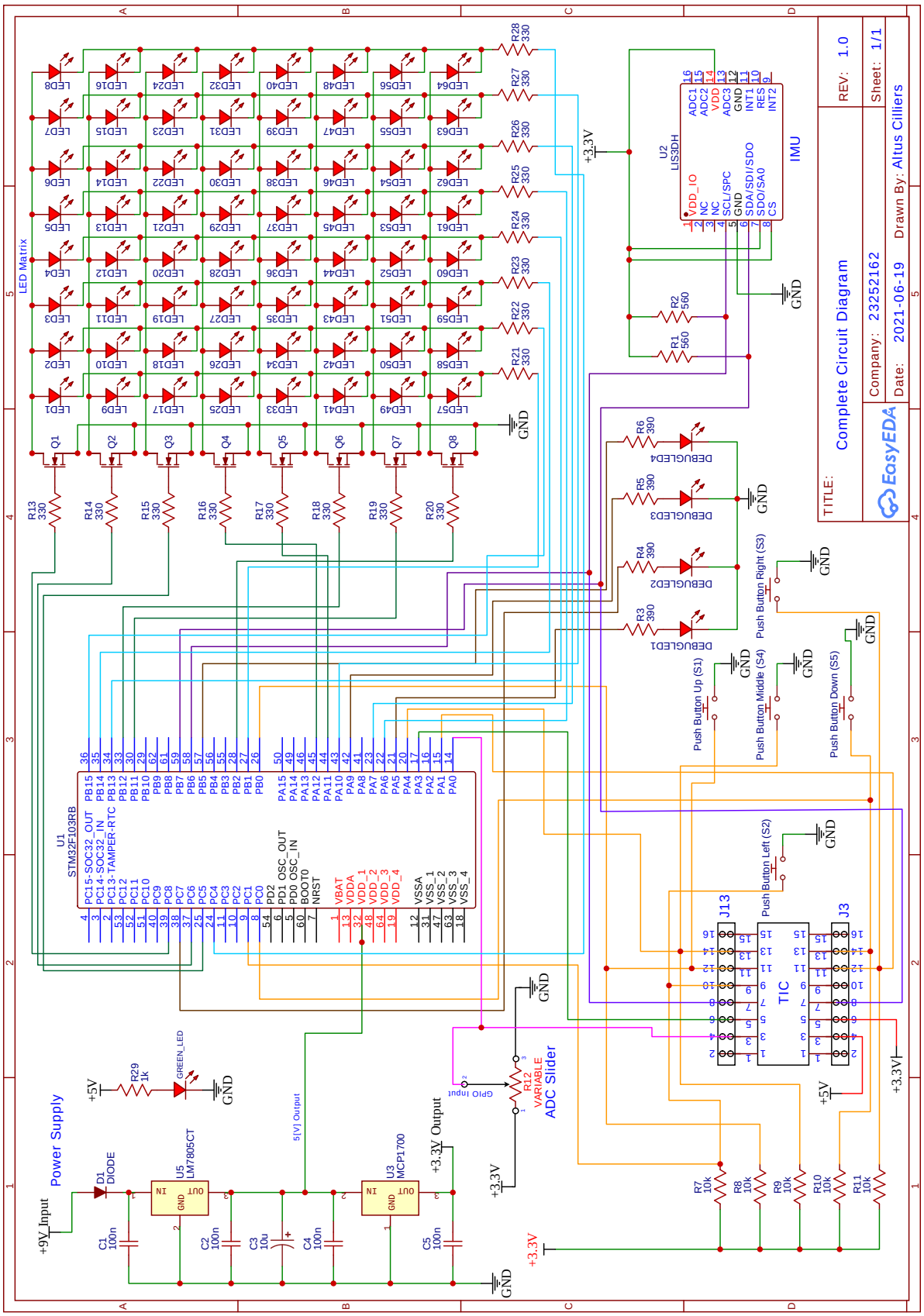


Figure 21: Complete Circuit Diagram

7.2 Appendix B: STM32F103RB Module Pinout and Configuration

MCU Pin	Hardware Element	Configuration
PB0	Up Button	GPIO Input
PC1	Left Button	GPIO Input
PA4	Middle Button	GPIO Input
PA1	Right Button	GPIO Input
PC0	Down Button	GPIO Input
PA0	ADC Slider	GPIO Input
PA2	UART TX	UART2 TXD
PA3	UART RX	UART2 RXD
PB6	I2C SCL	I2C2_SCL
PB7	I2C SDA	I2C2_SDA
PA5	Debug LED 1	GPIO Output
PC7	Debug LED 2	GPIO Output
PA9	Debug LED 3	GPIO Output
PB5	Debug LED 4	GPIO Output
PB1	LED Matrix Column 0	GPIO Output
PB15	LED Matrix Column 1	GPIO Output
PB14	LED Matrix Column 2	GPIO Output
PB13	LED Matrix Column 3	GPIO Output
PA6	LED Matrix Column 4	GPIO Output
PA7	LED Matrix Column 5	GPIO Output
PA10	LED Matrix Column 6	GPIO Output
PC4	LED Matrix Column 7	GPIO Output
PC8	LED Matrix Row 0	GPIO Output
PC6	LED Matrix Row 1	GPIO Output
PC5	LED Matrix Row 2	GPIO Output
PA12	LED Matrix Row 3	GPIO Output
PA11	LED Matrix Row 4	GPIO Output
PB12	LED Matrix Row 5	GPIO Output
PB11	LED Matrix Row 6	GPIO Output
PB2	LED Matrix Row 7	GPIO Output

Table 4: STM32F103RB Module Pinout and Configuration

7.3 Appendix C: Photo of Fully Built Physical Project

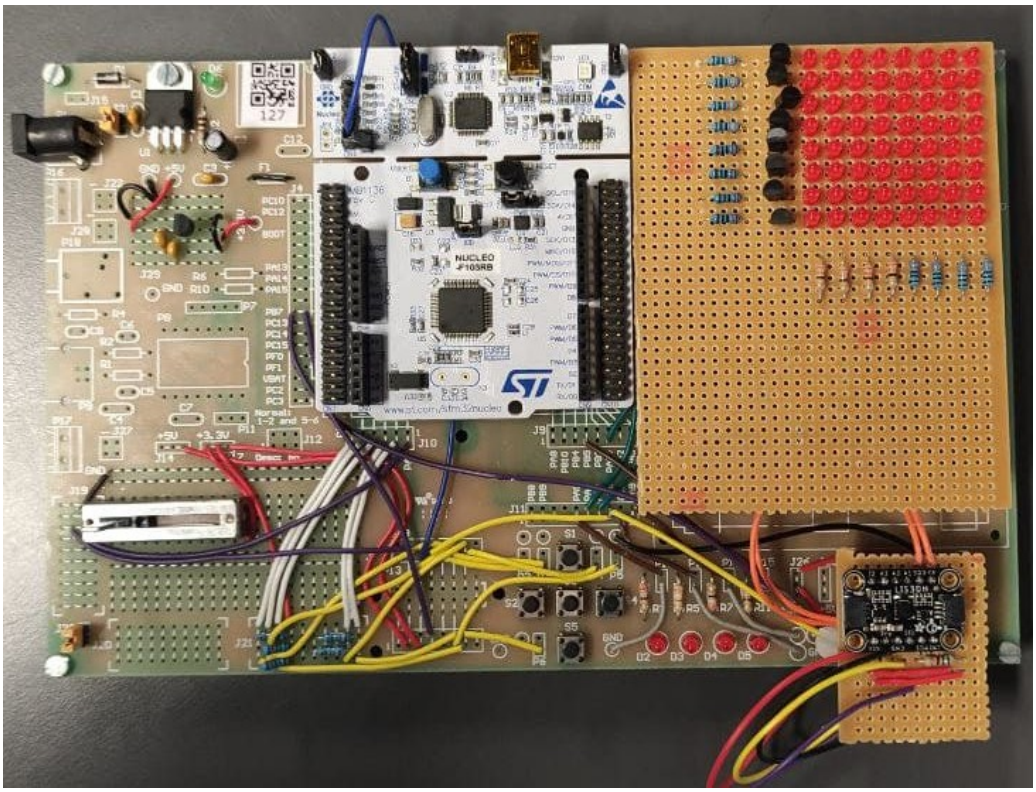


Figure 22: Fully Built Physical Project