Design (E) 314
Technical Report

Dot-Matrix Arcade Game Console

Author:
Ethan Kraus

Student Number:
21573751

June 21, 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*	21573751 Studentenommer / *Student number*
E Kraus Voorletters en van / *Initials and surname*	June 21, 2021 Datum / *Date*

Abstract

This report details the design and making of an arcade system. This arcade system uses an STM32F103RB Nucleo Board with MicroController Unit (MCU) to communicate with various peripherals including a Light-Emitting Diode (LED) Matrix, buttons, a slider potentiometer and Inertial Measurement Unit (IMU). Using these peripherals with communication via General Purpose Input Output (GPIO), Universal Asynchronous Receiver/Transmitter (UART) and Inter-Integrated Circuit (I^2C) a set of games including a Maze and Tennis game are played. It is found that the system works very well and with testing meets all the user requirements, however, timing is slightly unreliable and thus a Real-Time Operating System is suggested for improvement of the system.

Contents

1 Introduction **5**

2 System Description **6**

3 Hardware Design and Implementation **7**
3.1 Power Supply . . . 7
3.2 UART Communication . . . 7
3.3 Buttons . . . 8
3.4 Dot Matrix . . . 8
3.5 LEDs (Debug) . . . 10
3.6 Slider . . . 10
3.7 IMU Interface . . . 10

4 Software Design and Implementation **11**
4.1 Control Logic . . . 11
4.2 Button Bouncing . . . 14
4.3 Data Flow and Processing . . . 14
4.4 IMU Interface . . . 15
4.5 Peripheral Setup . . . 16

5 Measurements and Results **17**
5.1 Power Supply . . . 17
5.2 UART Communication . . . 17
5.3 Buttons . . . 17
5.4 Dot Matrix . . . 17
5.5 LEDs (Debug) . . . 18
5.6 Slider . . . 18
5.7 IMU Interface . . . 18

6 Conclusion **19**

A Complete Schematic **21**

B STM32 Pinout **22**

C 2-Player Tennis Game **23**

List of Figures

1 System Block Diagram . . . 6
2 L7805CV 5Voltage (V) Voltage Regulator Circuit . . . 7
3 MCP 1700-3302E 3.3Voltage (V) Voltage Regulator Circuit . . . 7
4 UART Protocol . . . 8
5 UART Communication . . . 8
6 Active Low Button Schematic . . . 8
7 Dot Matrix Schematic . . . 9
8 Slider Schematic . . . 10
9 Starting Sequence Pseudocode . . . 11
10 Maze Game Psedocode . . . 12
11 Maze Game Pseudocode . . . 13
12 Button Bouncing Pseudocode . . . 14
13 SysTick Timer Pseudocode . . . 14
14 displayLED function Pseudocode . . . 15

15 I²C Communication Protocol 16
16 UART Oscilloscope 17
17 Row 0 LEDs on Oscilloscope during Configuration 18
18 Complete Schematic 21
19 2-Player Tennis Game Pseudo-code 23

List of Tables

1 Peripheral Setup 16
2 STM32 Pinout 22

List of Abbreviations

ADC Analog-To-Digital Converter

EXTI External Interrupt

GPIO General Purpose Input Output

HAL Hardware Abstraction Library

I²C Inter-Integrated Circuit

IMU Inertial Measurement Unit

LED Light-Emitting Diode

MCU MicroController Unit

MOSFET Metal-Oxide-Semiconductor Field-Effect Transistor

PCB Printed Circuit Board

PDD Project Definition Document

STM ST Microelectronics

TIC Test Interface Connector

TTL Transistor-Transistor Logic

UART Universal Asynchronous Receiver/Transmitter

List of Symbols

bps bits per second

° degree

Hz frequency

mA milliAmpere

mm millimeter

ms millisecond

mW milliWatt

$k\Omega$ kiloohm

V Voltage

1 Introduction

This arcade system uses an STM32F103RB Nucleo Board with MCU to communicate with various peripherals including a LED Matrix, buttons, a slider potentiometer and IMU. Using these peripherals with communication via GPIO, UART and I^2C a set of games including a Maze and Tennis game are played. The arcade game has a set of requirements which needs to be met including:

- $9V$ supply power regulated to $5V$ and $3.3V$.
- Sampling of an accelerometer, potentiometer and buttons.
- Use an UART connection to transit information about the arcade at 115 200 *bps* with an 8N1 configuration.
- The games should be displayed on an 8x8 LED Dot Matrix, smaller than 60x60mm. Each row/column must be updated every millisecond (ms) and a start-up sequence must be followed to verify the LEDs all work.
- Buttons will allow a game to be selected (Left - Maze; Middle - Tennis; Right - 2-player Tennis)
- Once the games end a calibration pattern must be shown so a new game can be selected.

These user requirements will shape how the games are implemented. The maze game involves choosing one of four mazes and then moving a ball from a start position to end position. The tennis game involves moving a bat to hit a ball until the ball goes behind the bat and off the screen.

This report will first detail the hardware design, followed by the software design and finally measurements ensuring the design parameters are met for the maze and tennis game. A complete schematic and MCU pinout is shown respectively in Appendices A and B, while implementation of the 2-player Tennis Game is discussed in Appendix C.

2 System Description

Figure 1 shows the block diagram for the system. It includes the: power supply and regulation in red supplied to the MCU and various sensors; analogue signals from the slide potentiometer to the MCU's Analog-To-Digital Converter (ADC) input as well as to the Test Interface Connector (TIC) in light blue; regulated $3V$ signals in dark blue from the active low push buttons are connected to the external interrupt of the MCU and TIC; GPIO pins output to the LED matrix in purple; I^2C signals (SDA/SCL) sent from the accelerometer to the MCU and TIC in green; as well as UART communication (TX/RX) sent from the MCU to TIC in purple.

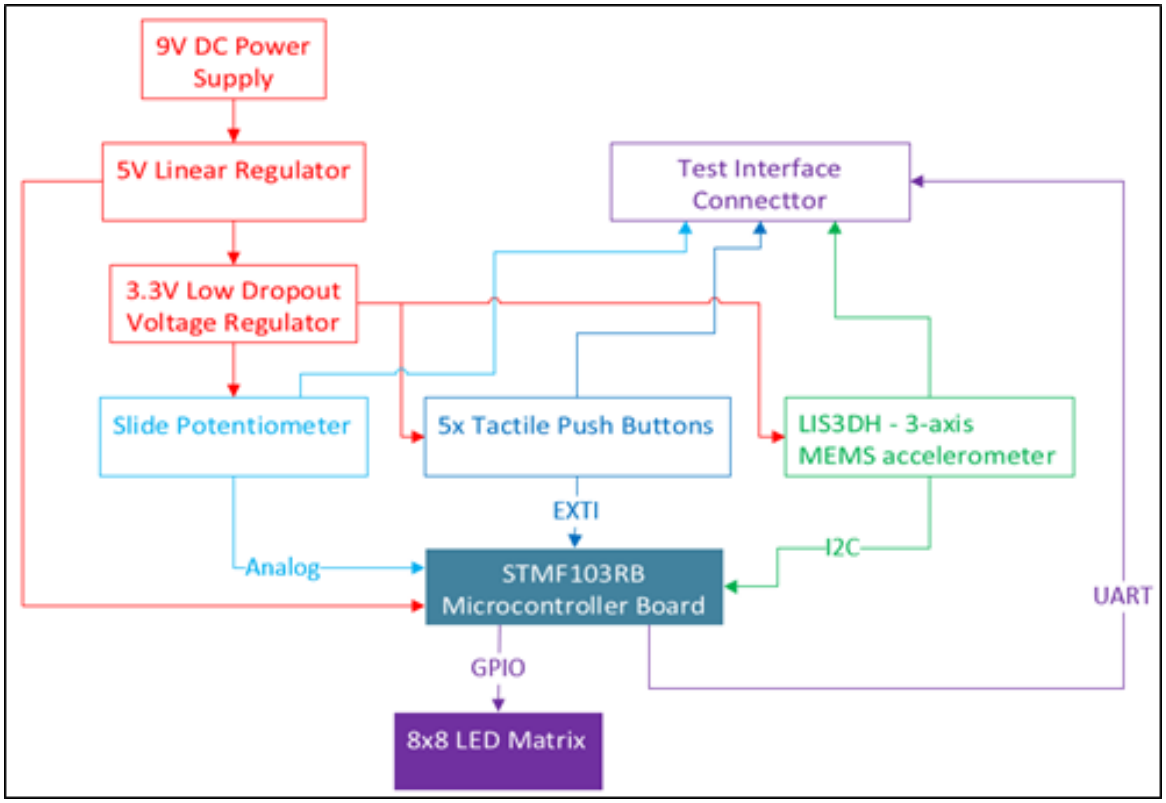

Figure 1: System Block Diagram

3 Hardware Design and Implementation

A STM32 Nucleo-64 development board with STM32F103RB MCU board is selected as it has the characteristics which meet the requirements to play the games including 64 pins for various tasks including GPIO and analogue input, UART and I^2C communication protocol as well as 128 kilobytes of flash memory [1, Tab. 2]. External hardware excluding the nucleo board and Printed Circuit Board (PCB) on which everything is placed is discussed throughout the rest of the section.

3.1 Power Supply

The $9V$ battery supply will first be regulated by a L805CV providing $5V$ power to the MCU as well as other components on the board. The L805CV is chosen as its output voltage range ($4.8 - 5.2V$ [2, Tab. 3]) falls within the Nucleo board external voltage input range ($4.75 - 5.25V$ [1, Tab. 7]). This $5V$ will then be regulated by an MCP1700-3302E/TO to provide $3.3V$, required for various components such as ADC as well as for GPIO high Transistor-Transistor Logic (TTL) signals. Capacitors will be added to the regulators to improve stability of the regulators. Their circuits are respectively taken from the Baseboard Schematic [3] and MCP1700 datasheet [4] and are shown respectively in Figures 2 and 3.

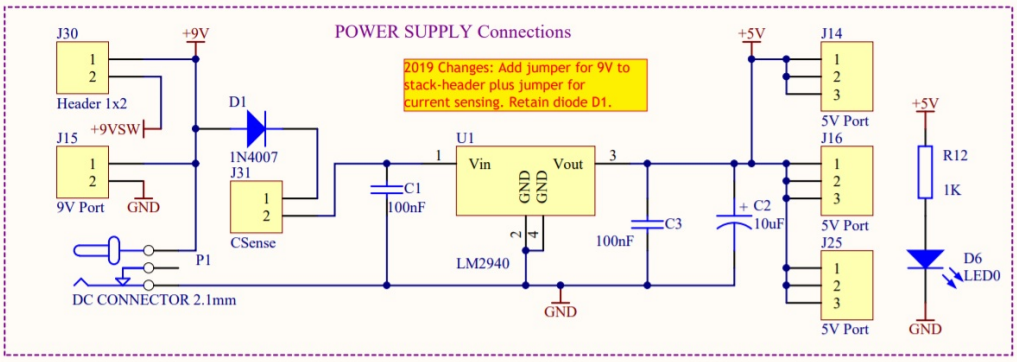

Figure 2: L7805CV $5V$ Voltage Regulator Circuit

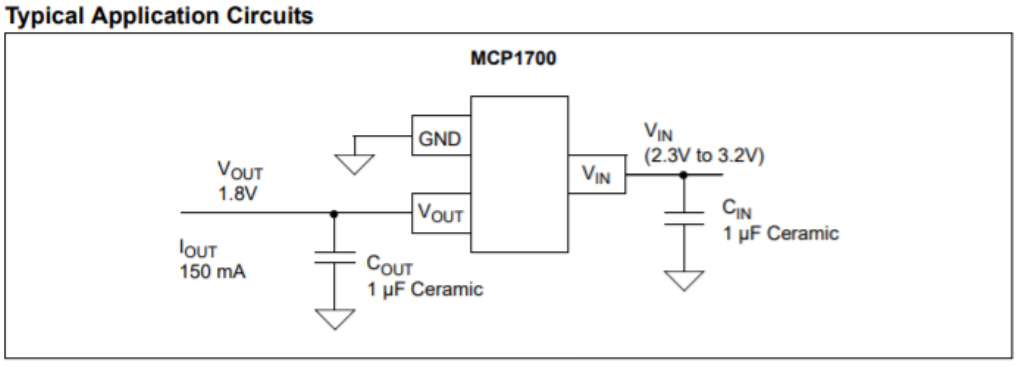

Figure 3: MCP 1700-3302E $3.3V$ Voltage Regulator Circuit

3.2 UART Communication

Throughout the program UART will be used for debugging as well as a way to identify the state of the arcade and games. The UART from the Nucleo board has been redirected to the board's RX pin. UART is selected to be of 8N1 (8 data bits, no parity bits and a single stop bit) protocol and is sent at a baud rate of 115 200 *bps* as shown in Figure 4.

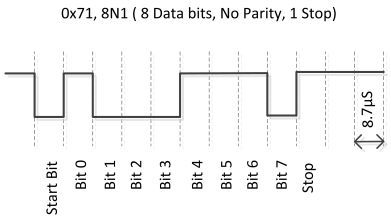

Figure 4: UART Protocol

The information for the configuration and state of each game is shown in Figure 5 from the Project Definition Document (PDD) [5, Tab. 5].

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 5: UART Communication

3.3 Buttons

Buttons are connected active low and are connected to $3.3V$ as shown in Figure 6. The value of pull up resistance is arbitrary, however, a large resistor should be used to ensure there is minimal current and thus power loss but not too large to effect the reliability of the reading [6]. $10k\Omega$ resistors are chosen. Capacitors are considered for button de-bouncing, however, it is decided that it will be simpler and cheaper to implement de-bouncing in software.

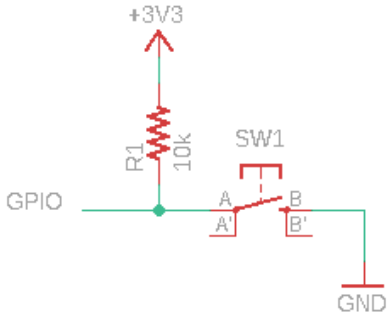

Figure 6: Active Low Button Schematic

3.4 Dot Matrix

Figure 7 shows the circuit schematic for the LED Matrix with the connected MCU GPIO pins, hence, each pin controlling a row or column of the LEDs. The resistor and Metal-Oxide-Semiconductor Field-Effect Transistor (MOSFET) positions have been placed as seen in Figure 7 and thus each row must be sunk every millisecond with the corresponding on LEDs columns set high, as this will cause the designed current to run through each of the LEDs; as opposed to powering a single column and sinking multiple MOSFETs which will result in the designed current being split amongst the LEDs which are sunk.

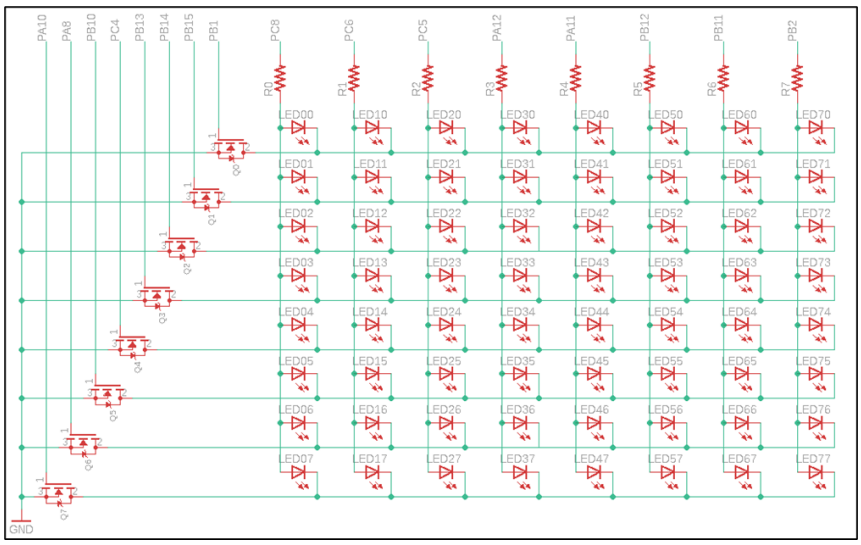

Figure 7: Dot Matrix Schematic

The absolute maximum current for the MCU is $150mA$ and $25mA$ per GPIO pin [7, Tab. 7], while the operating power dissipation is $444mW$ [7, Tab. 9]. Assuming the output voltage from each GPIO pin in $3.3V$, the total output current from the GPIO pins should be $444mW/3.3V$=$134.54mA$. During operation, the maximum current drawn from the board's peripherals and an external clock is $50mA$ [7, Tab. 15]. Hence, the most current each LED should draw is:

$$I_{max} = \frac{\frac{P_{max}}{V} - I_{peripheral}}{8} = \frac{\frac{0.444}{3.3} - 0.05}{8} = 10.5\,\text{mA} \tag{1}$$

If the GPIO pin outputs more than $8mA$ the output voltage is relaxed and can drop to $V_DD-1.3 = 2\,\text{V}$ [7, Tab. 36]; which is very close to the turn on voltage of the LEDs, hence, the output voltage should be limited to $8mA$. As the current increases so does the brightness [8, p. 3], the LED is tested varying the current from 0-$8mA$ and it is decided that this brightness at $8mA$ is best, thus the TTL output voltage from the board limits the current to be designed for to $8mA$. As seen in Figure 7 the total current which needs to be sunk could be a maximum of $8 \cdot 8\,\text{mA} = 64\,\text{mA}$, thus a MOSFET will be used to sink the current to ground and not to the MCU which can sink a maximum of $25mA$ [7, Tab. 7]. The 2N7000 N-Channel Enhancement Mode Field Effect Transistor is selected as it can drain a continuous $200mA$ [9, p. 2] which is well above the required $64mA$, furthermore, the maximum turn on and turn off times are 20ns which is much faster than the refresh rate of every $8ms$ and will not cause a problem. For a conservative resistance, the voltage drop across the MOSFETs can be approximated as negligible [9, Fig. 1] and the voltage across the LED with $8mA$ flowing through it is expected to be just above $1.9V$ [8, p. 3], thus, the resistor value used to match this output current is:

$$R = \frac{V_{GPIO} - V_{LED}}{I_{design}} = \frac{3.3 - 1.9}{8} = 175\,\Omega \tag{2}$$

Thus, a $0.18k\Omega$ resistor is selected from the E12 resistor series. The MOSFETs may require gate resistors if ringing is experienced (parisitic oscillation of the gate voltage caused by the gate's capacitance in series with the wires inductance leading to a positive feedback loop). After testing the dot matrix it if found that there is negligible ringing, thus, gate resistors are not required.

One single row/column of LEDs is to be displayed every one millisecond when the SysTick interrupt occurs. This results in each row/column of LEDs flickering on for one millisecond every $8ms$. Hence the duty cycle for a row/column of a single LED is $1ms/8ms$=12.5%, while the refresh rate per LED is $1/8ms$=$125Hz$ which is well above the flicker fusion threshold of approximately $60Hz$, therefore, the LEDs will appear as a solid light approximately 8x dimmer than if a constant current were applied to them.

3.5 LEDs (Debug)

Debug LEDs are used to show which Maze the game is on. The LEDs have been designed taking into account the maximum current expected to be drawn by the board from Section 3.4, thus each LED should not draw more than:

$$I = 10.5 - 8 \cdot \frac{3.3 - 1.9}{180} = 2.72\,\text{mA} \tag{3}$$

This is greater than the expected average current through an LED in the dot matrix of $I_{avLED} = \frac{\frac{3.3-1.9}{180}}{8} = 0.97\,\text{mA}$ thus to keep the brightness of the debug LEDs similar to those of the dot matrix:

$$R = \frac{3.3 - 1.9}{0.97} = 1.44\,\text{k}\Omega \tag{4}$$

Thus, $1.5k\Omega$ resistors are chosen to be in series with the debug LEDs.

3.6 Slider

A slider was originally used to control the vertical motion for the tennis game, however, is no longer used by the end of the project. It connected to the ADC input of the MCU as shown in Figure 8.

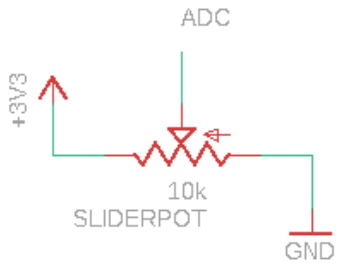

Figure 8: Slider Schematic

This will allow the voltage read at the ADC input to be read as a value from 0-3.3V as required by the ADC input depending on the resistance of the slider.

3.7 IMU Interface

The IMU is a LIS3DH: MEMS digital output motion sensor. It is connected to the MCU with long wires allowing the IMU to move independently from the PCB. The protocol selected to communicate with the IMU is selected to be I^2C. To achieve this the IMU requires a 3.3 V and GND power supply as well as to be connected to the SDA and SCL MCU bus lines. Furthermore, using the external 3.3V power, for device identification the SA0 line is set high and the CS line is set high to specify I^2C and not SPI communication.

4 Software Design and Implementation

The software is all implemented using STM32Cube IDE on the Nucleo board. Buttons are treated as external inputs and flagged when they are pressed. The slider is set as an ADC input while the IMU communicates via I^2C on the SDA (Serial Data) and SCL (Serial Clock) bus lines. UART is also configured for transmitting (TX) and receiving (RX), while GPIO outputs control all the LEDs and MOSFETs including those in the dot matrix as well as the ones used in debugging.

4.1 Control Logic

As shown in Figure 9, once the board is plugged in it runs through its calibration and then waits for a button press to start one of the games. Depending on the button pressed the board will move into a function where the game is run.

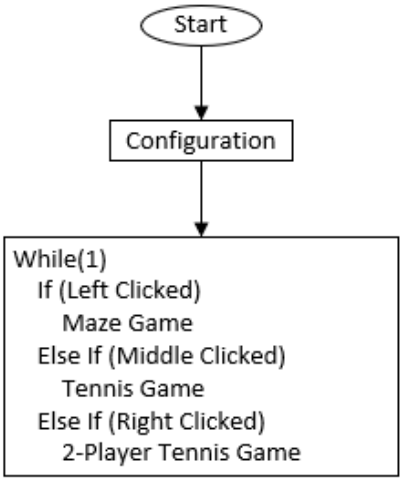

Figure 9: Starting Sequence Pseudocode

The Maze game works as shown in Figure 10 with the variable timer flag set to $300ms$ (timers are discussed in Section 4.3). The game is first initialised by declaring variables and placing the game indicating the maze be selected. Then the one of four mazes is selected using up/down buttons, the middle button is pressed to select the maze. Then the game is set up with the LEDs on indicating the maze wall, while, a light toggling at $300ms$ represents the current position of the ball and the end of the maze is represented by a LED toggling every $100ms$. The player must then move through the maze to reach the end using the buttons or IMU. Conditional checking will ensure the player does not move through an LED (MazeWall) or off the screen (limit). If the middle button is pressed or end is reached the game will end.

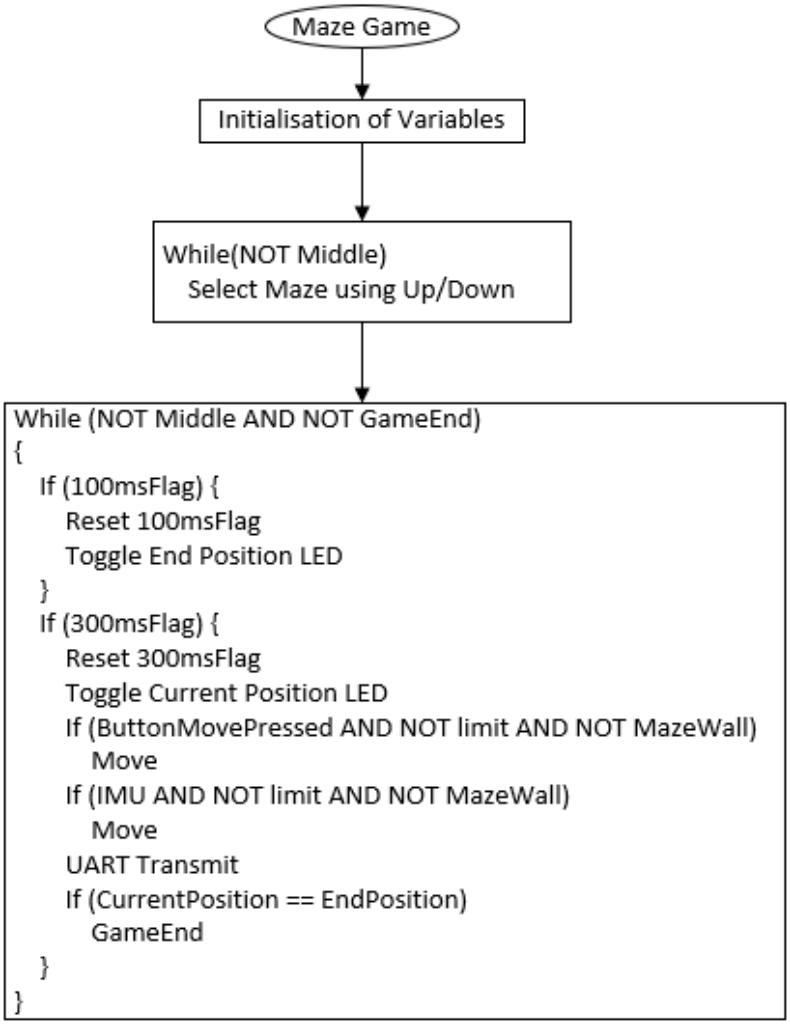

Figure 10: Maze Game Psedocode

For the tennis game the bat should be able to move based on buttons, the slider and/or the accelerometer. The bat will move every $100ms$, after which UART will be used to transmit the current details of the game. The speed of the ball will start moving every $700ms$ and decrease by $50ms$ every three times it hits the bat until the game is over or ball speed is at a minimum of $300ms$. The game ends if the ball goes off the left of the screen or if the game is quit. The final tennis game implementing buttons and the IMU works as shown in Figure 11 with the variable timer changing depending on the velocity. The game is first initialised by declaring variables and starting positions. The tennis ball then moves from LED [7, 4] directly left and the game begins. The tennis ball initially moves every $700ms$ at a velocity of one and then after every third hit by the bat the velocity is increased by one, thus the speed increased by $50ms$ to a maximum of $300ms$.

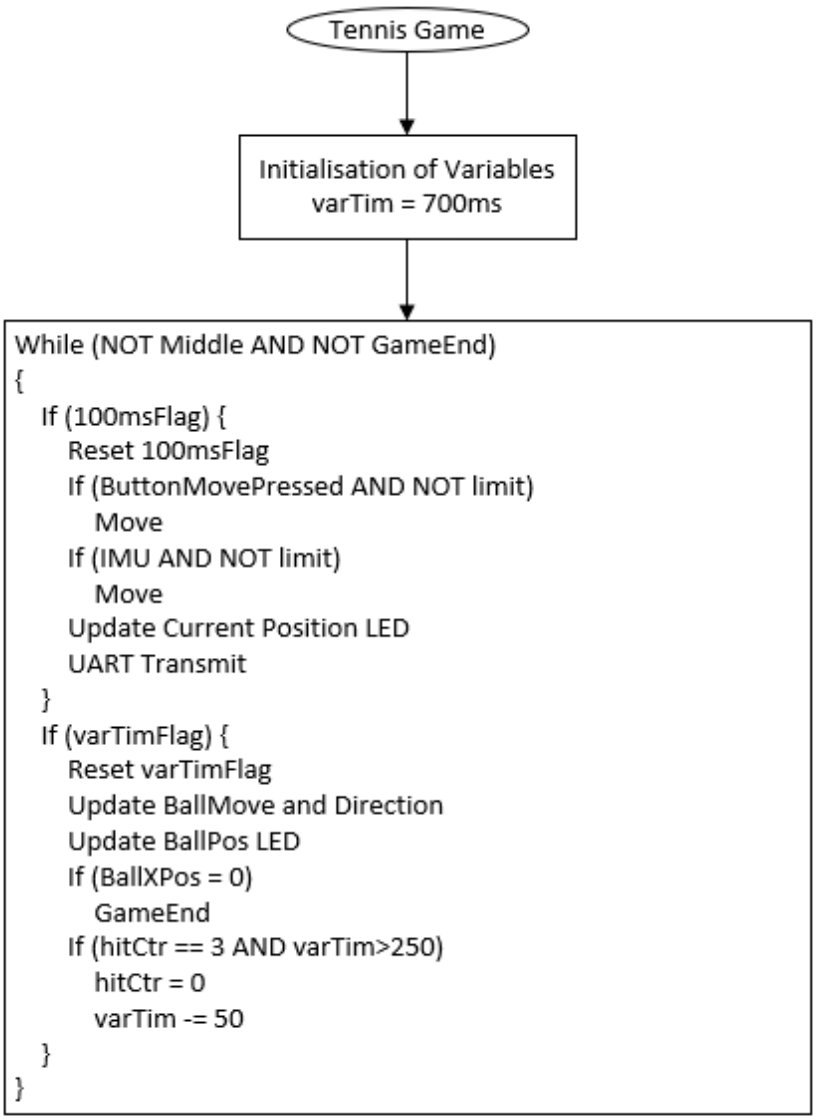

Figure 11: Maze Game Pseudocode

The two player tennis game is discussed in Appendix C

4.2 Button Bouncing

The buttons are managed as external interrupts, thus a simple time delay can be used to remove button bouncing as shown in Figure 12.

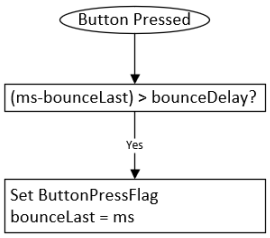

Figure 12: Button Bouncing Pseudocode

This time delay logic is directly implemented in the interrupt handler using global variables. It is found that a delay of $250ms$ works best to eliminate button bounce.

4.3 Data Flow and Processing

The timing of the games all depends on the SysTick timer which is called every millisecond. This allows a single column in the dot matrix to be updated every millisecond based on the counter (global variable "ms") as shown in Figure 13. Furthermore, this timer is used to set flags every $100ms$ as well as variably (300msFlag and varTimFlag) as used in the Maze and Tennis games respectively shown in Figures 10 and 11.

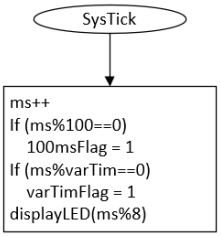

Figure 13: SysTick Timer Pseudocode

When flags are set the corresponding control logic is performed as shown in Section 4.1. The flag is then reset by the control logic so that it is set again in the required interval and the control logic can again be processed at the required time interval.

The dot matrix is controlled by the "displayLED" function and globally defined matrix with 64 boolean values. As shown in Figure 13, every millisecond this function will be called with an argument value from 0-7 depending on the row which needs to be updated and shown in that millisecond. Shown in Figure 14 the function will then from the data stored in the matrix get the corresponding eight bits of data for the corresponding row; if the bit value is high, the corresponding column GPIO will be turned on otherwise it will be turned off. This will allow the entire dot matrix's LEDs to be controlled individually as previously discussed in Section 3.4. Furthermore, implementing the control logic discussed in Section 4.1 is made much easier, as when a move is made or LED toggled, only the matrix which stores the values of the LEDs needs to be updated and the function will ensure the LEDs are correct.

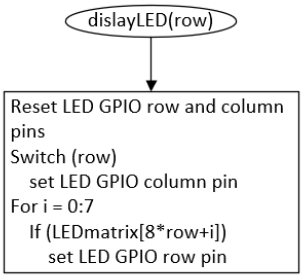

Figure 14: displayLED function Pseudocode

Buttons being pressed are handled as external interrupts and are triggered on a falling edge due to their connection being active low. When a button is pressed its corresponding flag is set, once the flag has been registered in the control logic and the move performed the flag is reset. If the flag has been reset and the bounce delay has been exceeded the button flag can be set again as shown in Section 4.2.

UART and I^2C call are made in blocking mode as it is found that the time they take to perform their tasks is consistent meaning that the control logic reliably performs at the time interval required and is furthermore negligible in comparison to the time interval required (minimum of $100ms$)and thus does not need to be handled as an interrupt or through direct memory access.

Similarly, when ADC was used, it was in blocking mode and found to take longer (2-3ms), however, its consistency allowed the UART and other control logic to function reliably while sticking to their time intervals.

The games are all run in their own functions as shown in Section 4.1 and have their own while loops. From the accuracy and reliability of the UART messages it can be concluded that the MCU is fast enough and will run through these while loops multiple times per $100ms$ entering inner functions if flags are set.

4.4 IMU Interface

As previously mentioned, the IMU communicates via I^2C and the SA0 pin is set high. Thus, when addressing the device to read or write, 0x33 and 0x32 are respectively used [10, Tab. 16]. The device can be left in default mode, with a full scale range of 2g, other than two registers which are CTRL_REG1 and CTRL_REG4 where the device needs to be set to normal mode with high resolution outputs, the X, Y and Z axis need to be enabled, and the refresh rate of the device set to $25Hz$ [10, p. 35-37]. $25Hz$ is selected as this is faster than the maximum refresh movement rate of $10Hz$ ensuring the movement is sampled between reading the register while power consumption is minimised. 12 bit two's complement are outputted on two registers per axis when high resolution is selected. Buttons are assumed pressed when one the axis of the IMU was at an angle of greater than 30° to the horizontal. Because of the 12 bit resolution the range varied from -1024 to 1023 representing -1g to 1g and $sin(30°) = 0.5$, therefore a button pressed was simulated when the value of the axis become greater than or equal to 512 or less than -512. Using the Hardware Abstraction Library (HAL) library and I^2C memory reads and writes, the device is communicated with as shown in Figure 15 [11].

Table 17. Transfer when master is writing one byte to slave

Master	ST	SAD + W		SUB		DATA		SP
Slave			SAK		SAK		SAK	

Table 20. Transfer when master is receiving (reading) multiple bytes of data from slave

Master	ST	SAD+W		SUB		SR	SAD+R			MAK		MAK		NMAK	SP
Slave			SAK		SAK			SAK	DATA		DATA		DATA		

Figure 15: I^{2}C Communication Protocol

4.5 Peripheral Setup

All peripheral including the SysTick timer, GPIO, External Interrupt (EXTI), ADC, UART and I^{2}C have been set up using the HAL library provided by ST Microelectronics (STM). The setup of the individual peripherals is shown in Table 1.

Table 1: Peripheral Setup

PERIPHERAL	**SETUP**
SysTick	$1ms$ Interrupt. This interrupt will implemented to control the timing for movement and peripheral readings for the various games.
GPIO (Output)	Push-Pull output mode with no pull-up/pull-down. GPIO outputs will mainly be used to control the dot matrix through turning on the corresponding LED column with MOSFET row in the displayLED function. GPIO outputs will also be used to control the debug LEDs to indicate which maze is selected.
GPIO EXTI	External Interrupt input mode with rising-edge trigger detection and no pull-up/pull-down. The five buttons will all be implemented using the external interrupts thus 5 interrupt lines will be required. Flags along with the external interrupts ensure the button presses are not missed. The buttons each have their own external pull up resistors and thus do not require internal pull.
ADC	Regular conversion launched in software with 1.5 cycles sampling time. ADC will be sampled if it is required that the slide potentiometer be used for vertical movement of the tennis bat. The slider is connected between $3.3V$ and ground as indicated in the Nucleo datasheet [7, Tab. 46]. The slider will have more than enough resolution with 12 bits as only 7 positional segments are required for the vertical movement.
UART	8N1 (8 bits, no parity bit and 1 stop bit) @ 115 200 *bps* with transmit and receive on separate pins. UART communication will be used for transmitting information regarding the current state to the TIC. The short messages and relatively high baud rate allow the messages to be sent in blocking mode. A separate UART channel will be used to communicate with a separate board so that the 2-player tennis game can be played, this will be used in interrupt mode so that when messages from the slave are completely received a flag can be set.
I^{2}C	100kHz clock speed. Clock and data line have their own pins. These pins will act as bus lines for communication between the IMU and MCU and be used to simulate button presses during the games.

5 Measurements and Results

Measurements to ensure the arcade system and each of its component performs correctly is detailed in this section.

5.1 Power Supply

To ensure the power supply is correct a multimeter is used to measure the voltage during operation. The $5V$ and $3.3V$ nodes were respectively measured to be $5.11V$ and $3.31V$ and varied negligibly during operation.

5.2 UART Communication

Using the USB connection from the MCU, a computer with TeraTerm software is connected allowing UART messages to be displayed live as well as logged with timestamps. The live communication was used to see that the UART was being correctly sent with the correct messages, while the logged files were used to verify the timing of the uart messages and thus flow of the games. The messages were correctly sent as specified in Figure 5, however, it was found that the timing varied by about 10% from the aimed for time of $100ms$, with the actual timing being up to $10ms$ faster or slower. UART messages were also verified with an oscilloscope for timing checking the signal and its frequency, as shown in 16.

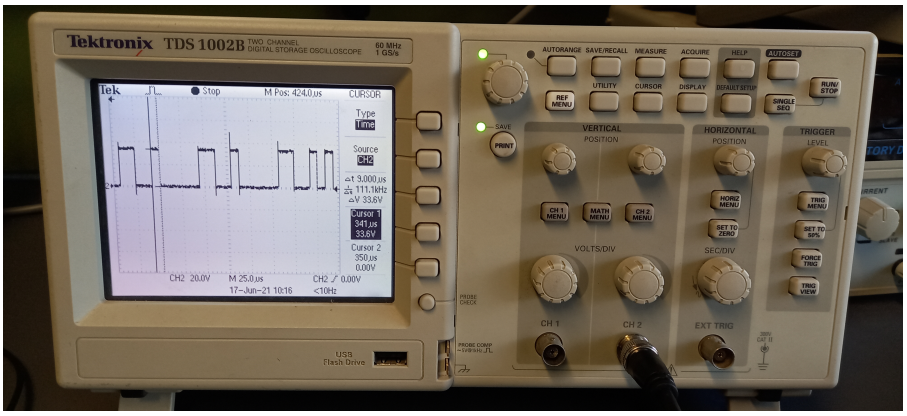

Figure 16: UART Oscilloscope

5.3 Buttons

For each of the buttons a multimeter was used to measure the potential difference of each of the button inputs with respect to ground. It was found as expected that the voltage remained at a high of $3.3V$ when not pressed and dropped to 0 when the button was pressed.

5.4 Dot Matrix

The steps to ensure the dot matrix was developed correctly is discussed. Correct planning ensured the LEDs fit in a 60x60mm box on the Veroboard, a ruler was used to measure the dimensions for final verification. Adding resistors to the LED matrix and using visual cues, the brightness of each of the LEDs is checked when a separate $3.3V$ source is connected to each of the columns and ground is connected to a row. Any change in brightness or unexpected behaviour was debugged checking for correct shorts amongst the soldered components on the Veroboard. The MOSFETs were then connected; $3.3V$ was applied to the gate terminal as well as each of the LEDs in a row and each of the MOSFETs were checked to be working by verifying the corresponding LEDs in the row were turned on. A multimeter was used to measure the voltage across the resistors which should read approximately $180\,\Omega \cdot 8\,\text{mA} = 1.44\,\text{V}$ to ensure the assumption of a Drain-Source voltage drop was negligible and the

current remained around $8mA$. The dot matrix was then connected to the MCU and controlled using 16 GPIO pins as seen in Figure 7. Using STM32CubeIDE software and SysTick interrupt the pins were programmed to output 3.3V to the LEDs at a rate of $125Hz$ with a duty cycle of 12.5%. Using an oscilloscope, the potential difference between the GPIO pin above the R0 and GND is measured when only LED00 is on, followed by the entire matrix being lit up. In both cases the maximum instantaneous potential difference (LED on in duty cycle) remained at $3.3V$. This measurement was repeated for the remaining 7 GPIO pins. Using an oscilloscope, the potential difference over the resistor R0 is measured when the entire matrix is lit up, followed by only having LED00 on. In both cases the maximum instantaneous potential difference was measured to be $1.4V$ for $1ms$ in a period of $8ms$. This measurement was repeated for the remaining 7 resistors. It was found that the LED matrix worked perfectly and there was no ringing, thus gate resistors were not required. Figure 17 shows how the oscilloscope was used to verify the timing of the LEDs and therefore SysTick timer when the dot matrix is in its home configuration screen; as expected a high signal is produced for the first and last millisecond in the period of the first column of the dot matrix.

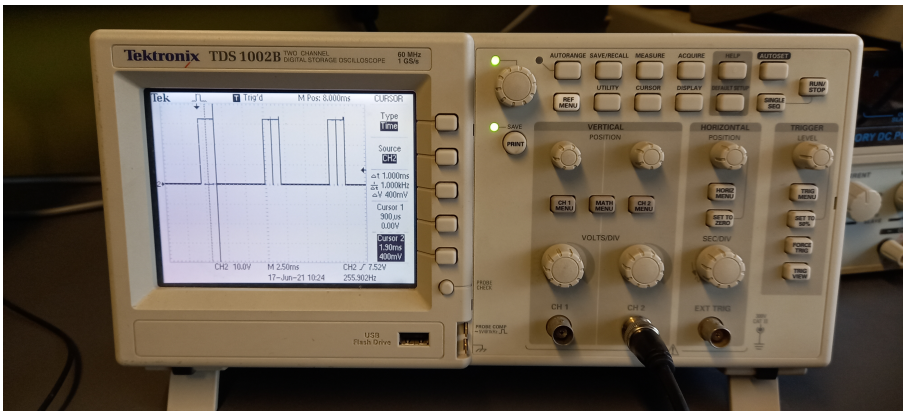

Figure 17: Row 0 LEDs on Oscilloscope during Configuration

5.5 LEDs (Debug)

Through wiring up the Debug LEDs to the MCU and setting the respective pins high, the voltage across each of their resistors was measured and found to be $1.44V$ indicating a current of $\frac{1.44\,\mathrm{V}}{1.5\,\mathrm{k\Omega}} = 0.96\,\mathrm{mA}$ which is very close to the 0.97 mA current designed for.

5.6 Slider

Using a multimeter the potential difference between the pin connected to the output of the slider and ground is measured. As expected, it is found that the pin's output voltage changes between 0 and $3.3V$ as the slider is moved.

5.7 IMU Interface

Using an oscilloscope, the SCL line for the MCU is verified to have a frequency of $100\mathrm{k}Hz$. Using a trigger the signal on the SDA line is checked to verify the signal being sent to the and from the MCU is correct and the same as Figure 15. In testing the actual IMU a read of register address 0x0F was first tested to get the device identity (0x33 [10, Tab. 24]), thus ensuring the MCU was correctly communicating with the IMU. After this the output was checked when the IMU was rotated, confirming the high resolution result was being correctly interpreted from the output registers. Finally, with a protractor in the background the angle of the IMU was changed by $\pm 30°$ about the x and y axis and it was checked that the UART outputted the correct corresponding button presses.

6 Conclusion

From the testing of the various peripherals it is found that the system meets all the requirements as set out in the Introduction. Thus, there are no requirements which were not complied with.

One of the shortcomings was the timing where as previously mentioned UART timing varied by approximately 10% from that designed for. The best solution would be to implement a Real-Time Operating System which would handle the various operations in a more customisable and timely manner. The logic and UART communication involved with the two-player tennis game is detailed in Appendix C, unfortunately, due to limited time and opportunity it was not properly tested.

Overall, this arcade system has been well designed and developed. The complete schematic for the various peripherals is shown in Appendix A while the MCU pinout is tabulated in Appendix B.

References

[1] *STM32 Nucleo-64 Board*, Datasheet, ST Microelectronics, August 2020.

[2] *L7805CV 5V Voltage Regulator*, Datasheet, ST Microelectronics, August 2006.

[3] *Design E314 Baseboard*, Schematic, EE Engineering, Stellenbosch University, 2019.

[4] *MCP1700-3392E/TO 3.3V Voltage Regulator*, Datasheet, Microchip, October 2013.

[5] *Project Definition*, Document, EE Engineering, Stellenbosch University, 2021.

[6] sparkfun. (2021) Pull-up resistors. [Online]. Available: https://learn.sparkfun.com/tutorials/pull-up-resistors/all

[7] *STM32F103RB Microcontroller*, Datasheet, ST Microelectronics, August 2015.

[8] *L-1334IT 3mm cylindrical 3mm LED*, Datasheet, Kingbright, March 2006.

[9] *2N7000 N-Channel Enhancement Mode Field Effect Transistor*, Datasheet, ON Semiconductor, October 2017.

[10] *LIS3DH*, Datasheet, ST Microelectronics, December 2016.

[11] *LIS3DH*, Application Note, ST Microelectronics, January 2011.

A Complete Schematic

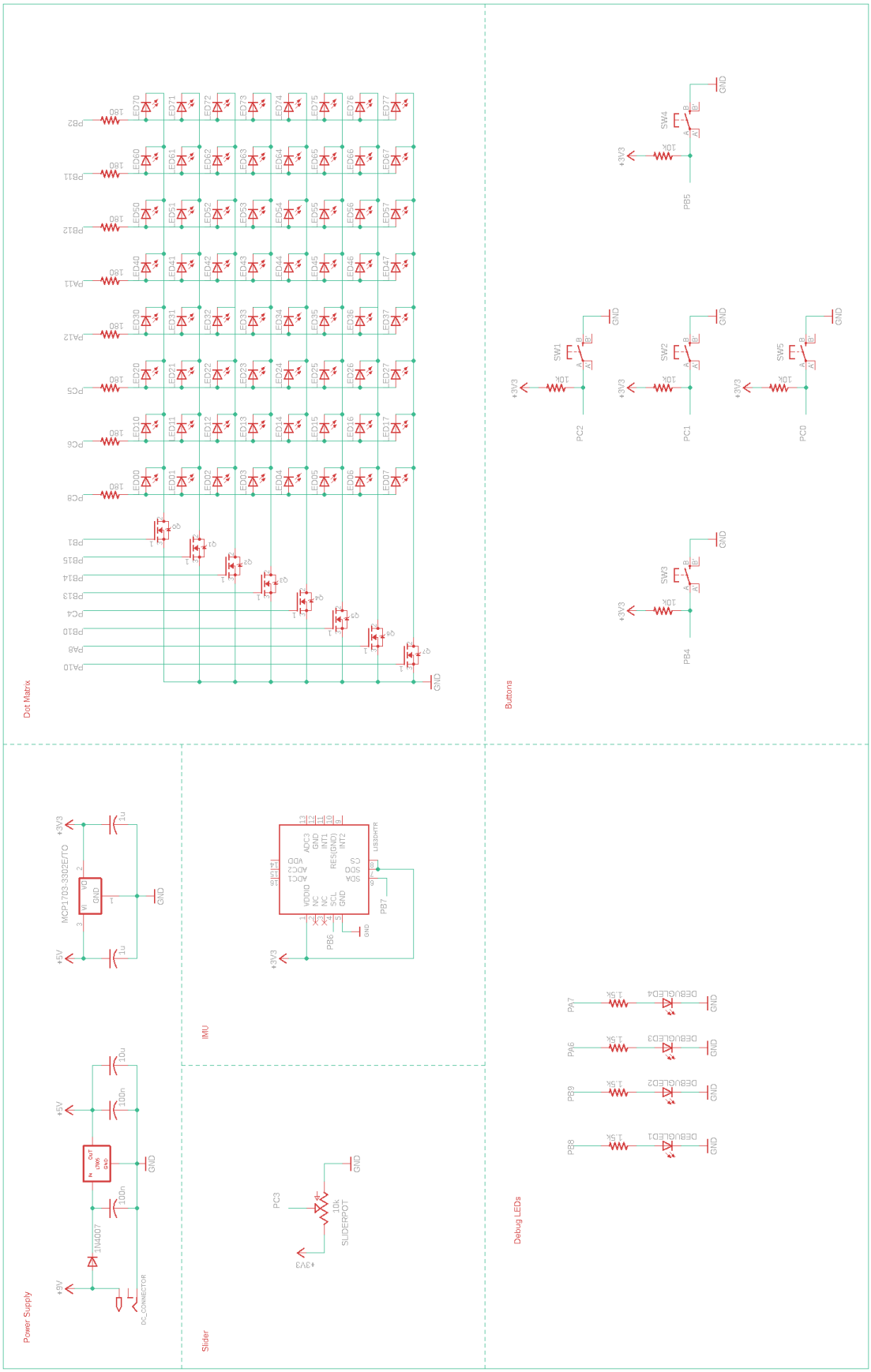

Figure 18: Complete Schematic

B STM32 Pinout

Table 2: STM32 Pinout

PIN	CONFIGURATION	HARDWARE	TIC
PA0			
PA1			
PA2	USART2_TX		J13 - 5,6
PA3	USART2_RX		
PA4			
PA5	LD2*		
PA6	GPIO_Output	Debug LED 3	
PA7	GPIO_Output	Debug LED 4	
PA8	GPIO_Output	LED Row 6	
PA9			
PA10	GPIO_Output	LED Row 7	
PA11	GPIO_Output	LED Column 4	
PA12	GPIO_Output	LED Column 3	
PA13	TMS*		
PA14	TCK*		
PA15			
PB0	GPIO_EXTI0	Down Button	J3 - 13,14
PB1	GPIO_Output	LED Row 0	
PB2	GPIO_Output	LED Column 7	
PB3	SWO*		
PB4	GPIO_EXTI4	Left Button	J13 - 9,10
PB5	GPIO_EXTI5	Right Button	J3 - 11,12
PB6	I2C1_SCL	IMU SCL	J13 - 7,8
PB7	I2C1_SDA	IMU SDA	J3 - 7,8
PB8	GPIO_Output	Debug LED 1	
PB9	GPIO_Output	Debug LED 2	
PB10	GPIO_Output	LED Row 5	
PB11	GPIO_Output	LED Column 6	
PB12	GPIO_Output	LED Column 5	
PB13	GPIO_Output	LED Row 3	
PB14	GPIO_Output	LED Row 2	
PB15	GPIO_Output	LED Row 1	
PC0			
PC1	GPIO_EXTI1	Middle Button	J13 - 13,14
PC2	GPIO_EXTI2	Up Button	J13 - 11,12
PC3	ADC1_IN13	Slider Potentiometer	J13 - 3,4
PC4	GPIO_Output	LED Row 4	
PC5	GPIO_Output	LED Column 2	
PC6	GPIO_Output	LED Column 1	
PC7			
PC8	GPIO_Output	LED Column 0	
PC9			
PC10	USART3_TX	2-Player Tennis Communication	
PC11	USART3_RX	2-Player Tennis Communication	
PC12			
PC13	B1*		
PC14	RSS_OSC32_IN*		
PC15	RSS_OSC32_OUT*		
PD0	RSS_OSC_IN*		
PD1	RSS_OSC_OUT*		
PD2			

*Standard Pins for Nucleo Board.

C 2-Player Tennis Game

The two player tennis game will be the same as the single player except the ball position and direction will be transferred between two boards allowing two players to play against each other. A separate UART communication channel (UART3) will be used to communicate between the boards. The two player tennis game works very similarly to the normal tennis game except it requires a UART receive as shown in Figure 19.

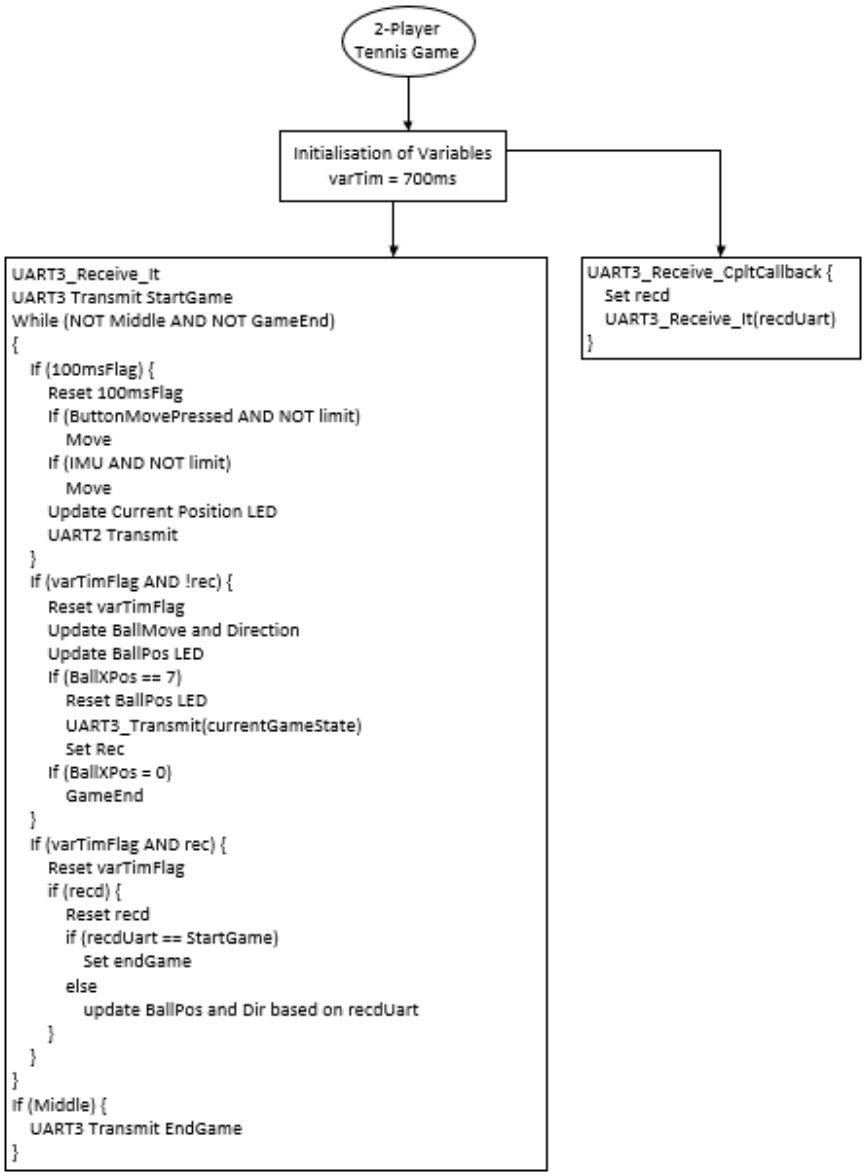

Figure 19: 2-Player Tennis Game Pseudo-code

As shown in Figure 19 the UART is set up on communication channel 3 for receiving and transmitting messages between the boards. This new UART channel is set up in interrupt mode, thus, when a message is received from the other board the complete callback function is called and the recd flag set so that the game can continue on the master board. Our MCU is set up as the master board and thus must send an initial command to the slave board to indicate the start of the game. When the ball crosses the backboard a UART message is sent to the slave board indicating the current state of the game, the master then waits for to receive a UART communication via interrupt to continue the game. The direction and ball position is to be adjusted accordingly to the position and direction sent by the slave.