ELECTRICAL & ELECTRONIC ENGINEERING
STELLENBOSCH UNIVERSITY

DESIGN (E) 314
TECHNICAL REPORT

Dot-Matrix Arcade Game Console

Author:
Rowan Twilley

Student Number:
23105577

June 13, 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.

[illegible]	23105577
Handtekening / *Signature*	Studentenommer / *Student number*
R C Twilley	20/06/2021
Voorletters en van / *Initials and surname*	Datum / *Date*

Contents

List of Figures

List of Abbreviations

IMU - Inertial Measurement Unit
MCU - microcontroller unit
LED – Light Emitting Diode
UART – Universal asynchronous receiver-transmitter
I2C - Inter-integrated circuit
USB – Universal Serial Bus

1 Introduction

The system being built is a classic retro arcade game console. The system will allow the player to play 2 classic arcade games, a simple maze game with a choice of 4 different levels and the other game being classic, “Pong”. The system allows the player to use various input controls for the 2 games.

The requirement is to make a retro arcade console programming a STM32 Nucleo Microcontroller. The Microcontroller forms the heart of the system and will oversee the processing of all the sensor inputs and will output the game on a custom built 8x8 LED matrix which acts as a low-resolution screen.

The report will discuss various aspects of how the project was developed including: The System description, Hardware Design and Implementation, Measurements and Results and Conclusion. The aim of the report is to guide the reader through the design steps taken to achieve the result of the working project both with respect to the hardware and software.

2 System description

Figure 1 below shows the block diagram for the entire system including the interfaces with which they all interact.

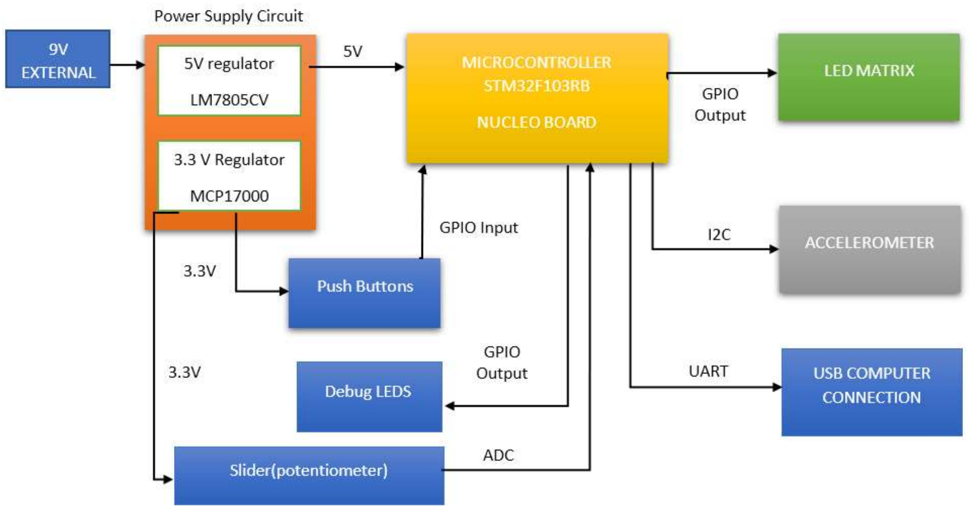


Figure 1 : System ``Block Diagram"

System operation and function

For the project we are required to use the STM32F103RB Nucleo-64 microcontroller. It is based on the ARM Cortex M4 32-bit CPU. It has a clock speed of 72MHz. It has 128kB of flash memory and 20kB of SRAM. It is powered externally by the 5V regulator from the power supply circuit.

The board takes input through the GPIO interface from 5 mechanical push buttons. The microcontroller receives input from an analog slider and uses the Analog to Digital Converter to convert the 0-3.3V voltage to a digital numerical number which can be used.

The microcontroller uses the GPIO Interface in Output mode to control the LED matrix consisting of 64 LEDS in an 8x8 grid to display the results of the game on essentially a low-resolution screen. This is done by using a single GPIO pin for each column and row on the microcontroller, so in total 16 pins are allocated for the LED matrix to allow full functionality.

The accelerometer will provide input to the microcontroller using the I2C communication protocol. This will allow the physical orientation of the baseboard to be used as input to the games played. The accelerometer uses a 3 axis MEMS system to communicate its current position space, using cartesian coordinates to the microcontroller which we can use to interpret angles and subsequently control our games.

The microcontroller outputs the various parameters of the system using the UART communication interface while the games are played. The Test Interface Connector (TIC) reads and validates the results that the system is displaying on the LED matrix.

The 4 debug LEDS are connected through GPIO output to show which maze level the player is currently playing, from left to right the LEDs are numbered from 1 to 4 and represent the current maze level that is being played.

The power supply circuit powers the various components needed for the project.

The project is powered by an external 9V DC source. The supply circuit contains two voltage regulators to convert the external 9V down to 5V, this is done with the LM7805CV voltage regulator, and then we further regulate the voltage down to 3.3 V with the MCP1700 regulator.

Table 1 shows a short description of the components with the respective connecting interfaces and operation voltages.

Table 1: Project Components respective interfaces and operation voltages

Component	Connecting Interface	Operating Voltage
STM32 module	N/A	2.0 V-5.5 V
FX230 UART	UART	3.3 V
LED Matrix	GPIO Output	3.3V
Slider	ADC	0 -3.3V
Debug LED’s	GPIO Output	3.3V/5V
Accelerometer	I2C	3.3V
Push Buttons	GPIO Input	3.3V – 0V

3 Hardware design and implementation

The design and implementation for each section of the project is discussed in this section of the report.

3.1 Power supply

The 5 V and 3.3 V power supply sub-circuits that are used for the project is shown in figure 2 and 3 respectively below.

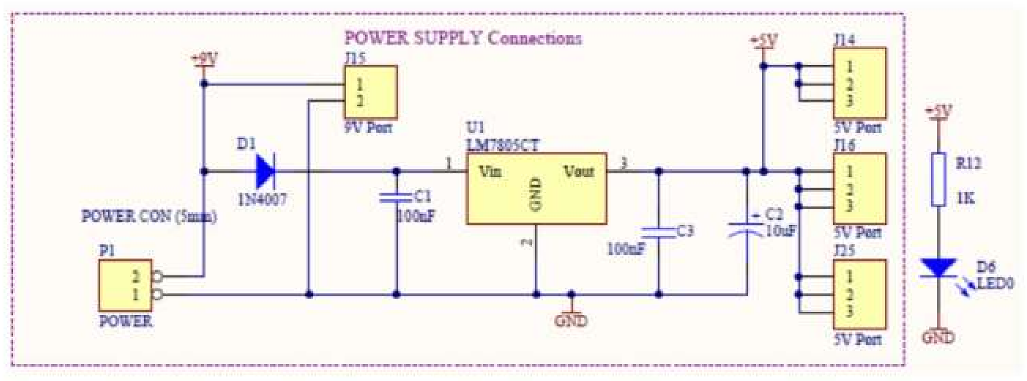


Figure 2 : 5 V power supply sub-circuit

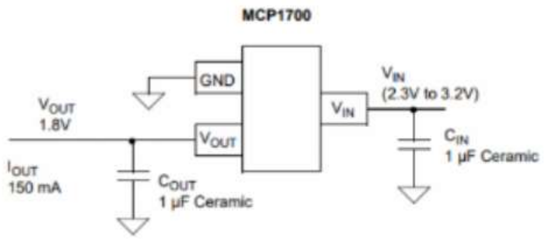


Figure 3 : 3.3 V power supply sub-circuit

The project is powered by an external 9V DC source. The power supply circuit consists of two voltage regulators to regulate the 9V supply first to 5V, this is done using the LM7805CV voltage regulator, and is further regulated to 3.3V, using the MCP1700 voltage regulator. The supply circuit needs additional components such as capacitors and diodes to ensure the circuit can operate properly under changing currents and voltages that it might be subjected to.

Referring to “figure 2: 5 V power supply sub-circuit” we use a protection diode (D1) IN4007 model to ensure that reverse polarity never occurs and allows current to flow in the forward direction only. The capacitors C1, C2, C3 protect the supply circuit from the variations from the 9V battery and ensures a sufficient current will flow once the power is being supplied. C2, C3 are also bypass capacitors which are used to filter out high frequencies and possible noise that could make its way into the system.

The values of C1, C2 and C3 are given on the datasheet and are 10uF, 0.33uF and 0.1uF, respectively.

A green LED to the power supply is connected to indicate that the circuit is in operation. The LED needs a series resistor to prevent it from blowing due to excessive current. The resistor has forward voltage of 2.6 V and can take 18mA maximum current. So, we can design the resistor with these parameters.

$$R = \frac{5-2.6}{18*10e-3} = 133\ \Omega$$

This is the minimum resistance value we can use in the design. So, to be safe I chose to use **R = 1KΩ.** This ensures that the LED will never blow due to overcurrent and is a standard resistor size. Choosing a high resistor value also ensures less power dissipation since less current will flow through the LED.

Referring to “figure 3: 3.3 V power supply sub-circuit”, the datasheet for the MCP1700 gives this circuit diagram a common implementation for the 3.3 V voltage regulator.

The only design choices here were the values for C1 and C2. These capacitors are voltage stability capacitors.

C1(Cin) was chosen to be 1 uF.
C2 (Cout) was chosen to be 100nF.

3.2 UART communications

The UART communication interface serves as a means for testing results of the output on the LED matrix. The microcontroller sends UART messages with information in 8N1 format depending on what game the player is currently playing, this includes information such player position, goal position, goal, and player states (on or off) depending on if the LED is lit up or not. The tennis game sends game information such as ball position, ball velocity, ball direction and bat position.

To test the UART interface, the microcontroller is connected to a PC via a USB cable.

Another chip on the MCU board called the ST link microprocessor is used to convert data to USB format which can be communicated over the USB cable. This microprocessor is connected to the microcontroller pins which is set up to use the UART2 channel on the microcontroller. The output of the RX pin on the ST link microprocessor is connected to the test interface connector.

3.3 LED Matrix

The LED matrix is constructed in a grid of 8x8 3 mm red LEDs.

Some design choices I made when designing the LED matrix:

1. Every row of the matrix has a current limiting resistor of 1 K Ohms.
2. Every column of the LED matrix has a MOSFET. The purpose of connecting the n-type MOSFET transistors to each column of the matrix is to ensure that the LEDs light up with equal brightness by regulating the current flowing through each matrix column by maintaining equal voltage drop across the LEDs.
3. Each row connects the positive terminals of the LED to a GPIO input on the microcontroller, while each column connects the negative terminal of the LED to a GPIO input of the microcontroller.

The microcontroller supplies 3.3V to the LED matrix when the GPIO pin is pulled high.

Setting the GPIO row and column pin high will turn the corresponding LED in that row and column high, i.e. If column 2 and row 5 are both pulled high then the LED in the position [5,2] will light up.

Figure 4 shows the LED matrix design that is implemented in the project.

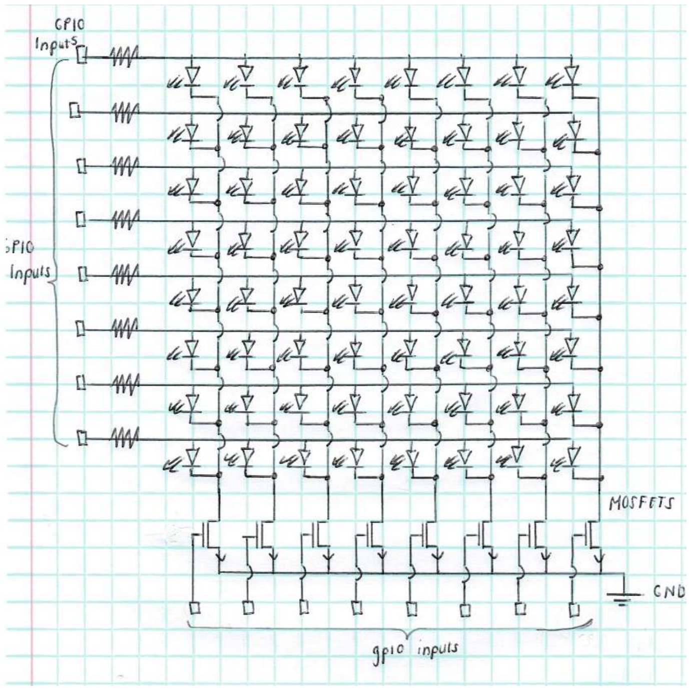


Figure 4 : LED Matrix Design schematic

The 3mm red LEDs used in the matrix have a maximum current rating of 20mA and a forward voltage of 1.8V. The resistor must be chosen to be placed in series with the LEDs to prevent overcurrent and subsequent blowing of the LED. We design the resistor to keep the LED under its maximum ratings.

$$Rmin = \frac{3.3 - 1.8}{20mA} = 75\ \Omega$$

Series Resistor value of 1KΩ was chosen to ensure the safe operation of the LEDs.

The n type MOSFET transistors are needed to enable multiplexing of the LED matrix. Using MOSFET transistors allow the columns of the matrix to connect to ground and this allows the current being produced by the GPIO pins from the 3.3V supply to be sunk to ground instead of being sunk by GPIO pins connected to the microcontroller.

Current sourced by the GPIO

The current sourced from the GPIO will be the same as the current flowing through the LED only when a single Led is on.
The current sourced = 1.41 mA

Current Sunk by the GPIO

The current sunk, is the total current when all the rows are pulled high by the GPIO and a single column is pulled low (making a single column of LEDs turn on). All the current of the rows will flow into a single path, so current sunk to ground is:

$$8 \times 1{,}41\text{mA} = 11.23 \text{ mA}$$

We know that the max current that can be sunk by a single GPIO is 8mA but since the GPIO connected to the columns is connected to the gate of the MOSFET, no current will flow to that GPIO but instead will be sunk to ground.

LED duty cycle

The duty cycle of the LED will be the max time it takes for an LED to turn on and then cycle through all the LEDs in the column and turn the original LED back on.

$$T_{LED} = 1 \text{ ms}$$
$$\text{Duty cycle} = T_{LED} / 8 \text{ LEDs} = (1)/ (8) = 12.5\ \%$$
$$\text{Duty Cycle} = 12.5\ \%$$

3.4 Push Buttons

The mechanical push buttons used in the project are connected to the microcontroller through GPIO input interface. There are 5 buttons in total. The buttons are implemented as “active low”. When the buttons are pressed it shorts to GND, this is registered as a button press.

The nucleo board setup for GPIO inputs to be set up as PULL UP, this negates the need for external pulling up of resistors that would otherwise need to be connected to the 3.3 V external power supply circuit. This allows for lower hardware costs and is a much simpler implementation since extra resistors are not needed.

3.5 Debug LEDs

The debug LEDs serve to indicate which level, 1 to 4 is currently being played. The LEDs are numbered from left to right from 1 to 4. Each maze level corresponds to a LED on the student board. The LEDs need a series resistor to prevent burning out due to overcurrent and is designed the same as for the resistors on the LED matrix. Thus, a resistor value of 1KΩ is used.

3.6 Slider

The slider is used for input control to the bat of the tennis game. The slider requires a 3.3V connection and a 0 V connection to GND. The slider acts as a variable resistor (potentiometer) and will use voltage division to output a scaled voltage between 3.3V and 0V to the microcontroller. The microcontroller uses its built in Analog to Digital interface to convert the voltage from the slider to a 12-bit unsigned number (ranging from 0 -4096) to be used to set the y position of the bat when playing the tennis game.

3.7 IMU (Inertial Measurement Unit)

The IMU used for the project takes advantage of the accelerometer that forms part of the package.

The IMU is a premade component and so no low-level circuitry was needed to add to the design.

The IMU has 4 main pins that interface with the Microcontroller:

- 3.3V pin connects to the external 3.3V power supply provided through the student board.
- The GND pin connects to the student board ground.
- The I2C_SDA pin connects to the PB7 pin.
- The I2C_SCL pin connects to the PB6 pin.
- CS (clock signal) pin on the IMU is connected to 3.3V.
- SDO pin on the IMU is connected to ground.

The IMU is connected to the student board in the prototype area on the bottom right of the board. It connects with 10cm long wires which enables the IMU to be moved separately from the student board when being tested on the test interface station.

4 Software design and implementation

4.1 High Level Description of program

The software developed for the retro arcade game console follows that of a simple state machine. The program has 4 main files for each section of the implementation of the code. This modularity helps segregate sections and breaks the problem into manageable parts – allowing easier debugging.

he 4 main files are stated:

1. UserGeneral.c
2. MazeGame.c
3. TennisGame.c
4. MazeGameSelect.c

All the above c files have associated header files.

The program starts in the main.c which performs the calibration sequence on startup and as soon as the loop is reached in main.c the program hands over all the control logic to the UserGeneral.c.

The userGeneral.c file handles all the control logic and facilitates the state machine to enter the selected game depending on the user inputs.

Simple diagrams of the operation of entering the games are seen in figure 5.

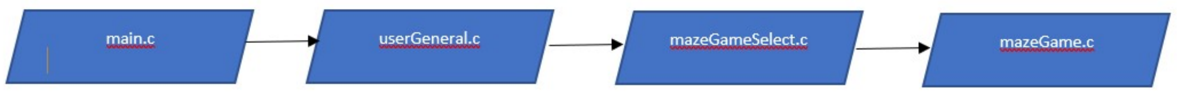


Figure 5 : Flow diagram to play MazeGame

main.c
userGeneral.c
tennisGame.c

Figure 6: Flow diagram of how to play Tennis Game.

The program registers what buttons are pressed when in the main menu screen (4 LEDs on corners) then will set flags based on the button presses and allow the user to either play the Tennis Game if the middle button is pressed.
If the left button is pressed then the MazeSelect options will be presented and when pushing the middle button on the selected number, the game will load that specific maze level.

The program will return to the main menu when the player exits the game using the middle button once in the selected game.
If the player reaches the end Goal in the maze or if the player misses the ball in the Tennis Game, then the game ends and the main menu screen is loaded.

4.2 Control Logic and Data Flow and Processing

The following flow diagram illustrates the control logic of the whole finite state machine implemented in the code.

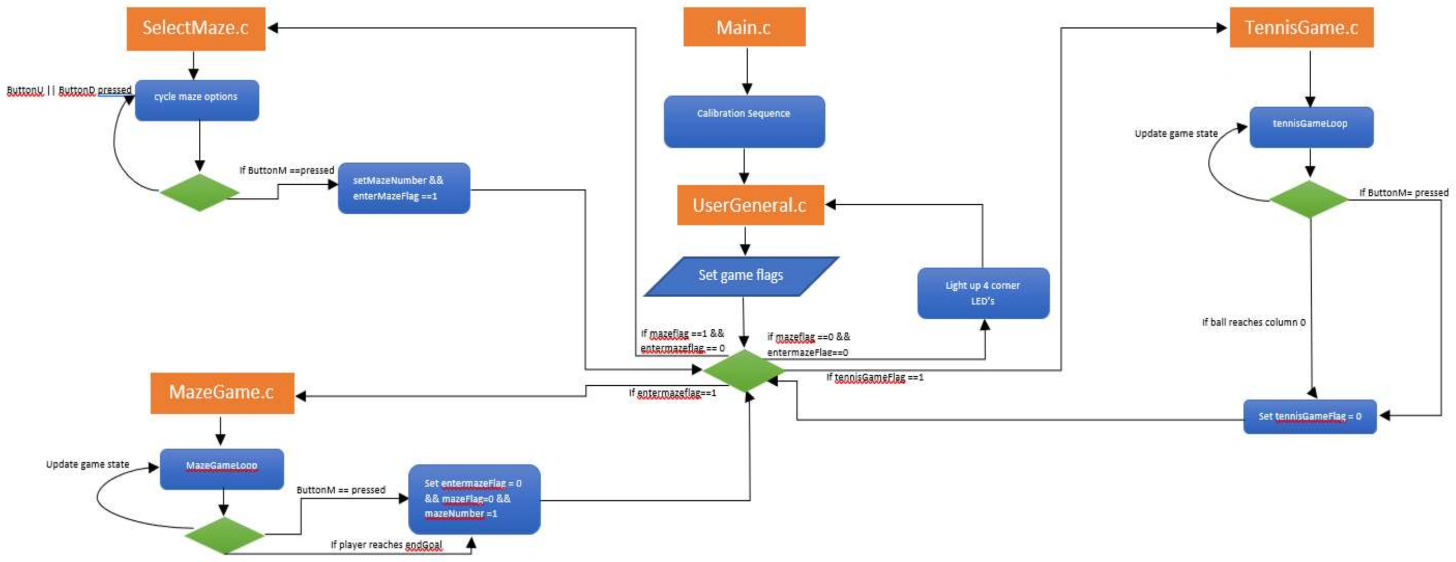


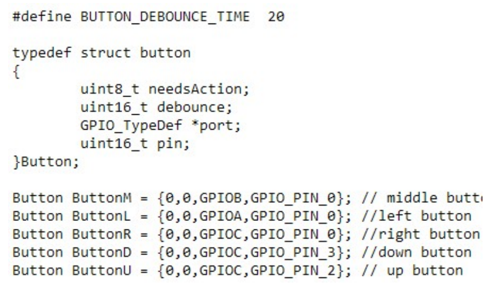
Figure 7 : Control Logic and Data Flow and Processing Diagram

The control logic can be seen from the state diagram in figure 7. The program starts in the main.c file, and first preforms the calibration on startup. The program enters the UserGeneral.c file and uses a function called "mainloop" which is the loop which will run continuously as long a the MCU is operational. The program checks for button presses and debounces the button presses, it then sets flags based on what button was pressed. The program moves into the next function called "state" which will then select the next function that the program should jump to based on the flags that we previously set by the button presses.
If the middle button was pressed then the game jumps to the "tennisGameLoop" and the tennis game will begin. The tennis game loop will run as long as the inTennisGameFlag is still set to a logical high. If the middle button is pressed or the ball moves past the bat and hits the left edge of the screen the

inTennisGameFlag is reset to a logical low and the program know that it should no longer play the tennis game and should default to loading the corner lights. Once the tennisGame ends the program moves back to the “mainloop” and waits for user input again to set new flags. If no input is received, then the program will remain in the “mainloop”.
If the left button is pressed the MazeFlag is set to logical high. The program then checks the flags and sees that it should now move to the “selectMaze” function in the SelectMaze.c file. The “selectMaze” function lights up a number on the display, giving the player the option to select a maze from 1 to 4. When the middle button is pressed on the selected maze number, the number of the selected maze is stored in a variable and passed to the “mazeGameLoop” function in the next step. The “enterMazeFlag” is set to logical high and now the program returns back to the “mainloop”.
The flags “enterMazeGameFlag” is set to logical high, this means the program will now enter the “MazeGameLoop” in the MazeGame.c file. The specific maze will be displayed based on the number of the maze which was passed from the “selectMaze” function previously. The program will continue to play the maze game until the player reaches the endgoal, or until the middle button is pressed. In both of these cases, the “enterMazeFlag” and “mazeFlag” are both set back to a logical 0. The program knows to no longer enter the mazeGame when checking the game “state” function. The 4 corners will light up. The last thing that happens is that the “mazeNumber” variable is set back to 1. This is so that the first selectMaze option will be maze 1, when the player tries to choose a new maze number when pressing the left button again.

4.3 Button Bounce Handling

The button debounce was implemented in the userGeneral.c file and the code seen is credited to [5] Nicol Visser, 16986431. However, all the implementation of the subsequent code, after the button is debounced is my own.



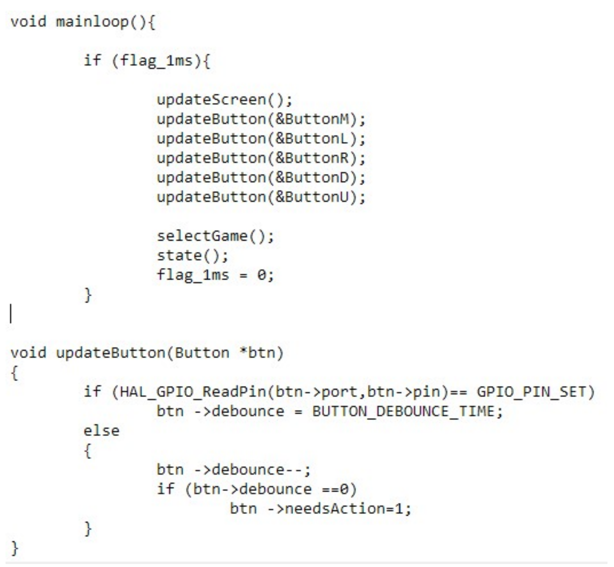
```
#define BUTTON_DEBOUNCE_TIME  20

typedef struct button
{
        uint8_t needsAction;
        uint16_t debounce;
        GPIO_TypeDef *port;
        uint16_t pin;
}Button;

Button ButtonM = {0,0,GPIOB,GPIO_PIN_0}; // middle butt
Button ButtonL = {0,0,GPIOA,GPIO_PIN_0}; //left button
Button ButtonR = {0,0,GPIOC,GPIO_PIN_0}; //right button
Button ButtonD = {0,0,GPIOC,GPIO_PIN_3}; //down button
Button ButtonU = {0,0,GPIOC,GPIO_PIN_2}; // up button
```

Figure 8 : Button debounce struct and declaration.

A new struct was created for the buttons (figure8) with the parameters:

- needsAction – if the button has waited 20ms and is still low, then the button was pressed, the program should now respond to the press.
- Debounce - is just the 20ms needed to debounce the button.
- Port – the letter that the button is connected to through the GPIO.
- Pin – the number of the pin for the specific port that the button is connected to.

Each button was initialized with the parameters according to the specific GPIO pin and port that it was connected to in hardware.



```
void mainloop(){

        if (flag_1ms){

                updateScreen();
                updateButton(&ButtonM);
                updateButton(&ButtonL);
                updateButton(&ButtonR);
                updateButton(&ButtonD);
                updateButton(&ButtonU);

                selectGame();
                state();
                flag_1ms = 0;
        }
|

void updateButton(Button *btn)
{
        if (HAL_GPIO_ReadPin(btn->port,btn->pin)== GPIO_PIN_SET)
                btn ->debounce = BUTTON_DEBOUNCE_TIME;
        else
        {
                btn ->debounce--;
                if (btn->debounce ==0)
                        btn ->needsAction=1;
        }
}
```

Figure 9 : updateButton definition and call statement

The updateButton function is called once every 1 ms in the mainloop of the program. If a specific button is pressed, then the updateButton will wait 20ms for the debounce to settle and then set the needsAction parameter to 1. Depending on where in the program we check for, the needsAction flag will determine what the button press should do.
i.e., If we are in the maze or tennis games and we check ButtonM.needsAction == 1 is true then we will implement the code to exit the game and return to the main menu.

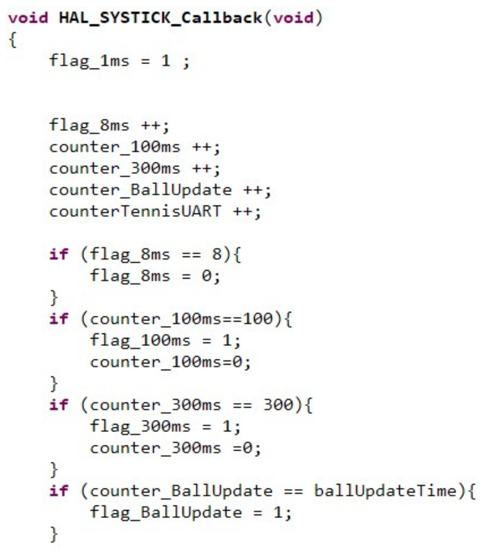
4.4 IMU interface

The IMU uses the 12C communication interface. The IMU is set up such that the SDA and SCL pins on the IMU are connected to PB7 and PB6 respectively. The IMU is set up with the following address and control register values. The address has value of 0x30. The Control Register has value of 0x20. The control register data has a binary sequence of 0b01000011. This allows the acceleration in the x,y, and z directions to be read.
Only the x and y accelerations need to be read. The x accelerations serve as the left and right movement and the y serves as the up and down movement. The IMU needs to experience an angle of at least 30 degrees in order for it to register as a movement command. This is why I have programmed the player movement with the condition of greater or smaller than 31 (accx, accy,accz) before a movement registers. This ensures the condition.

4.5 Peripheral Setup (Timers, ADC, UART, 12C etc)

Timer

The timer used for the software is the HAL Systick Callback Function which runs once every 1 ms and generates an interrupt. This callback function was copied from the Driver files and the weak Keyword had to be removed when copying it into my userGeneral.c file. The timing in the program is achieved using flags that are set in the HAL Systick Callback function.



```
void HAL_SYSTICK_Callback(void)
{
    flag_1ms = 1 ;


    flag_8ms ++;
    counter_100ms ++;
    counter_300ms ++;
    counter_BallUpdate ++;
    counterTennisUART ++;

    if (flag_8ms == 8){
        flag_8ms = 0;
    }
    if (counter_100ms==100){
        flag_100ms = 1;
        counter_100ms=0;
    }
    if (counter_300ms == 300){
        flag_300ms = 1;
        counter_300ms =0;
    }
    if (counter_BallUpdate == ballUpdateTime){
        flag_BallUpdate = 1;
    }
```

Figure 10 : HAL Systick Callback function and how timing is achieved.

We have counter and flag variables that control the time intervals that are required. The 8ms counter and timer is used to do column scanning in the games.

The 100ms and 300 ms counters and flags are used to set the blinking rate of the player and end goal in the maze game.

<u>UART</u>

The UART channel used is USART2 and TX is set to Pin PA2 while the RX is set to RX is set to pin PA3. The USART channel has a baud rate of 115200 Bits/s and is configured in 8N1 mode, with Transmit only. The UART has different messages sent at different timing intervals depending on which game is being played:

Stage	**Timing Interval(ms)**
Calibration Sequence	1000
Maze Game	300
Tennis Game	100

<u>ADC</u>

The analog to digital converter is set to pin PC1 and the channel used is ADC1 IN11. This connects to the slider and converts the sliders position to a digital number that can be interpreted by the program for movement of the player as input. The ADC converts the 0-3,3V voltage from the slider to a 12-bit number. It ranges from 0 -4096 and is used to set the y position of the player during the tennis game.

<u>GPIO Inputs</u>

The push buttons are set up as GPIO inputs:

Button	**Pin**
Middle	B0
Left	A0
Right	C0
Up	C2
Down	C3

The GPIO inputs for the buttons are set up as PULL UP, this means that the PIN will always be at 3.3 V. If the button is pressed, then the pin is shorted to ground, and the program will register this as a button press. The buttons are set to pull up as this negates the need for external pull up resistors.

<u>GPIO Outputs</u>

The 16 pins to control the maze are set up as GPIO outputs and are active low meaning that 3.3 V outputted when the LED must be turned on, and when they are not on, they are at 0V potential.

The figure below shows the GPIO output configuration:

Matrix row/col	**GPIO PORT**	**GPIO PIN**
Row1	GPIOB	GPIO_PIN_5
Row2	GPIOC	GPIO_PIN_5
Row3	GPIOC	GPIO_PIN_6
Row4	GPIOC	GPIO_PIN_8
Row5	GPIOA	GPIO_PIN_12
Row6	GPIOB	GPIO_PIN_12
Row7	GPIOB	GPIO_PIN_11
Row8	GPI0B	GPIO_PIN_2
Col1	GPI0B	GPIO_PIN_1
Col2	GPI0B	GPIO_PIN_15
Col3	GPI0B	GPIO_PIN_14
Col4	GPI0B	GPIO_PIN_13
Col5	GPI0B	GPIO_PIN_4
Col6	GPI0B	GPIO_PIN_10
Col7	GPI0A	GPIO_PIN_10
Col8	GPI0B	GPIO_PIN_3

Figure 11 : GPIO Output Pin Configuration

The debug LED's are also set to GPIO outputs and is only switched on when selecting the maze game, and while playing the selected maze game. It serves a visual indicator or which maze level is currently loaded.

5 Measurements and Results

5.1 Power supply

Method for testing:

The power supply circuit provided a 9V input source and measurements of the output voltage was taken at the 5 V regulator output pin using a multimeter.
To test 3.3V we measure the voltage at the output pin of the 3.3V circuit using a multimeter.

Measured Results:

We measure a voltage of 5.10V at the output of the voltage regulator.

We measure a voltage of 3.28V at the output of the 3.3V circuit.

5.2 UART Communications

Method for testing:

- To test the UART timing we connect the RX pin of the MCU to the oscilloscope and measure the difference in time between the timing intervals and then compare this to the expected results.
- For the Calibration Sequence we expect a timing interval of 1 second between UART transmit messages.
- For the Maze Game we expect UART messages with a timing of 300ms between UART transmits.
- For the Tennis Game we expect UART messages with a timing of 100ms between UART transmits.
- The UART line has an inactive high meaning when nothing is transmitting. It is defaulted to 3.3V and the start bit will be 0. So to measure the timings of the UART message signals we look at difference in time between the vertical lines on the oscilloscope. These vertical lines are where the actual message is transmitted. Since the actual message is sent at a baud rate of 115200 bits/s, the message will be sent in a total of a few microseconds. This means when we look at the message on the oscilloscope scale of 100 or 250 ms per interval, the message will seem to be a straight vertical line dropping from 3.3 V to 0 V.

Measured Results:

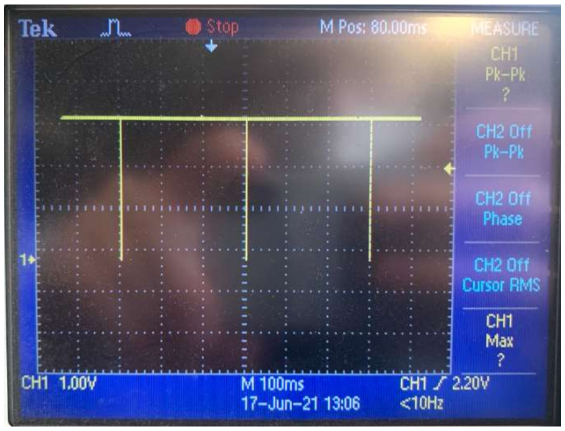
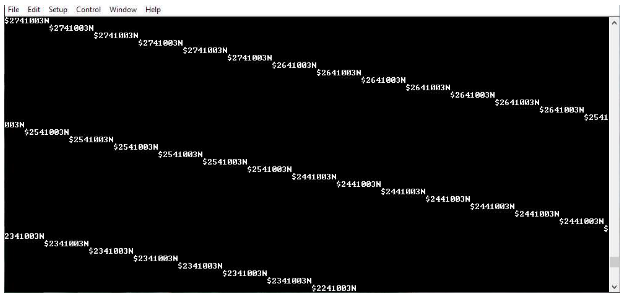


Figure 12: Maze Game UART Timing



```
$2741003N
         $2741003N
                  $2741003N
                           $2741003N
                                    $2741003N
                                             $2741003N
                                                      $2641003N
                                                               $2641003N
                                                                        $2641003N
                                                                                 $2641003N
                                                                                          $2641003N
                                                                                                   $2641003N
                                                                                                            $2641003N
                                                                                                                     $2541
003N
    $2541003N
             $2541003N
                      $2541003N
                               $2541003N
                                        $2541003N
                                                 $2541003N
                                                          $2441003N
                                                                   $2441003N
                                                                            $2441003N
                                                                                     $2441003N
                                                                                              $2441003N
                                                                                                       $2441003N
                                                                                                                $2441003N
                                                                                                                         $
2341003N
        $2341003N
                 $2341003N
                          $2341003N
                                   $2341003N
                                            $2341003N
                                                     $2341003N
                                                              $2241003N
```

Figure 13: UART message Tennis Game

From Figure 11 we see that the timing interval of the maze game UART messages are exactly 300ms apart, by measuring the difference between subsequent vertical lines.

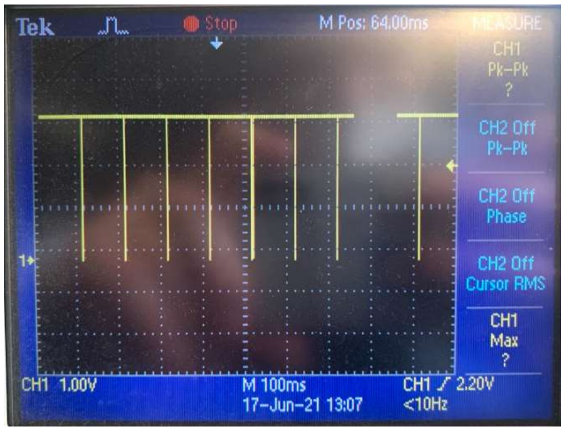
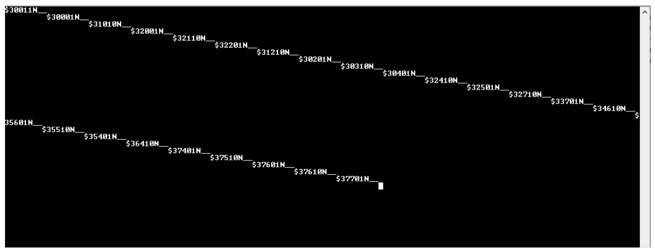


Figure 14: tennis game UART timing



```
$30011N__
         $30001N__
                  $31010N__
                           $32001N__
                                    $32110N__
                                             $32201N__
                                                      $31210N__
                                                               $30201N__
                                                                        $30310N__
                                                                                 $30401N__
                                                                                          $32410N__
                                                                                                   $32501N__
                                                                                                            $32710N__
                                                                                                                     $33701N__
                                                                                                                              $34610N__
                                                                                                                                       $
35601N__
        $35510N__
                 $35401N__
                          $36410N__
                                   $37401N__
                                            $37510N__
                                                     $37601N__
                                                              $37610N__
                                                                       $37701N__
```

Figure 15: UART message for Maze Game

From figure 13 we see that the timing interval of the tennis game UART messages are exactly 100ms apart, from measuring the difference between subsequent vertical lines.

5.3 Buttons

Method for testing:

- The buttons are tested by connecting it to an oscilloscope and measuring an active low (0V) voltage level. This will confirm that the button is being pressed.
- The button bouncing can be seen on an oscilloscope as sudden changes fluctuations in high and low voltage levels at the instant the button is pressed. The time needed for the voltage level to settle to an active low can be calculated. This will be the minimum amount of time the programmer will need to wait in order to ensure that a button was in fact pressed. Without the debounce code, multiple button presses will be registered when in fact the user only presses the button once.
- The button bouncing can also be seen on a rising edge. This is seen in the figure below.

Measured Results:

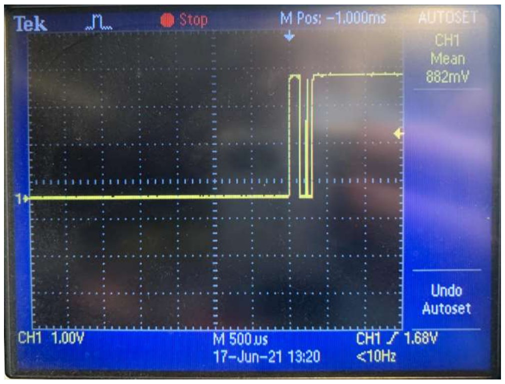


Figure 16: Button bouncing on rising edge.

- The button bounce period measured on the rising edge was measured as 300 microseconds.
- We see a clear 0V when the button is pressed and a clear 3.3V when the button is not pressed this means the button is working as intended.

5.4 LED (debug)

Method for testing:

- The Debug LEDs can be tested by visual inspection. When in the maze select screen the debug LED's 1 to 4 should turn on when selecting the maze 1 to 4 and stay on until the player exits the current maze stage or completes the level.
- The current for the debug LED's is measured with a multimeter.
- The resistor value can be determined from the current and forward voltage.

Measured Results:

- We chose to use a 1 kohm resistor as stated in the hardware specification section.
- The current through the LED is measured to be 1.31 mA by measuring using the multimeter. This is around with what we expected as I = $\frac{3.3-1.7}{1000}$ = 16 mA.

5.5 DOT Matrix

Method for testing:

- The matrix is tested by setting each row and column corresponding to each GPIO high individually and observing if the LED corresponding to the row and column position lights up.
- The led matrix timing is tested by connecting a single column of the matrix to the oscilloscope and measuring the difference between 3.3V pulses when a single row and column is set high. This will give you the duty cycle of the LED matrix when multiplexing the LEDs in the array when using column scanning for the games.
- The current sunk and sourced can also be measured here by using a multimeter to measure the total current sourced by a single GPIO pin to a row in the LED matrix. The current sunk to ground will be all the current flowing when a single column of the LED is turned on. The

current sunk to ground is measured as 11.42mA. This agrees with the theoretical calculations of 11.23mA sunk to ground.

Measured Results:

- We measure a duty cycle of 0.127 ms from the oscilloscope, this is in line with the expected 1/8 ms = 0.125 ms.
- The LEDs all have the same brightness as seen by inspection.
- We measure a current sourced from the GPIO by measuring the current across the 1k ohm resistor as 1.41 mA.
- We measure the current sunk by the ground as 11.23 mA by measuring the current flowing through a column when all the LEDs in a specific column is on.
- The GPIO pin for each column is connected to the base of the MOSFET and so current sunk will be in the order of micro amps.

5.6 Slider

Method for testing the slider:

- The slider is tested by setting the slider to the smallest resistance and measuring the output voltage of the slider. This is done using a multimeter. We expect when there is no resistance that the output pin should allow 0V to be read as there will be no voltage drop across the slider since the slider acts as a potentiometer (voltage divider).
- The slider is again set to the highest possible resistance and then measuring the voltage across the output pin of the slider. A multimeter measurement is taken. When the slider is set to the highest resistance, we expect 3.3V. to be read at the output since there will be 3.3 V drop across the slider and so 3.3V will be read at the output of the slider.

Measured Results:

- Measuring the slider at the lowest point (all the way to left), we measure an output voltage of 0.1V.
- Measuring the slider at the highest point (all the way to right), we measure an output voltage of 3.2 V.

5.7 IMU

Method for testing the IMU:

- The IMU is tested by connecting the oscilloscope to the SDA and CLK data pins and checking that the oscilloscope readings match the expected write line sequences.

Measured Results:

- Unfortunately, no measurements of the IMU were taken, I was unable to gather the testing data before the demo4. The IMU was, however, successfully implemented.

6 Conclusions

From the measured results in the "Measurements and Results" section, it is evident that all the elements of the design comply with the requirements given in the PDD. All the elements of the design work and function well this is seen in the practical part of the project. The software on the MCU executes well and is robust.

6.1 System non – compliance

The system complies with all design criteria as set out in the project description. The system successfully implements all the components of the project: LED matrix, slider, buttons, power supply circuit, IMU, Debug LED's and UART communication. The software design is robust and no after extensive testing no glitches were found. The system thus complies with all design criteria.

6.2 Design shortcomings and Possible Improvements

6.2.1 LED Matrix Positioning

The LED matrix could have been made more compact, moving the adjacent LEDs closer would make the display better looking.
Designing the LED matrix series resistors to be moved to the top of the Veroboard would have allowed the full LED matrix to fit onto the student board without the LED matrix hanging off the edge of the student board.

6.2.2 IMU Connecting Wires

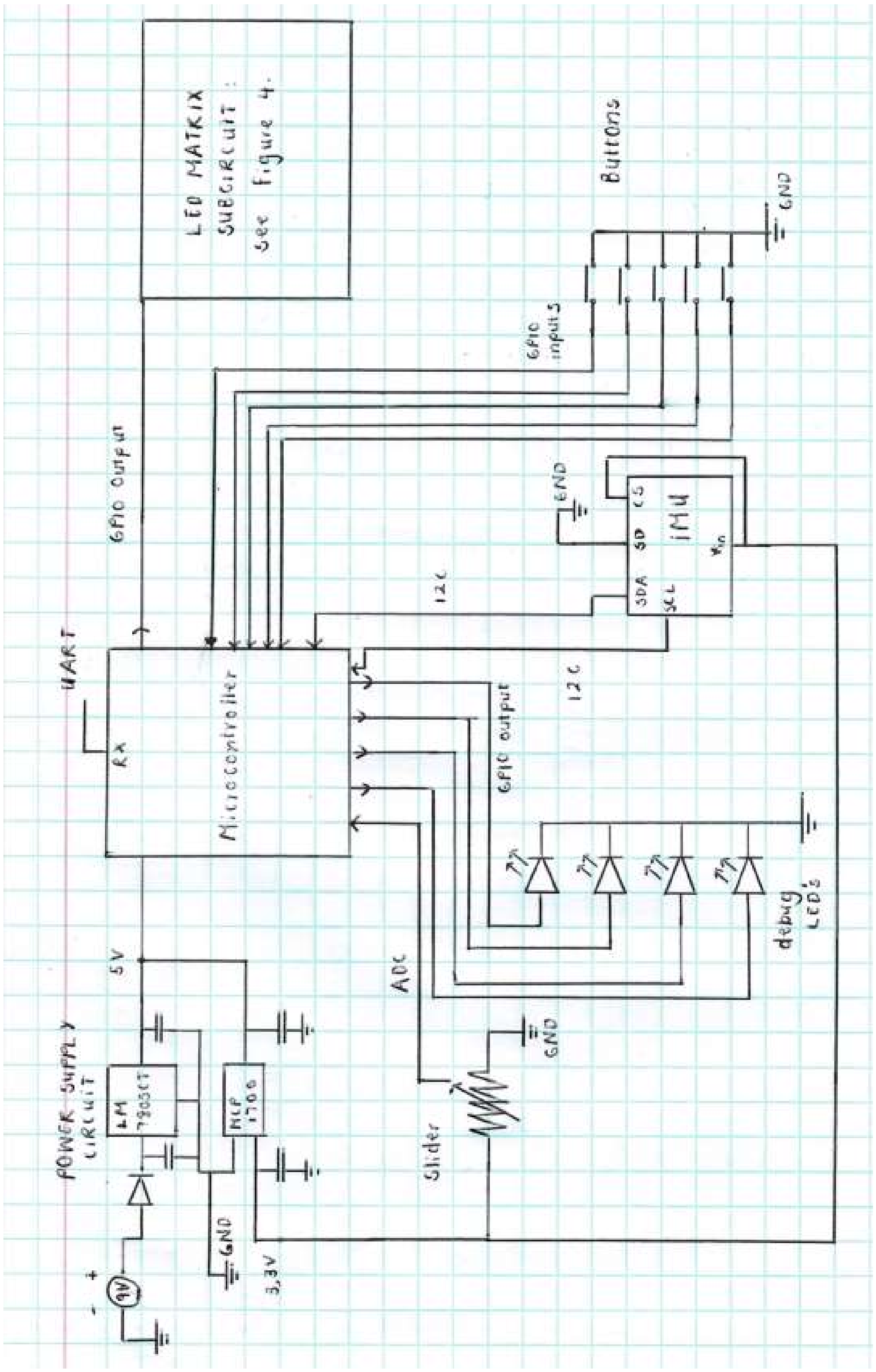
Longer connecting wires to the IMU could have been used, this would have made using the IMU for playing the games easier as it would be more maneuverable. The wire length used to connect the IMU was about 10 cm as this was the longest possible wires, I was able to source. Using wires about 20cm long would have been a better design choice.

6.2.3 Wire management

Wire management on the student board could have been planned and would have made the system look neater and more professional.

7 Appendix

7.1 *Appendix A – full circuit schematic Diagram for the Game Console*



7.2 *Appendix B – full Pinout of the Microcontroller*

PIN	Pin Configuration	Hardware Element
PB5	GPIO_Output	Matrix row 1
PC5	GPIO_Output	Matrix row 2
PC6	GPIO_Output	Matrix row 3
PC8	GPIO_Output	Matrix row 4
PA12	GPIO_Output	Matrix row 5
PB12	GPIO_Output	Matrix row 6
PB11	GPIO_Output	Matrix row 7
PB2	GPIO_Output	Matrix row 8
PB1	GPIO_Output	Matrix column 1
PB15	GPIO_Output	Matrix column 2
PB14	GPIO_Output	Matrix column 3
PB13	GPIO_Output	Matrix column 4
PB4	GPIO_Output	Matrix column 5
PB10	GPIO_Output	Matrix column 6
PA10	GPIO_Output	Matrix column 7
PB3	GPIO_Output	Matrix column 8
PB9	GPIO_Output	Debug LED 1
PA6	GPIO_Output	Debug LED 2
PC7	GPIO_Output	Debug LED 3
PA9	GPIO_Output	Debug LED 4
PA2	USART2_TX	USART2_TX
PA3	USART2_RX	USART2_RX
PB0	GPIO_Input (pull up)	Button Middle
PA0	GPIO_Input (pull up)	Button Left
PC0	GPIO_Input (pull up)	Button Right
PC3	GPIO_Input (pull up)	Button Up
PC2	GPIO_Input (pull up)	Button Down
PB6	I2C1_SCL	IMU
PB7	I2C1_SDA	IMU
PA14	SYS_JTCK-SWCLK	SYS_JTCK-SWCLK (Debug Serial Wire)
PA13	SYS_JTMS-SWDIO	SYS_JTMS-SWDIO
PC1	ADC1_IN11	Slider

8 References

[1] Dr A Barnard, Dr. C Fisher. (2021, June, 19). Design E314 PDD (version 6) [PDF]. Available: https://learn.sun.ac.za/pluginfile.php/2860111/mod_resource/content/0/DesignE314_2021_PDD%20%284%29.pdf

[2] Dr G Brown, (2016,June,5).Discovering the STM32 Microcontroller (version 1) [PDF]. Available : https://learn.sun.ac.za/pluginfile.php/2624659/mod_resource/content/1/Discovering%20the%20STM32%20Microcontroller.pdf

[3] Microchip Technology INC. (2005-2013), MCP1700 Data Sheet [PDF]. Available: https://learn.sun.ac.za/pluginfile.php/2791334/mod_resource/content/0/MCP1700.pdf

[4] Dr G Brown, (2016,June,5).Discovering the STM32 Microcontroller (version 1) [PDF]. Available : https://learn.sun.ac.za/pluginfile.php/2624659/mod_resource/content/1/Discovering%20the%20STM32%20Microcontroller.pdf

[5] Nicol Visser, student number: 16986431, button debouncing code implementation.

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

[7] *FT230X USB to Basic UART IC*, FT230X Datasheet, Future Technology Devices International Ltd, 2016.