ELECTRICAL & ELECTRONIC ENGINEERING
STELLENBOSCH UNIVERSITY

DESIGN (E) 314
TECHNICAL REPORT

Dot-Matrix Arcade Game Console

Author:
Stéphan van Biljon

Student Number:
22534865

June 6, 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*	22534865 Studentenommer / *Student number*
S. van Biljon Voorletters en van / *Initials and surname*	June 6, 2021 Datum / *Date*

Contents

List of Figures

List of Tables

List of Abbreviations

LED Light Emitting Diode

TIC Test Interface Controller

RX Receive to

TX Transmit from

SDA Serial Data

SCL Serial Clock Line

UART Universal asynchronous receiver-transmitter

ADC Analogue to Digital Conversion

I2C Inter-Integrated Circuit

GND – Ground

IMU – Inertial Measurement Unit

TS – Test Station

SB – Student Board

LCD – Liquid Crystal Display

GPIO – General Purpose Input Output

MOSFET – Metal Oxide Semiconductor Field Effect Transistor

CS – Chip Select

FSM – Finite State Machine

HAL – Hardware Abstraction Layer

MSB – Most Significant Bit

LSB – Least Significant Bit

FS – Full Scale Selection

LHS – Left Hand Side

RHS – Right Hand Side

List of Symbols

T_{AMB} – The ambient temperature of the environment

T_{jmax} – The absolute maximum allowable junction temperature

R_{thJA} – The thermal resistance between the junction and the environment

Vf – Forward voltage: voltage needed over LED terminals to turn it on

Q – Quantization Error

SD0 - I2C less significant bit of the device address

1 Introduction

The dot matrix arcade console is designed to fulfill various requirements. These requirements document the aspects of the console that need to be implemented with the hardware and software components that the rest of the report will detail. A high-level description of the system is intertwined with the requirements in order to give a brief but comprehensive overview of the project.

The game console needs to generate regulated 5V and 3.3V supply voltage from a 9V voltage source. The 5V supply is used to supply voltage to the microcontroller and the 3.3V supply is used to provide supply voltage to the peripherals of the console (The slider and the IMU). The microcontroller needs to sample these peripherals in order to provide input to the program. The rate at which the IMU, slider and buttons are sampled is dependent on the game being played.

The system needs to communicate data such as the position of the bat and ball as well as their characteristics over a UART connection for debugging and program verification purposes. A custom LED display needs to be built. This LED dot matrix is the only visual feedback for the user interacting with the system. It needs to consist of an 8 x 8 (60mm x 60mm) grid which is made up of 64 LEDs. This display needs to verify the working condition of LEDs through a calibration pattern at the start of code execution. Once a game has ended, the four corners of the LED matrix need to light up and wait for a new game to be selected.

Buttons are an integral part of the inputs that the system must sample. They are used to select a game (single player tennis or maze) via the middle or left button as well as controlling the movement of the ball/bat in the various games.

The games that the game console can play includes the maze and single player tennis games. For the maze, one of 4 possible mazes can be chosen. The buttons and the IMU are used to move the ball through the maze where the ball can be blocked by obstacles. The game ends once the ball reaches the goal position or the middle button is pressed. For tennis, a vertical bat is moved vertically and horizontally in order to bounce a ball off of the bat. The slider, buttons and IMU are all used for bat input at different stages of the project. The ball bounces off the bat and the top, bottom, and right-hand boundaries of the screen. The game ends when the ball hits the boundary of the left-hand side of the screen, or the middle button is pressed.

The system is essentially designed to provide its own power supply to power the sensors for the game. These sensors are then used to interact with the game in its different states as it is displayed on the dot matrix. Different games are played, and the user can interact with and complete or abandon the games at will.

2 System description

The complete system block diagram is shown below in figure 1.

The power supply supplies 9V to the 5V power regulation either through a bench power supply, or with a nominal 9V battery. The 5V power regulation steps down the 9V and provides regulated power to the STM32F103 microcontroller as well as providing 5V to the 3.3V power regulation. The 3.3V power regulation further steps down the 5V supplied to it and produces a 3.3V output which is used to power the peripherals to the STM32F103 (such as the potentiometer and accelerometer).

The potentiometer receives a supply voltage of 3.3V and outputs a range of 0-3.3V analogue voltage which is dependent on the position of a sliding contact on a resistive strip. The output can be sampled by the TIC or the STM32F103 as an input for movement for the arcade game. The accelerometer is another component that receives a supply voltage of 3.3V from the power regulation. The accelerometer will output the 3-axis acceleration of the device on the data line (SDA) which will again be sampled by the TIC and the STM32F103 as input for movement for the arcade game.

The LED Matrix is the display for the arcade game. The STM32F103 will output current to the dot matrix which will turn LEDs on and off in accordance with the control logic of the microcontroller. The debug LEDs are a separate display that receives current output from the microcontroller which is then used to turn LEDs on and off for debugging or maze selection purposes.

The buttons do not require a power supply. They act as a switch which can control the voltage on the STM32F103 pins. The microcontroller therefore samples a button push as an active low input which is useful for changing the state of the arcade game. The buttons are also sampled by the TIC. The TIC connector is an intermediate connection between the test-station and the development board. It samples inputs from the potentiometer, accelerometer, buttons, 3.3V regulation, 5.5V regulation and the UART lines of the microcontroller. It also provides an interface for the test-station to act as a power supply to the board during testing.

The NUCLEO STM32F103 microcontroller is the processing unit of the game console which receives 5V from the power regulation and samples the input from the potentiometer, the accelerometer, and the buttons. It also outputs UART data on a TX/RX line to the TIC. The microcontroller processes the analogue voltage from the potentiometer through ADC which allows the digital representation of the voltage to be stored on the microcontroller. The microcontroller also communicates with the accelerometer through I2C protocol, within which, the STM32F103 acts as the master which can read data from or write data to the slave (accelerometer).

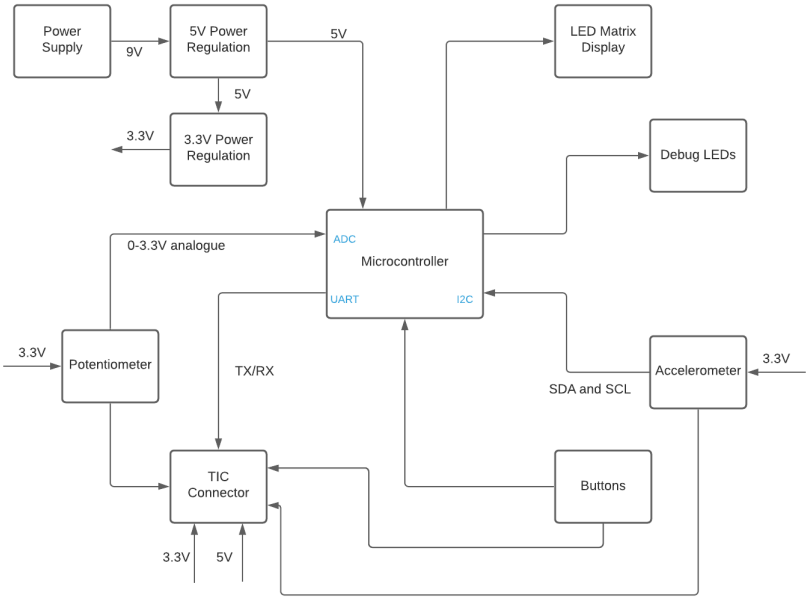

Figure 1: Complete System Block Diagram

3 Hardware design and implementation

3.1 Power supply

5V power supply circuit

The 5V power supply circuit has an input voltage supply of 9V from a nominal battery or a bench power supply. The supply circuit uses a standard 7805 regulator (U1) in a TO220 package to regulate the input down to 5 V [1]. This regulated voltage output will be used as an input voltage to the STM32F103 microcontroller as well as the 3.3V power supply circuit. If the MCU used USB5V instead of an external 5V supply, the USB port could be damaged. The diode (D1) acts as a protection against reverse polarity current produced as a result of capacitors charging and discharging. The capacitors (C1, C2, C3) ensure the regulator stability and are used to filter out incoming noise. The resistor R12 is used to limit the current into the LED (D6) which acts as a visual confirmation of voltage on the 5V line.

3.3V power supply circuit

The 3.3V power supply circuit has an input voltage supply of 5V from the 5V power supply circuit. The supply circuit uses a MCP1700 regulator, provided in a TO-92 package [1]. This regulator regulates the input down to 3.3V. This regulated voltage output will be used as a power supply for various peripherals on the board such as the slider and the accelerometer which need a 3.3V input. The two capacitors, Cin and Cout act as bypass capacitors that help to filter out noise and ensure the stability of the regulator.

Calculations

The minimum supply voltage input for the overall power supply circuit is 7V-35V [2 pg. 4] for the 7805 regulator. A 9V battery or a bench power supply will thus be sufficient without going over or under the rated voltage. The thermal resistance junction-ambient (R_{thJA}) of the 7805 equals 50 °C/W for a TO-220 package [2 p. 4]. Assuming T_{AMB} = 25 °C and T_{jmax} = 150 °C, the maximum power that can be dissipated:

$$\begin{aligned} \text{PDMAX} &= (\text{TJMAX} - \text{TAMB}) * \text{RTHJA} \\ &= (150\ °\text{C} - 25\ °\text{C}) *\ 50\ °\text{C/W} \\ &= 2.5W \end{aligned}$$

In order to calculate the actual power dissipation, we need to find the current that the regulator provides to the board. The maximum current drawn by the MCU = 150mA [3 pg. 37] and the current drawn by the dot matrix is ~12mA (1.5mA per row (explained in dot Matrix section) for 8 rows). The other components draw a negligible amount of current. Therefore, the power dissipated by the 7805 regulator:

$$\begin{aligned} \text{PD} &= (\text{Vin} - \text{Vout}) * \text{Isourced} \\ &= (9V - 5V) * 162mA \\ &= 0.648\ \text{W} \end{aligned}$$

This power dissipation is only ~25% of the maximum power dissipation, therefore no thermal heat sinking hardware will be required.

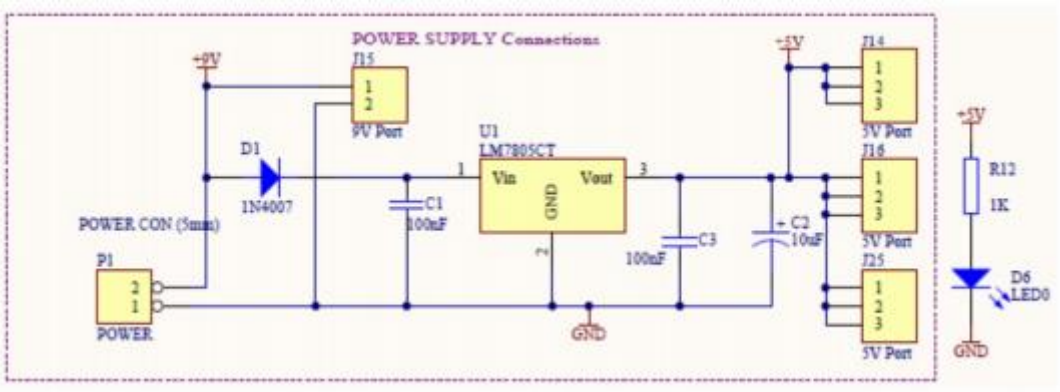

Figure 2: 5V Power Supply Subcircuit [1 pg. 7]

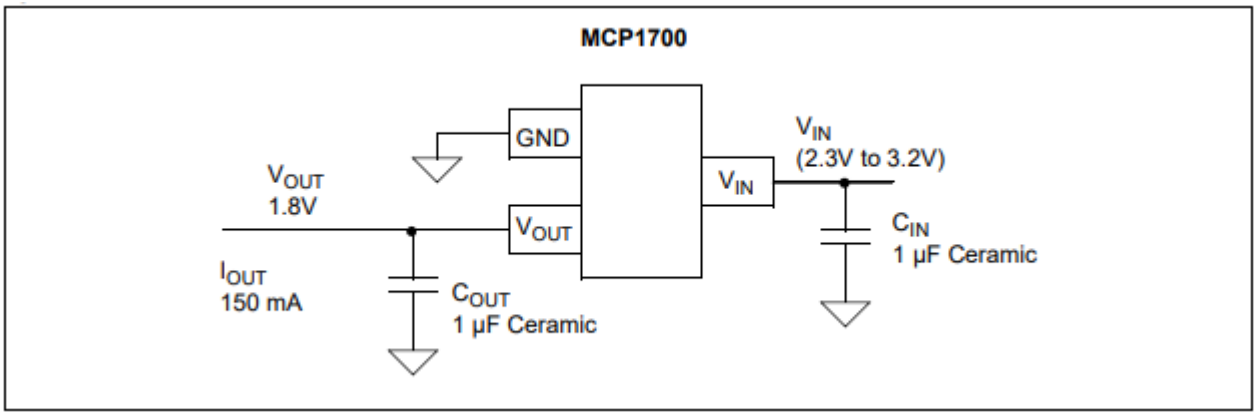

Figure 3: 3.3V Power Supply Subcircuit [4 pg. 2]

3.2 UART communications

A UART connection is used, in transmit only mode for the MCU, that operates in an 8N1 configuration (8 data bits, no parity bits and a single stop bit), at a data rate of 115200 baud [1 pg. 2]. The UART is implemented in software, but it is transferred through hardware. The NUCLEO F103RB is split into a top half and a bottom half. The top half is responsible for configuring the ARM STMF103 microchip on the bottom half. The bottom half transfers UART information (TX) to the top half (RX pin), which then acts as a common node with the TIC through a female wire (Thereby allowing the test station to sample the MCU UART.).

The interaction between the transmitter and the receiver happens through memory mapped registers. The UART channel is also connected to the ST-Link chip on the NUCLEO which will allow a virtual COM Port to be made when the board is connected to a PC through a USB cable [1 pg. 16].

UART has been chosen as a communication protocol due to its suitability for the requirements for this project. The hardware and software complexity of UART is quite simple, therefore being easy to setup, program, and debug. The primary communication protocol should not be over-complicated for its functions. UART is also ideal for short distance, serial communication between two devices (This is the situation between the STM32F103 and the test station). No clock signal is needed for the UART and only one wire is needed to communicate, thereby reducing the expense of the hardware. This is ideal for an amateur project.

The following protocol will be used for the UART:

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'

Figure 4: UART Protocols [1 pg. 22]

For the above protocols, the tennis message is sent every 100ms while the Maze message is sent every 300ms. The StdNum and Calibration messages are sent once at the beginning of software execution.

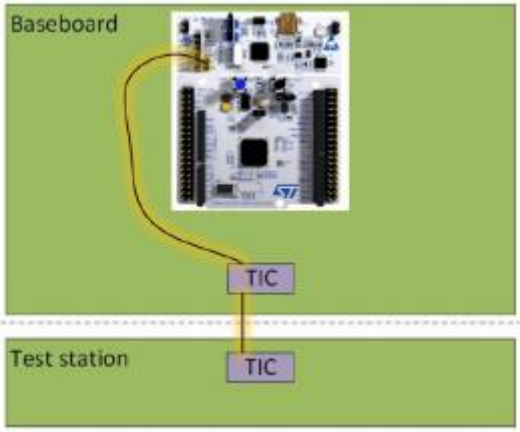

Figure 5: UART Hardware Connection with TS [1 pg. 17]

3.3 Buttons

The buttons used on the student board connect between a MCU pin and ground. 5 buttons have been implemented (S1, S2, S3, S4, S5). They are specifically used as switches in order to provide input to the MCU for the game console (Up, Left, Middle, Right, Down). The buttons are configured to be active-low. This means that when a button is pressed, the MCU pin is connected to GND and will therefore have a low value. If the button is not pushed, the voltage on the MCU pin (3.3V) is not pulled down to ground. The MCU will therefore be able to recognize a button input and adjust for movement within the games accordingly.

The buttons are required to have some sort of a resistance in order to preserve the safety of the regulator. For this design, the internal pull up resistors of the MCU were used instead of series resistors in hardware. This decision was made in order to save both time and money. The internal resistors are also independent of any hardware which can be carelessly implemented and fail. These internal resistors have a typical value of 40kΩ for STM32 devices [5 pg. 115]. This is a large enough resistance to properly protect the regulator.

Buttons are chosen because switches are some of the simplest input mechanisms to implement in hardware and software. Buttons are low complexity hardware, which means they are cheap and widely available. Button debouncing will need to be performed to stop multiple inputs from one button press, however, this will be done in software to save board space and costs.

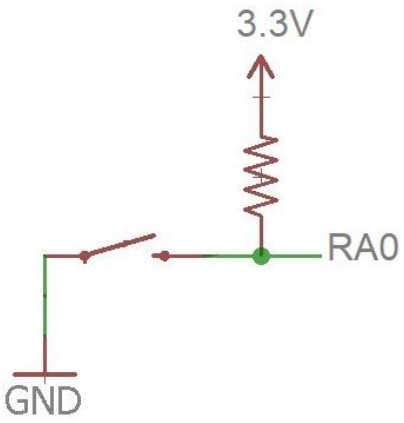

Figure 6: Active Low Configuration [6]

3.4 LEDs (Debug)

The debug LEDs (D2, D3, D4, D5) are used as a debugging interface for timing as well as for displaying the numerical selection of mazes. The LEDs are connected through a 1kΩ resistor to GPIO output pins on the MCU. This ensures that the current through the LED is limited such that the LED has an appropriate brightness and voltage over its terminals (more than the forward voltage – further discussed in the Dot Matrix section). When a maze is selected, the appropriate LED will light up to show the choice of maze (D2 associated with Maze 1, D3 associated with Maze 2, etc.).

LEDs have been chosen as a debugging tool due to the fact that they are separate from the dot matrix and can light up to provide timing information independent of the games being played. LEDs also have a simple implementation for a low cost.

3.5 Dot Matrix

The dot matrix is designed to be a display for feedback to the user, using images of mazes, balls, and bats throughout the operation of the game console. It consists of an 8x8 matrix of LEDs where the positive of each LED is connected in series with a current limiting resistor to a GPIO pin of the MCU while the negative of each LED is connected in series with a MOSFET to a GPIO pin. 16 GPIO pins have been used to control the state of the LEDs. There are 8 rows and 8 columns consisting of 64 LEDs in total. Each of the 16 GPIO pins are connected to a row or a column. Each GPIO pin is therefore responsible for 8 LEDs (pins source/sink current to/from the LEDs). The dot matrix has been used due to the simplicity of turning LEDs on and off as a display. The dot matrix is also significantly less expensive than a manufactured LCD.

Calculations

In order to ensure the appropriate functioning of the dot matrix, some calculations need to be performed. Sufficient current needs to be sourced to the LEDs to ensure that they stay on, regardless of the state of any other LED. This current must not damage the MCU or compromise the TTL/CMOS compliance of the MCU.

The maximum current that each GPIO pin can source/sink is 25mA [3 pg. 37]. There is no provided data sheet for the LEDs and so a reasonable assumption can be made that the forward voltage of the LEDs (Vf) = 1.7V. We therefore need enough current to provide Vf for the LED without sourcing too much current from a GPIO pin. The maximum current that a pin can source without compromising TTL/CMOS compliance is 8mA [3 pg. 65]. If this current is increased, the forward voltage of the LEDs may not be met. 8mA is below the maximum output of 25mA, however, to be safe, the current through

each LED is to be limited as much as possible in order to protect the GPIO pins while maintaining appropriate brightness.

We therefore need to choose a resistor value that will limit the current to ~1.5mA (~20% of the maximum current):

$$R = \frac{Vo - Vf}{ILED}$$

$$= \frac{3.3V - 1.7V}{1.5mA}$$

$$= 1066.67\ \Omega$$

An approximate value of 1kΩ has been chosen for the series resistor in order to limit the current to an appropriate level. Since each LED will source 1.5mA of current, a GPIO pin will need to provide a maximum of 12mA of current to a row. To avoid this situation, only one column of LEDs is turned on every millisecond at a time such that each column can be commanded every 8ms. This results in a duty cycle of 1/8 or 12.5% for a column of LEDs. This solution is implemented in software.

Finally, the current that is sunk by each GPIO pin need to be taken into account. When one LED in a column is turned on, a GPIO pin will sink 1.5mA of current. If all 8 LEDs are turned on, then the GPIO pin will need to sink 12mA of current. This is above the maximum operating current to preserve TTL/CMOS compliance. 2N7000 MOSFETs are therefore added to the negative terminals of an LED column. The MOSFETs can sink up to 200mA [7 pg. 1] and allows the GPIO pin to be connected to the circuit without being damaged.

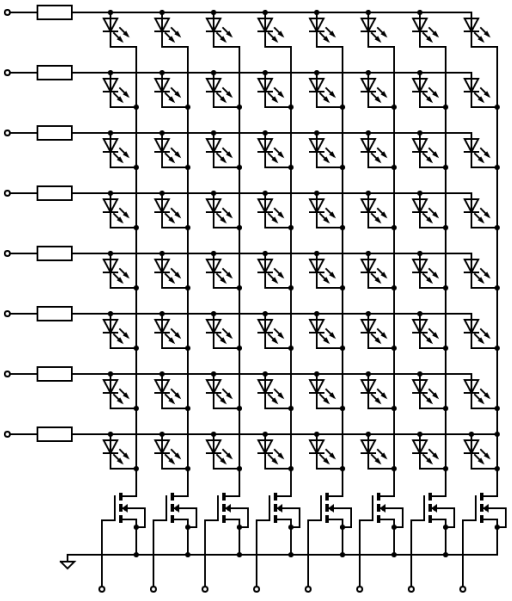

Figure 7: Complete Circuit Diagram of LED Dot Matrix

3.6 Slider

The slider is used to output a variable, analogue voltage of 0-3.3V to the MCU GPIO pins as well as the TS. The slider is connected to 3.3V and GND. A sliding contact on a resistive strip within the slider determines the output voltage of the slider. The sampled, analogue output voltage sent to the GPIO pin of the MCU is then stored as a 12-bit number within the STM32F103.

A slider has been chosen as an alternative input method to the buttons. One slider can effectively act as two buttons, making the slider input method both less hardware intensive and faster to sample.

3.7 IMU

A 3-axis LIS3DH MEMS accelerometer has been used as an IMU. This IMU is a device which can measure acceleration in 3-dimensional space with an X, Y and Z axis. This acceleration data is transferred to the MCU through I2C protocol. The IMU acts as the slave device while the MCU acts as the master. The LIS3DH has a power supply of 3.3V that is provided by the regulator circuit. It also consists of an SDA pin which transfers data between the slave and master, an SCL pin which ensures the correct timing of the data transfer, a CS pin which decides the protocol used for data transfer (CS = 1 for I2C) and a SD0 pin which determines the slave address of the IMU. The acceleration data sampled by the slave, is sent to the master which is then stored in MCU registers where the data can be processed.

The IMU has been used, due to the speed of input (the IMU can replace all 4 movement input buttons) as well as the intuitive feel, low power consumption and customization that exists within the device.

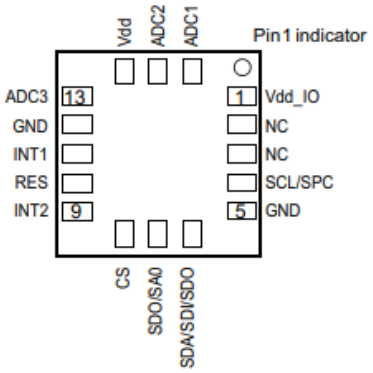

Figure 8: LIS3DH Pin Layout
[8 pg. 8]

4 Software design and implementation

4.1 High-Level description of the program

The software of the STM32F103 is used to control the state of the game console. The software can be described as a FSM within which the console may be in different states and depending on the input received, transitions or output might be produced.

When the program is executed, it will continuously check for button presses (or other input) in the game loop which can change game states as well calling functions from the game loop or interrupts to update the dot matrix to display the game states as visual feedback to the user. The functions used to update the dot matrix change the state of GPIO output pins using HAL functions. GPIO input pins are sampled, and the data is processed in order to decide if a state change or an output is necessary.

Due to the fact that a game should be able to be played for a theoretically infinite amount of time, a game loop has been implemented such that the program will have an initial state when execution of the main function begins but there is no final state that the program will repeatedly reach.

4.2 Data Flow and Processing with Control Logic

At the start of the program execution, the calibration sequence for the dot matrix is performed. This leads to the game console being in one of 3 game states which are implemented with a game loop. Each game (Maze and Tennis) can be escaped, back to game mode 0, if the middle button is pressed or if the exit condition involving the respective balls are met. The data flow and processing of the IMU is discussed in section 4.4.

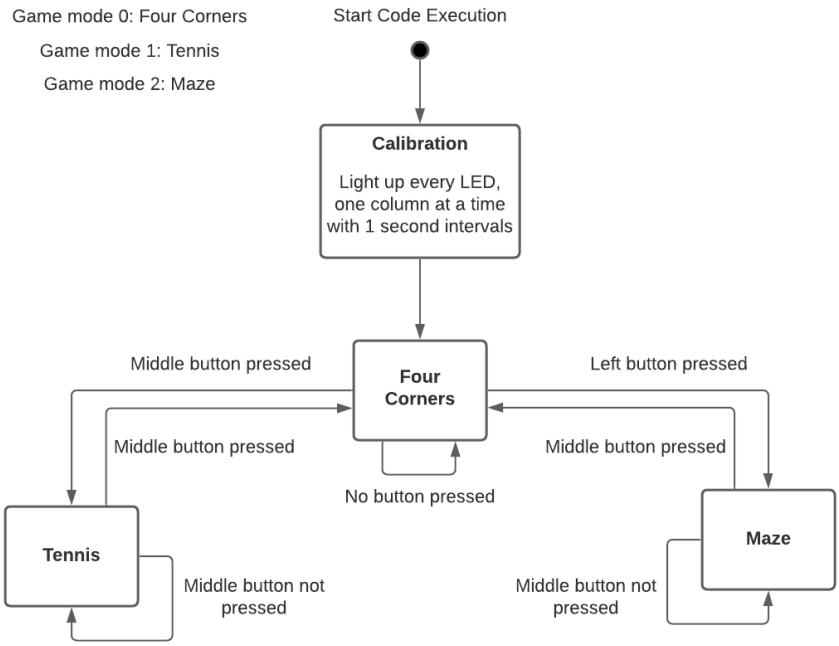

Figure 9: Overview of Program Execution

The screen updating function is displayed in the figure below. It is called from the SysTick timer every 1ms and is used to physically set the LED states via HAL functions as well as calling the maze and tennis functions to update the LEDs that need to be controlled (The digital LED states are stored in a 2D, 8x8 array). All the data flow in the “game mode 1” branch is further expanded on in the “Tennis” section below while all the data flow in “game mode 2”, after a maze has been chosen, is expanded on in the “Maze” section below.

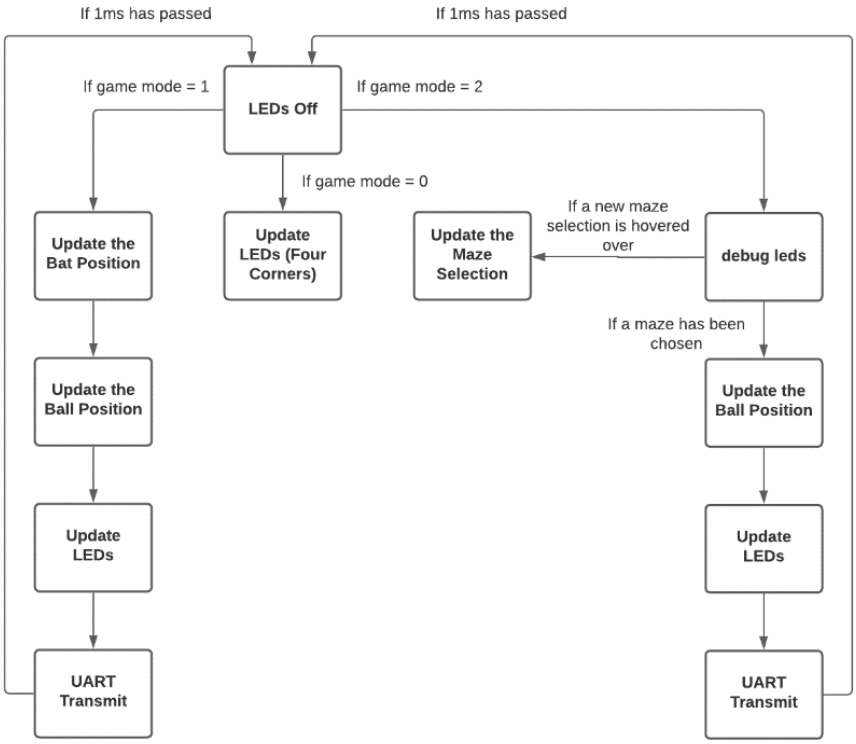

Figure 10: Screen Updating Flowchart

Maze

When the maze is first entered, the LED that was on for the previous ball position is turned off. If a movement input is detected, then the position of the ball in the maze will change. This ball can only move every 300ms and so if a movement input is detected but the ball cannot move yet, the updated ball position is stored and will be assigned to the ball once it is allowed to move. If no movement is detected, then the ball and the goal will flash (by changing the LED states) once every 300ms and 100ms, respectively. The obstacles for the maze are set within the updating screen function.

Next, the counters that keep track of ball movements as well as ball and goal flashing are incremented. These counters are reset every time a ball is allowed to move. The function to do this is called from the update screen function which is called every 1ms. Finally, the UART will be transmitted every 300ms, and the transmission is also controlled through counters that are incremented and reset in the same way as the previous ones discussed. This process repeats every 1ms when the update screen function is called from the SysTick timer.

The two ways to exit the maze game are either by pressing the middle button or when the ball reaches the same position as the goal (this position is dependent on which of the 4 mazes are being played). This termination of the maze game can occur at any point in the cycle and so has not been included in the flowchart.

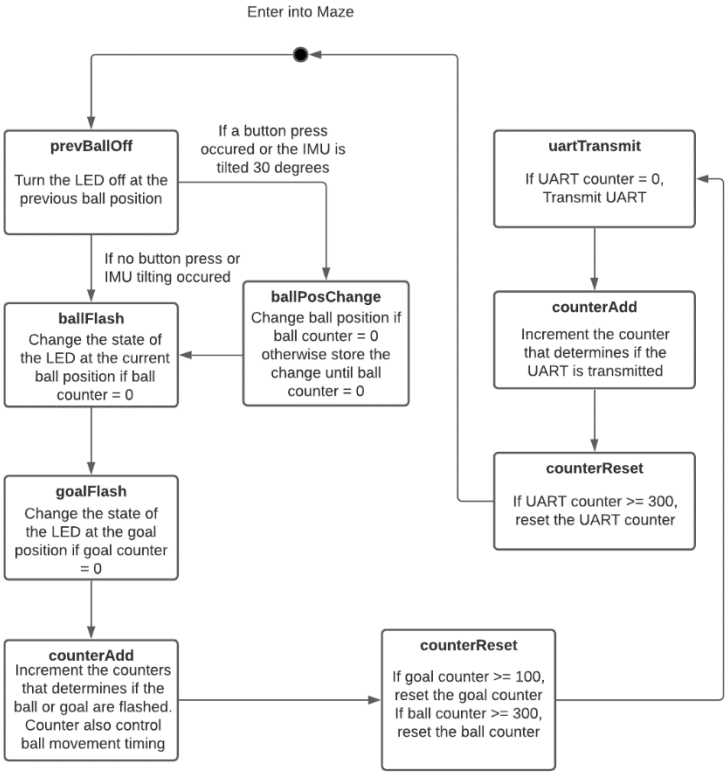

Figure 11: Maze Game State Flowchart

Tennis

When Tennis is first entered, the two LEDs that were on for the previous bat position are turned off. If a movement input is detected, then the position of the bat will change. The bat can only move every 100ms and so if a movement input is detected but the bat cannot move yet, the updated bat position is stored and will be assigned to the bat once it is allowed to move. This process of movement input for the ball is repeated (if movement input is detected), with the extra component that if the ball hits the bat for the third time, the velocity of the ball is increased.

Otherwise, the counters that are responsible for the ball and bat timing are incremented and reset if the appropriate conditions are met. Finally, the UART will be transmitted every 100ms. The transmission is also controlled through counters that are incremented and reset in the same way as the previous ones discussed. This process repeats every 1ms when the update screen function is called from the SysTick timer.

The slider was initially used for the up and down movement of the bat for Demo 3. This input was replaced by the movement of the bat via buttons and the IMU for Demo 4. The two ways to exit the tennis game are either by pressing the middle button or by letting the ball reach an X position = 0. This termination of the tennis game can occur at any point in the cycle and so has not been included in the flowchart.

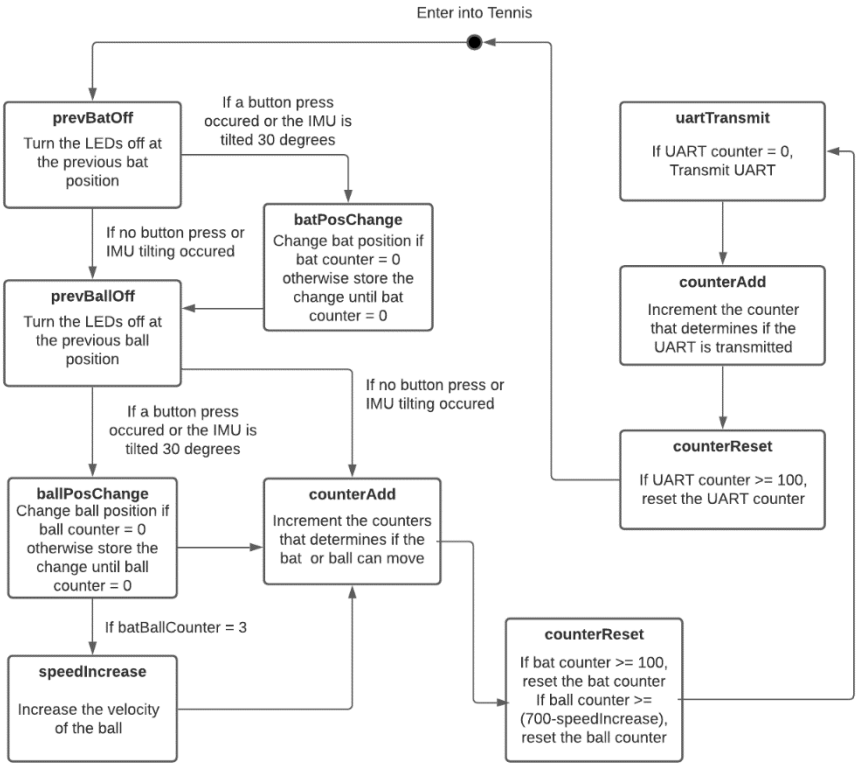

Figure 12: Tennis Game State Flowchart

4.3 Button Debouncing

Due to the mechanical nature of buttons causing them to bounce when pressed and providing incorrect inputs, the buttons have been debounced using software. As can be seen in the "Measurements and Results" section, the buttons seem to have an input bounce for about 2ms. This means that we must wait at least that amount of time when a button is pressed before sampling what the input is.

In software, the state of the MCU GPIO input pin connected to the button is continuously checked. If this input changes from high to low for the first time (indicating a button press), then a flag is set in the SysTick timer which causes a counter to be incremented every 1ms. To be on the safer side in the case of any extreme bouncing, the bounce counter is incremented 30 times (thereby causing the program to wait 30ms before sampling the input at the GPIO pin. It is extremely unlikely (impossible for a human) that the button will be pressed more than once every 30ms (33 times per second).

When the GPIO input is only sampled after an appropriate delay, this allows the correct input to always be sampled and for the button bounce to be eliminated.

4.4 IMU Interface

The MCU interfaces with the IMU through the I2C protocol. To first set up the communication line between the IMU and the MCU, a few things need to be done. Once the IMU is powered up, it calibrates itself and then enters power down mode. You need to write into the CTRL_REG1 address of the IMU to set it to a power on mode [9 pg. 12].

This is done in the program by initiating a master transmit to the IMU (slave) with the address of the slave so that the IMU knows you want to communicate with it. Once the acknowledgement of successful communication is received from the slave (HAL_OK), the address of the CTRL_REG1 is sent to the IMU

as well as the value that needs to be written to the control register. This value sets the data rate of the IMU to 100Hz, in normal power mode, as well as enabling the X, Y and Z axes for data sampling.

Now that the IMU is setup with a specific data rate, in an appropriate power mode, and with all the axes enabled, data may be read by the MCU. The MCU initiates a master write, within which it sends the address of the data registers that need to be read from, to the slave. According to the LIS3DH application note, the full acceleration of an axis is stored within two registers that are concatenated for the full acceleration in a 16-bit two's complement form [9 pg. 14]. The addresses of these two registers for the X, Y and Z axes are all sequential [8 pg. 31] and so the address of the first part of the X acceleration data is sent as the address of the data register that needs to be read. This address is also sent in such a way that its MSB is set to 1 to auto-increment the read address after every master read.

Next, a master read is initiated and the slave is asked for 6 bytes of data: (X_H, X_L, Y_H, Y_L, Z_H, Z_L). These 6 bytes are then extracted from the IMU and sent via the SDA, concatenated, and stored in variables for data processing. This process of communicating with the slave with regards to the register you want to read from and what data you want to read, is repeated every time the game loop is run. This results in a movement input every time the acceleration data of the IMU changes enough to indicate a 30-degree tilt in the X, Y or Z axes.

4.5 Peripheral Setup

SysTick Timer

The SysTick timer is setup as a standard, 24-bit down counter with auto reload capability, a maskable system interrupt generation when the counter reaches 0 and a programmable clock source [3 pg. 19]. It is used as a 1ms timer that triggers an interrupt which allows flags to be set or counters to be incremented within the SysTick handler. This timer was chosen to provide a convenient way for the program to keep track of ball or bat movement timings as well as UART transmission timings.

ADC Channel 1

ADC was used in order to be able to sample the varying voltage output of the slider. The sampled analogue voltage value is stored in a digital form. The ADC was setup so that the voltage is stored in 12 bits (12-bit ADC resolution). This allows the voltage to be represented by 4096 points. The lower and upper reference voltage for the ADC is 0V and 3.3V, which results in a quantization error of:

$$Q = \frac{VREFP - VREFN}{2^N - 1}$$

$$= \frac{3.3V - 0V}{4095}$$

$$= 0.81 \frac{mV}{LSB}$$

These sampled points of the slider output voltage can then be used in data processing to determine movement input for the bat.

UART Channel 2

UART was implemented as a convenient way to transmit serial data between a master and a slave over short distances. It is a simple communication protocol to implement and is set up to have a baud rate of 115200 bps, with 8 data bits, 1 stop bit and no parity checking, using TTL signal levels [1, p. 21].

The UART channel is setup to be asynchronous since no clock signal is involved in the serial data transfer. Global interrupts for UART have been implemented so that UART can be using for debugging purposes. For this program, the MCU was only used to transmit UART, however, it also has the capability to receive UART communications.

Finally, the UART has also been setup with 16x oversampling for the purposes of improved noise rejection. The UART communications use the SysTick timer to determine the timing of transmissions.

I2C Channel 1

I2C is used for the purpose of communication between the MCU and the IMU. I2C is setup for synchronous communication between a slave and a master since it makes use of a serial clock line (SCL) with a clock speed of 100kHz. The SDA was also setup for the purposes of data transfer between the master and the slave. No interrupts were setup for the I2C because HAL return values could be used to check the status of master read or transmit functions. No clock stretching was setup for I2C.

No pull-up or pull-down resistors were implemented on the SCL and SDA lines because there is no possibility of bus contention with only one master and one slave. The addressing of the I2C protocol was setup to be a 7-bit address with an implicit LSB.

Due to the fact that the FS bits of the IMU were set to 00, the IMU can measure ±2g (this allows the highest degree of accuracy). For a 16-bit two's complement acceleration measurement, a decimal scale of -32768 to 32767 is used to represent the acceleration. If we measure 1g (gravitational acceleration) components for the different axes, the decimal scale that the acceleration can be measured on will be from -16384 to 16383. This acceleration value will be stored within the MCU registers after the IMU transmits the data through the SDA.

5 Measurements and Results

5.1 Power supply

The power supply is designed to control the power regulation to provide 5V to the STM32F103 and TIC as well as 3.3V to the potentiometer, the accelerometer, and the TIC.

Testing Method

- Connect a nominal 9V battery or bench supply to the female audio jack of the board. Connect positive to the shorter metallic contact and negative to the longer metallic contact.
- Observe the green LED present at D6. If it is on, it indicates a voltage is present on the 5V line [1, p. 6].
- Connect multimeter probes to the metallic connections at F1 and GND.
- Record the voltage measured by the multimeter.

Results and Interpretations

The tabulated measurements can be seen below in table 1.

Component	Measured Voltage (V)
5V Power Regulator	5.1
3.3V Power Regulator	3.3

Table 1: Recorded Power Supply Measurements

The power supply is capable of controlling the 5V and 3.3V power regulators to provide consistent 5V and 3.3V to the appropriate peripherals. The measured values are within an acceptable range of deviation and are thus sufficient for successful verification of the power supply requirements.

5.2 UART Communications

The UART communications are designed to operate at a baud rate of 115200 bps, with 8 data bits, 1 stop bit and no parity checking, using TTL signal levels [1, p. 21]. These communications also have specific timings for different game states. For the maze: the position of the ball, as well as the visibility of the goal and the IMU functionality must be displayed every 300ms. For the tennis: the position of the ball, it's velocity and direction, as well as the position of the bat and the IMU functionality must be displayed every 100ms.

Testing Method (Part 1)

- Connect a nominal 9V battery or bench supply to the female audio jack of the board. Connect positive to the shorter metallic contact and negative to the longer metallic contact.
- Connect a USB cable to the USB port of a PC and the USB input of the NUCLEO microcontroller.
- Download and open a program called "Teraterm". Select "Serial" on the opening screen.
- Click on setup and then serial port. Confirm that the speed, data, parity and stop bits match the specifications (It may be necessary to set the baud rate to 115200 bps).
- Start the execution of your software for each game, one at a time.
- Observe the UART communications that are displayed by Teraterm.

Testing Method (Part 2)

- Part 2 follows on directly from part 1.
- Connect oscilloscope probes to the UART output of the STM32F103 (Rx) and to GND of the board
- Observe the recorded voltage levels on the oscilloscope.
- Identify the interval between successive stop and start bits (This interval is the timing of the UART communications)

Results and Interpretations

For part 1, the results are shown in the images below:

As can be observed from the images below, the UART output on Teraterm is of the correct form for both the maze as well as the tennis game according to the specifications of the project definition document. The timing of the tennis UART is shown to be ~100ms and the timing of the maze UART is shown to be ~300ms which confirms that the UART has been successfully verified to meet the specifications.

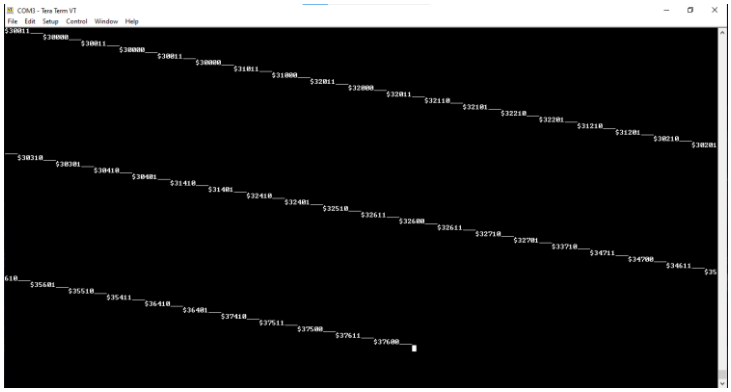

Figure 13: Maze UART Output

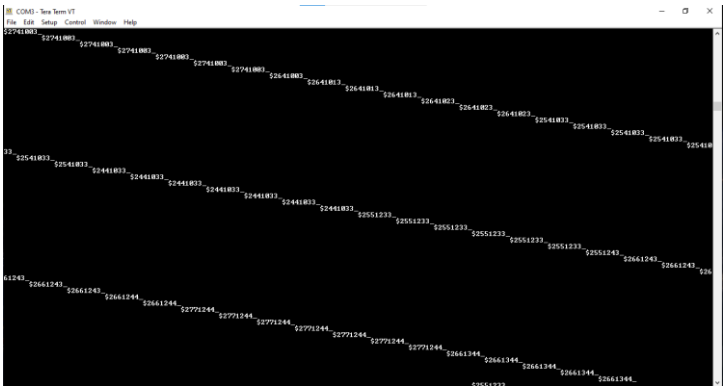

Figure 14: Tennis UART Output

For part 2, the results are shown in the images below:

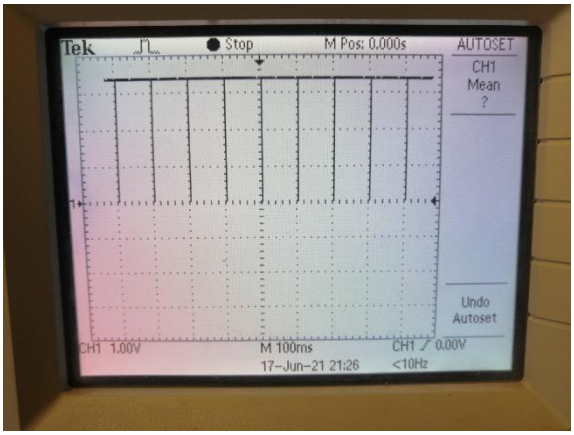

Figure 15: Tennis UART Timing

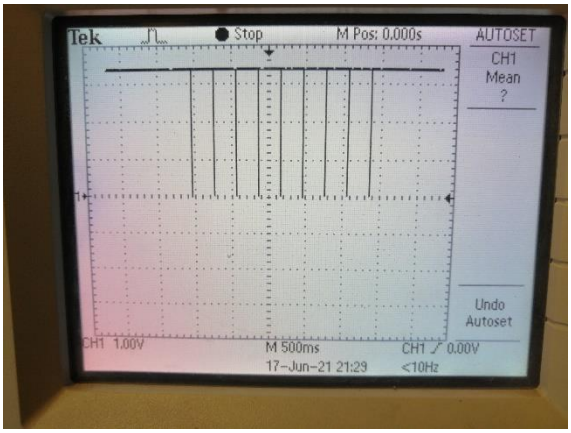

Figure 16: Maze UART Timing

5.3 Buttons

The buttons have been designed to act as a switch between the output of a microcontroller pin and ground. The buttons are designed to be active low (Implies that the STM32F103 pin output will be ground when the button is pushed or high otherwise). The buttons also have to be debounced in software as hardware was deemed too cumbersome to implement.

Testing Method

- Connect a nominal 9V battery or bench supply to the female audio jack of the board. Connect positive to the shorter metallic contact and negative to the longer metallic contact.
- Connect oscilloscope probes to P3 and to GND. P3 is connected to the left button (S2) and to PC0 of the microcontroller.
- Record the transition from high to low voltage when S2 is pressed as well as any button bounce.

Results and Interpretations

The recorded results are shown in the image below:

In the figure below, the rising edge of a transition from 0v to 3.3V can be observed. This occurs when the button is released (active high). When the button is pressed, an inverted process occurs whereby high goes low with the same button bounce that can be observed in the figure below. We have successfully verified that the button does function as a switch. The button is debounced in software as seen in the software design and implementation section.

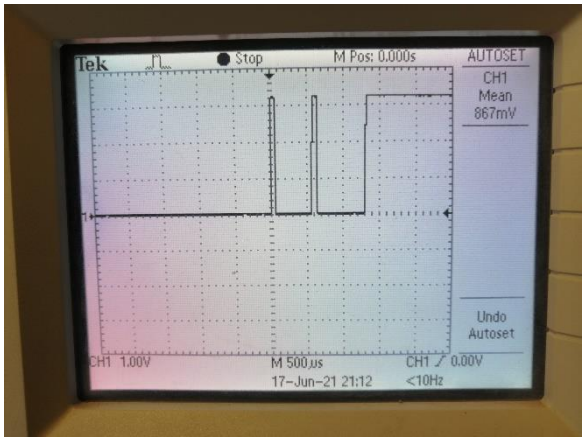

Figure 17: Button Bounce & Transitions

5.4 LEDs (Debug)

The function of the Debug LEDs is to both provide a debugging interface for timing purposes as well as to display the selected maze option on the appropriate LEDs.

Testing Method

- Connect a nominal 9V battery or bench supply to the female audio jack of the board. Connect positive to the shorter metallic contact and negative to the longer metallic contact.
- Connect the multimeter probes to the positive of LED D2 and to GND.

- Record the observed voltage over the LED and current through the LED.

Results and Interpretations

The tabulated measurements can be seen below in table 2.

Measurement	Measured Voltage(V)/Current(mA)
V_{LED}	1.83
I_{LED}	1.40

Table 2: Recorded LED D2 Measurements

As can be seen from the above measurements, the voltage over the LED will result in the LED turning on (assuming the necessary forward voltage to turn an LED on is ~1.7V). The current through the LED is also a safe current which ensures an appropriate brightness. This means that the debug LEDs have been successfully verified to meet their requirements.

5.5 Dot Matrix

The dot matrix is designed to be a display for feedback to the user while playing the arcade game. This feedback is in the form of LEDs (connected in series with 1kΩ resistors) lighting up as a result of the current that is sourced by the relevant MCU pins. The MCU is protected by MOSFETs on the dot matrix to prevent the pins from sinking too much current.

Testing Method

- Connect a nominal 9V battery or bench supply to the female audio jack of the board. Connect positive to the shorter metallic contact and negative to the longer metallic contact.
- Connect multimeter probes to the positive of an LED and to GND (In the case where it is the only LED that a MCU pin sinks current for).
- Record the observed voltage over the LED and current through the LED.
- Repeat the above process for the case where a MCU pin sinks current for all 8 LEDs in a column.
- Do a visual observation to check that the appropriate LEDs turn on (with sufficient brightness) and off whenever MCU output is given for the LEDs to do so.

Results and Interpretations

The tabulated measurements can be seen below in table 3.

	V_{LED} (V)	I_{LED} (mA)
Pin sinks for 1 LED	1.77	1.34
Pin sinks for 8 LEDs	1.72	1.44

Table 3: Recorded LED Measurements

As can be seen from the measurements above, the voltage over the LED is always sufficient to keep the LED on (V_{LED} > Vf) and the current through the LED is of a suitable level in order to not damage the MCU pin sourcing the current while keeping the brightness of the LED appropriate. The MOSFETs protect the pin when it sinks for 8 LEDs as the pin is not designed to sink for 11.5mA

(even if its maximum ratings are above this). The dot matrix also passes the visual inspection test with the correct LEDs changing. The dot matrix has therefore been verified to work correctly.

5.6 Slider

The slider is designed to output a range of 0-3.3V of analogue voltage which is determined by the contact with a resistive strip within the slider. The slider needs to provide appropriate voltage levels when the contact with the resistive strip is varied.

Testing Method

- Connect a nominal 9V battery or bench supply to the female audio jack of the board. Connect positive to the shorter metallic contact and negative to the longer metallic contact.
- Connect multimeter probes to PC1 and to GND.
- Observe the measured voltage as the contact with the resistive strip is varied from right to left (GND to VCC)

Results and Interpretations

The tabulated measurements can be seen below in table 4.

Contact position	Measured Voltage(V)
GND	0
VCC/2	1.59
VCC	3.32

Table 4: Recorded Slider Output Measurements

The above results show that as the slider position is varied from GND to VCC, the output of the slider varies from 0V to 3.32V. This is the expected range that the slider was designed to operate within. The slider is therefore successfully verified to meet its requirements.

5.7 IMU

The LIS3DH - 3-axis MEMS accelerometer is designed to act as a slave device to the MCU which communicates via I2C protocol. The IMU must measure the gravitational acceleration in the X, Y and Z axes and communicate this information to the STM32F103.

Testing Method

- Connect a nominal 9V battery or bench supply to the female audio jack of the board. Connect positive to the shorter metallic contact and negative to the longer metallic contact.
- Connect oscilloscope probes to PB7 (SDA) and to GND.
- Execute the section of code where the IMU is calibrated, asked for data, and then gives data.
- Observe the I2C address frame for a Master read and write on the oscilloscope.
- If the start and stop bits are displayed as well as the appropriate acknowledge bits from the IMU, then the IMU is functional and communicates with the MCU.

- Check the register contents (in software) of the MCU which store the acceleration data to verify that the IMU samples the correct data.

- Do a visual check to ensure that the IMU influences the movement for the games in an appropriate manner.

Results and Interpretations

The implementation of the IMU was successful in hardware, however, it was not functional within the software of the MCU. The IMU could not communicate the appropriate acknowledge bit (instead displaying HAL_BUSY) and so communication between the IMU and the MCU unit was severed. This could be due to the IMU not being able to reset or that the current I2C instance is broken if the supply voltage is not a stable connection. Explanations for the shortcomings of the IMU are further expanded upon in the conclusion of this report.

6 Conclusions

The system that was built for this project fulfils all the compulsory requirements except for one. The system is unable to use the IMU to sample data during game play. This is proposed to be attributed to the fact that the IMU could not acknowledge that the MCU was trying to communicate with it when the MCU was requesting data or writing data to the IMU over the SDA. The measured power supply of the IMU met the requirements for an appropriate input while the software implementation of the I2C protocol featured all the necessary requirements to calibrate, communicate with and sample from the IMU. No sensor data was able to be detected over the SDA line and as such, master read and write transmissions were always stuck in a HAL_BUSY mode (even after the firmware was checked and known bug fixes implemented).

A design shortcoming that is quite glaring is that the only display feedback for a user is the LED dot matrix. The dot matrix was difficult and time consuming to implement, and it also had the possibility to damage the microcontroller if careful considerations were not made regarding the LED currents. It would be recommended that an LCD screen is used instead of an LED dot matrix. An LCD screen is much quicker and more elegant to setup since the LED dot matrix had to be built first. The dot matrix also has limited functionality tied to its mono-colour lights and limited resolution.

A possible addition to this project in the future is audio. Audio for the games such as sound effects or background music can contribute to the retro aesthetic of the game console. The MCU has the capability to interface with I2S audio via SPI.

7 Appendices

Complete Circuit Schematic

Refer to the PCB layout of the Base Board for component number positions.

Name	Hardware Component	Value
C1	5V Regulator	100nF
C2	5V Regulator	10uF
C3	5V Regulator	100nF
Cx	3.3V Regulator (LHS)	100nF
Cy	3.3V Regulator (RHS)	100nF
R3	Debug LEDs	1kΩ
R5	Debug LEDs	1kΩ
R7	Debug LEDs	1kΩ
R11	Debug LEDs	1kΩ
R12	5V Regulator	1kΩ
Rx	All LED Matrix Resistors	1kΩ

Table 5: Resistor and Capacitor Values in the Circuit

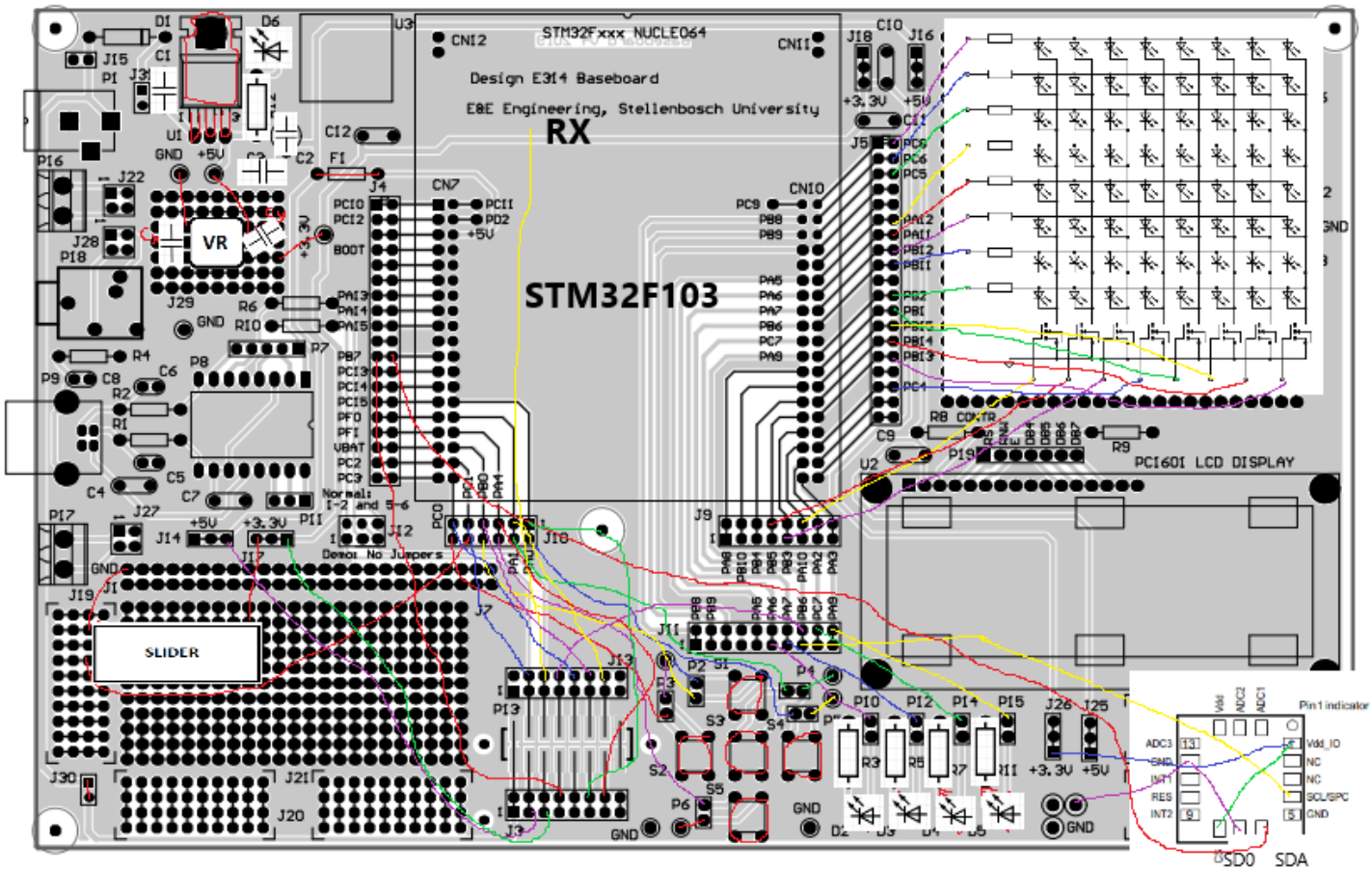

Figure 18: Full Circuit Schematic on PCB

STM32 Pinout Table

Pin	Hardware Component	Configuration
PA0	Button	GPIO_Input (Pull-up Resistor)
PA1	Button	GPIO_Input (Pull-up Resistor)
PA2	UART	USART2_TX
PA3	UART	USART2_RX
PA4	Button	GPIO_Input (Pull-up Resistor)
PA6	Debug LED	GPIO_Output
PA7	Debug LED	GPIO_Output
PA9	Debug LED	GPIO_Output
PA10	LED Dot Matrix	GPIO_Output
PA11	LED Dot Matrix	GPIO_Output
PA12	LED Dot Matrix	GPIO_Output
PB0	Button	GPIO_Input (Pull-up Resistor)
PB1	LED Dot Matrix	GPIO_Output
PB2	LED Dot Matrix	GPIO_Output
PB3	LED Dot Matrix	GPIO_Output
PB5	LED Dot Matrix	GPIO_Output
PB6	IMU	I2C1_SCL
PB7	IMU	I2C1_SDA
PB11	LED Dot Matrix	GPIO_Output
PB12	LED Dot Matrix	GPIO_Output
PB13	LED Dot Matrix	GPIO_Output
PB14	LED Dot Matrix	GPIO_Output
PB15	LED Dot Matrix	GPIO_Output
PC0	Button	GPIO_Input (Pull-up Resistor)
PC1	Slider	ADC1_IN11
PC4	LED Dot Matrix	GPIO_Output
PC5	LED Dot Matrix	GPIO_Output
PC6	LED Dot Matrix	GPIO_Output
PC7	Debug LED	GPIO_Output
PC8	LED Dot Matrix	GPIO_Output

Table 6: Pins, Hardware Components and Pin Configurations

References

[1] Dr. A. Barnard, Dr. C. Fisher, “Design (E) 314 Projection Definition”, Project Standard [Online], 15 Mar. 2021 [Revised 21 May 2021], Available: https://learn.sun.ac.za/pluginfile.php/2860111/mod_resource/content/0/DesignE314_2021_PDD%20%284%29.pdf

[2] Jameco Electronics,” Positive voltage regulators”, L7800 series, 3 Aug. 2006

[3] STMicroelectronics,” Medium-density performance line ARM®-based 32-bit MCU with 64 or 128 KB Flash, USB, CAN, 7 timers, 2 ADCs, 9 com. interfaces”, STM32F103x8/STM32F103xB, 1 Jun. 2007 [Revised 21 Aug. 2015]

[4] Microchip,” Low Quiescent Current LDO”, MCP1700, Nov. 2005 [Revised Oct. 2013]

[5] STMicroelectronics,” Arm® Cortex®-M4 32b MCU+FPU, 210DMIPS, up to 1MB Flash/192+4KB RAM, USB OTG HS/FS, Ethernet, 17 TIMs, 3 ADCs, 15 comm. interfaces & camera”, STM32F405xx/STM32F407xx, 15 Sep. 2011 [Revised 14 Aug. 2020]

[6] Chris Coulston. (Unknown). *A common input device - buttons and switches* [Online]. Available: https://inside.mines.edu/~coulston/courses/EENG383/lecture/lecture08.html

[7] ON Semiconductor,” Small Signal MOSFET 200 mAmps, 60 Volts”, 2N7000G, Apr. 2011

[8] STMicroelectronics,” MEMS digital output motion sensor: ultra-low-power high-performance 3-axis "nano" accelerometer”, LIS3DH, 21 May 2010 [Revised 12 Dec. 2016]

[9] STMicroelectronics,” LIS3DH: MEMS digital output motion sensor ultra low-power high performance 3-axis “nano” accelerometer”, AN3308, 14 Jan. 2011