Design (E) 314
Technical Report

Dot-Matrix Arcade Game Console

Author:
BJ Edwards

Student Number:
22738002

June 20, 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*

22738002

Studentenommer / *Student number*

BJ Edwards

Voorletters en van / *Initials and surname*

20/06/2021

Datum / *Date*

Abstract

This report will describe the design, construction, programming and testing of a dot-matrix arcade game console.

Contents

List of Figures

List of Tables

Listings

List of Abbreviations and Acronyms

ADC analog-to-digital converter

GPIO general-purpose input/output

I2C inter-integrated circuit

IMU inertial measurement unit

LED light emitting diode

MOSFET metal–oxide–semiconductor field-effect transistor

SCL serial clock

SDA serial data

UART universal asynchronous receiver/transmitter

1 Introduction

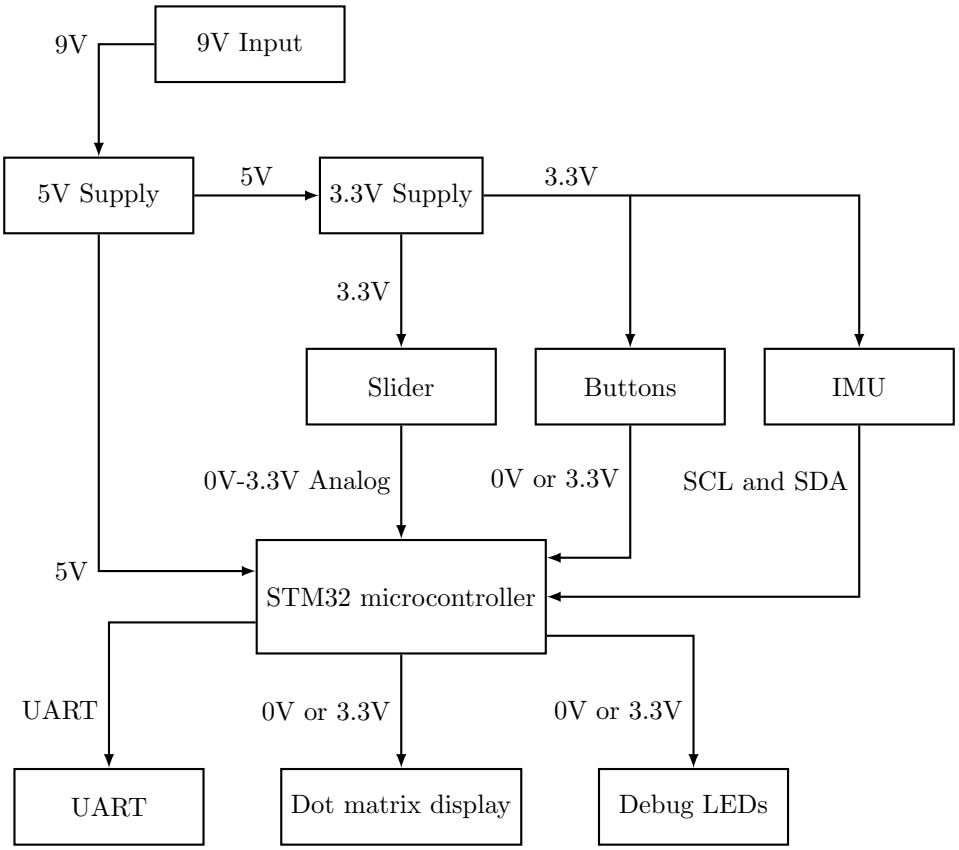
The required system should represent a dot-matrix game console. It should allow the user/'player' either solve a player-selected maze or play a tennis game. The user should be able to interact with the system through buttons, a slider, and an accelerometer. The system should have a visual output through a display and it should output the current state of the system over a communication channel. The system should be able to receive power from a consumer battery.

This report will describe the hardware and software design and implementation of the required system.

2 System description

The system is provided with a 9 V DC input that is fed to a 5 V DC power supply. The 5 V power supply delivers 5 V to the microcontroller (NUCLEO-F103RB) and 5 V to a 3.3 V power supply. The 3.3 V power supply is used to provide 3.3 V to the slider, buttons and the IMU. The slider is connected to microcontroller and provides an analog voltage of 0 V to 3.3 V. The buttons are connected to the microcontroller and provide 0 V or 3.3 V to the microcontroller. The buttons are wired to be active-low. The IMU is connected to the microcontroller through I2C communication. The I2C communication is made up of a SDA and SCL line. The microcontroller is connected to the dot matrix display as an output and it outputs either 0 V or 3.3 V to the display. The microcontroller is connected to the debug LEDs as an output and it outputs either 0 V or 3.3 V to the debug LEDs. The microcontroller uses UART to output information regarding the state of the system. A system block diagram is shown in Figure 1.

Figure 1: System block diagram



3 Hardware design and implementation

This section provides a description of hardware elements and their implementation, calculations for components and motivation for specific design choices.

3.1 Power supply

A constant supply of 5 V and 3.3 V is needed for the system.

For the 5 V supply, an L7805 regulator has been chosen. A schematic is shown in Figure 2. The L7805 regulator will regulate the 9 V input down to 5 V. The 9 V input will be provided at J15. The regulator will be placed at U1. An input capacitor C1 (100 nF) will be used to smooth out interruptions to the supply and low frequency distortions. Two output capacitors C2 (10 µF) and C3 (100 nF) will be used to help provide a clean output to the connected circuit. A 1N4007 diode is chosen for D1. The diode will help protect the circuit if the polarity of the input voltage is reversed. An LED will be placed at D6 to indicate that the power supply is working.

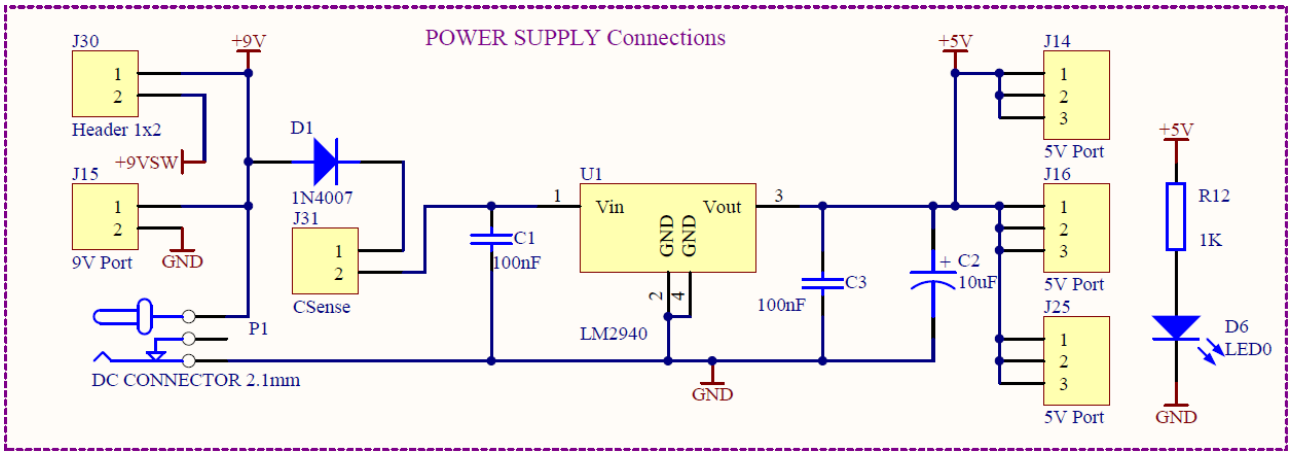


Figure 2: Power supply connections

For the 3.3V power supply, a MCP1700 regulator has been chosen.Two 1 µF ceramic capacitors have been chosen for the input and output capacitors, as shown in the data sheet[1]. Figure 3 shows the schematic for the 3.3 V supply.

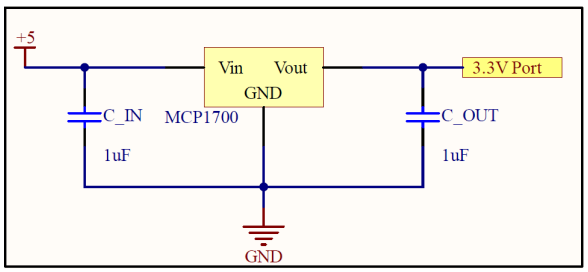


Figure 3: MCP1700 schematic

3.2 UART communications

The default UART2 channel is used. The channel is connected to the ST-Link chip and will automatically show up as a virtual COM port when the microcontroller is connected to a PC through the

USB debugging cable. Therefore, no connections is needed for UART debugging. For normal UART output, the output is taken from the pin labelled 'RX' on the ST-Link.

3.3 Buttons

The 5 push-buttons will be wired up as active-low as shown in Figure 4. The GPIO pin will read a voltage-high when the button is not pressed and a voltage-low when the button is pressed. A 10 kΩ resistor is used for each button to limit the current through the button in order to protect the regulator circuit. If the resistor is not present, the 3.3 V source will short to ground.

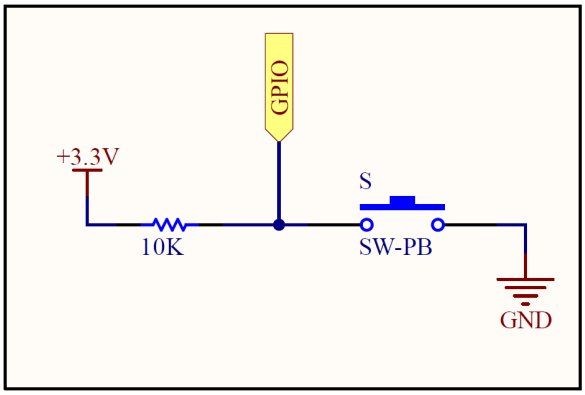


Figure 4: Button schematic

3.4 Debug LEDs

For the 4 debug LEDs, use 3 mm red LEDs. The LEDs are connected to GPIO outputs. The measured forward voltage of the LEDs is 1.7 V. The GPIO pins can source a maximum current of 8 mA each. A maximum current of 5 mA is chosen for each LED.

Resistor calculations:

$$V_{source} = V_R + V_D$$

$$R = \frac{V_R}{I_D}$$

$$R = \frac{V_{source} - V_D}{I_D}$$
$$R = \frac{3.3 - 1.7}{5 \times 10^{-3}}$$
$$R = 320\,\Omega$$

$$\therefore R = 330\,\Omega$$

3.5 Dot matrix display

The dot matrix display is built by creating an 8 by 8 matrix of LEDs. Figure 5 shows a schematic of the dot matrix display. Each row is sourced through a GPIO pin and sunk to ground through a MOSFET

(Q1 to Q8). For the MOSFETs, 2N7000 N-channel, enhancement-mode MOSFETs are used. The MOSFETs are used as switches for the columns. By having the MOSFETs sink the current and not GPIO pins, the pins are protected from potentially sinking too much current and being damaged in the process [2]. The MOSFETs have a maximum turn-on time and turn-off time of 10 ns, which is fast enough to use the LEDs for display applications. The maximum gate threshold voltage of the MOSFETs is 3 V for $V_{DS} = V_{GS}$, $I_D = 1\,\text{mA}$ and the GPIO pins have an output voltage of 3.3 V, which means that the MOSFETs will be switched on. The MOSFETs have a low on-resistance, which means that the brightness of the LEDs will not visibly appear to be affected by the MOSFETs [3].

The drain-source on-voltage of the MOSFETs has a typical value of 0.14 V when $V_{GS} = 4.5\,\text{V}$ and $I_D = 75\,\text{mA}$ [3]. Therefore, V_{DS} can be ignored when calculating the resistor values for the dot matrix. The same calculations from Section 3.4 can be used to calculate the resistor (R1 to R8) values for the dot matrix. Therefore, use 330 Ω resistors.

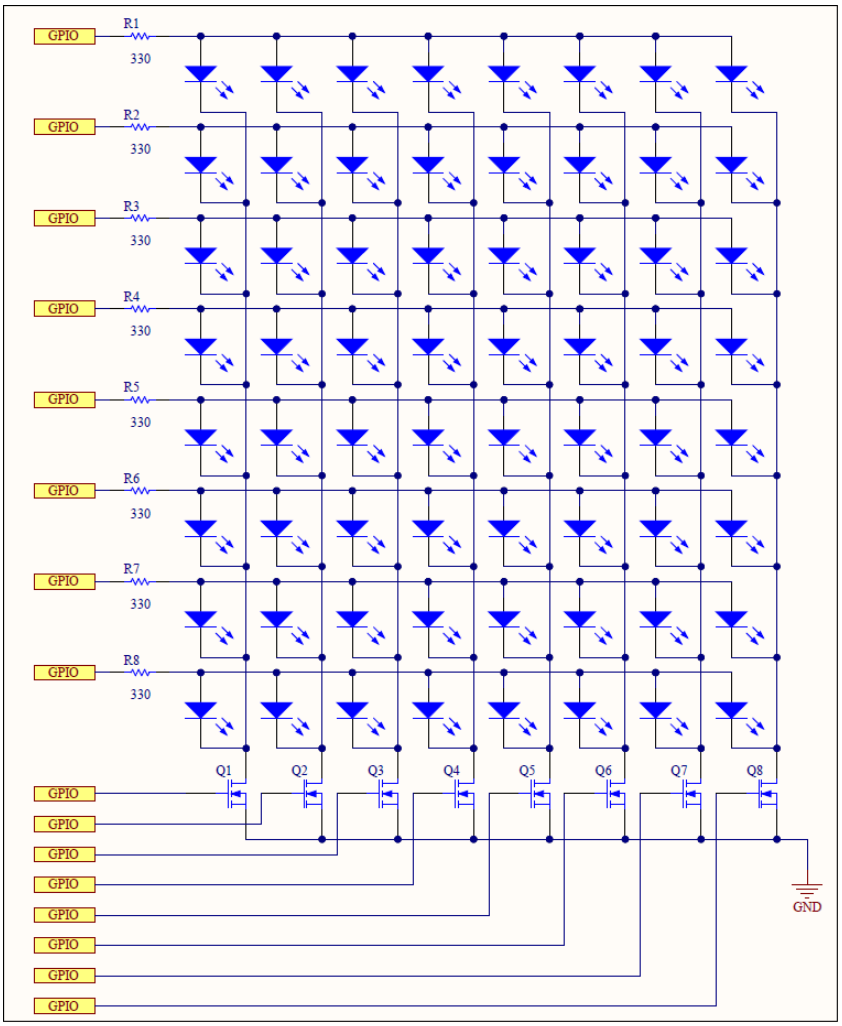


Figure 5: Dot matrix schematic

3.6 Slider

A low profile slide potentiometer is used for the slider. It is wired up as shown in Figure 6 to be a voltage divider [4]. Pin 1 receives an input of 3.3 V, pin 2 is the output of the slider and pin 3 is connected to ground. The slider provides an analog input to the microcontroller. The input is converted through ADC.

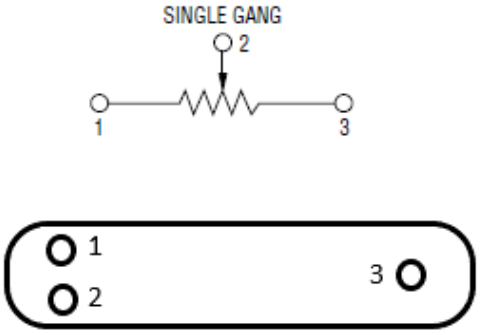


Figure 6: Slider schematic

3.7 IMU

For the IMU, the LIS3DH accelerometer has been chosen. The IMU communicates to the microcontroller through I2C. Table 1 shows the pin connections that have to be made for the IMU. Setting CS to logic-high, I2C communication is enabled [5].

The SDO/SA0 pad is used to modify the less significant bit of the device address (IMU address). If it is connected to the voltage supply, the least significant bit is '1' (address 0011001b). If the pad is connected to ground, the least significant bit is '0' (address 0011000b). The address is completed with a Read/Write bit. [5]. Connect the pad to ground to set the device address to '0011000b'.

IMU pin	Connected to
GND	0 V
VIN	3.3 V
SDO/SA0	0 V
CS	3.3 V
SCL	Microcontroller SCL
SDA	Microcontroller SDA

Table 1: IMU pin connections

4 Software design and implementation

This section provides a description of software design and implementation. It covers the high-level description of the program, its control logic and the, data flow and processing, and the setup of and interaction with peripherals.

4.1 High-level description of program

The program for the system is designed as a finite state machine. It consists of five different states. The state of the machine is changed either by user input or the state changing to a pre-defined different state. There is one state for calibration, one idle state, one selection state and two game states. The program starts by running a calibration sequence and then it enters an idle state where it waits for user input. The user can then either select to play a tennis game or the user can select a maze to solve. A selection state is entered to select a maze before the maze game state is entered. If the tennis

game is lost or the user quits, then the state is changed back to the idle state. If the maze is solved or the user quits, then the state is changed back to the idle state.

Figure 7 shows a state diagram of the system with the actions that cause state changes. Table 2 provides a description of each state. The states are represented in the software by using the enumerate data type.

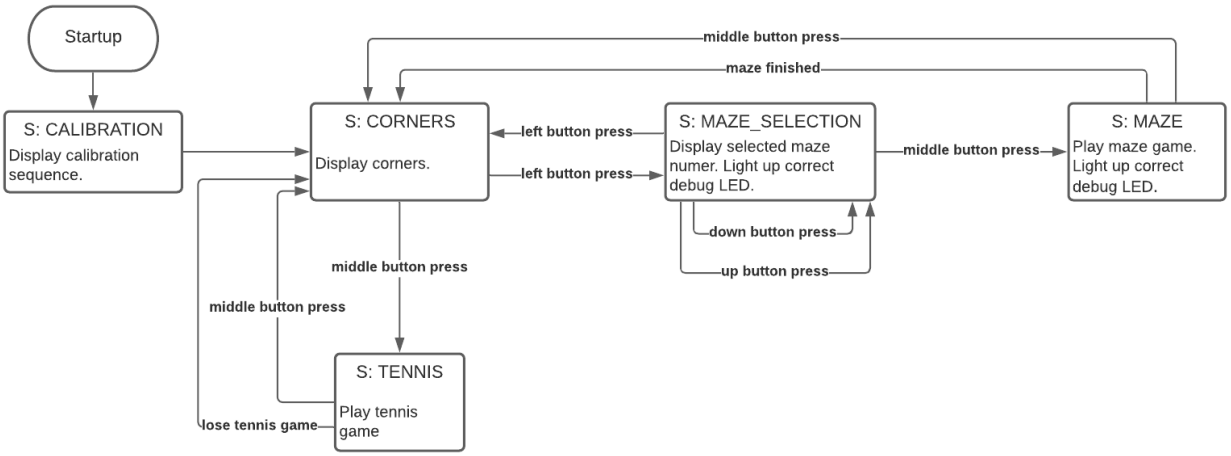


Figure 7: State diagram of system

State	Description
CALIBRATION	Calibration sequence
CORNERS	Display matrix corners
TENNIS	Play tennis game
MAZE_SELECTION	Select a maze to play
MAZE	Play maze game

Table 2: State description

4.2 Control logic

Figure 8 shows a diagram of the control logic used in the software.

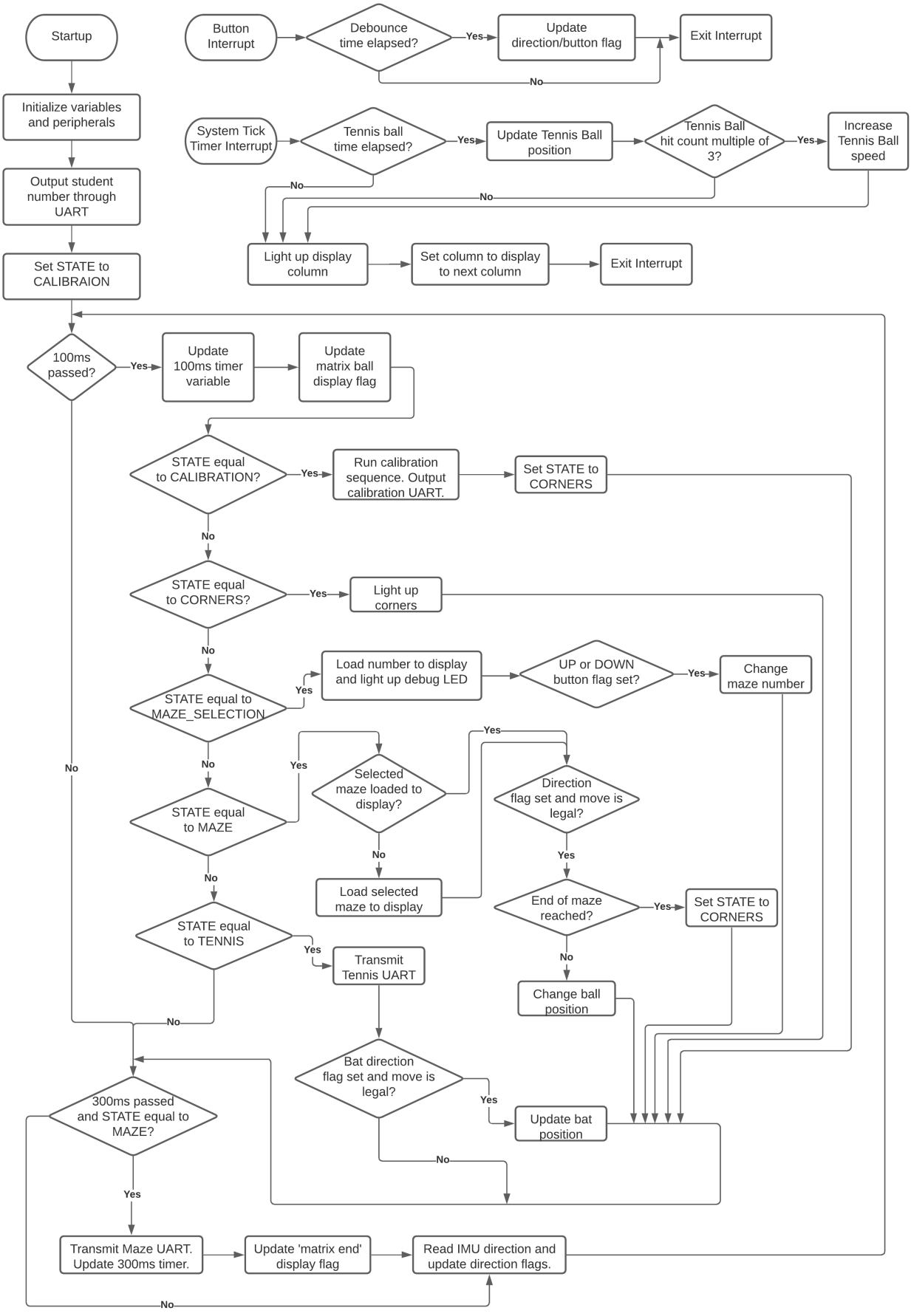


Figure 8: Logic diagram of system

4.3 Button bounce handling

The hardware measurement for button bouncing can be found in Section 5.3. From the measurements it is clear that a delay of 20 ms can be used in software to handle button debouncing. Button debouncing is therefore achieved by ignoring all the inputs from a button for 20 ms after the first input from the button is received.

4.4 Data flow and processing

All of the global variables are declared and initialized in a source file called *matrix.c*. The variables are then included in the other source files by using the *extern* keyword. The variables that can change at any time, such as the flags for the button presses, are declared with the *volatile* keyword.

The function prototypes for the *matrix* source file is contained in a header file *matrix.h*.

The *matrix* file contains the information for the display of the mazes and the numbers when selecting the mazes. A two-dimensional array is used to represent the state of the LEDs in the dot matrix display (the display array). Whenever a the display is completely changed, the array containing the information for the specific display is copied into the display array. If only a small amount of changes occurs in the display, such as the tennis ball moving, the display array is directly edited.

Displaying information on the dot matrix is achieved through column scanning. This is done in the System Tick Timer Interrupt function. The interrupt is triggered every millisecond. Inside the interrupt handler, a function is called that displays one of the columns of the display array and increments sets the column to display to the next column. The next column is then displayed when the interrupt is called again. The display function therefore loops through all the columns every 8 ms.

The bat in the tennis game and the ball in the maze game is only allowed to move every 100 ms. When a button is pressed, a corresponding external interrupt is triggered. A flag for that button is then set to '1'. If the bat in the tennis game or the ball in the maze is allowed to move (e.g. enough time has passed since the last move), then the bat/ball is moved in the direction of the pressed button and the button flag is set to '0'. If the bat/ball is not allowed to move, the button flag is set to '0'. This means that all button presses that occur when no moves are allowed, are ignored. If the bat/ball is allowed to move, the current direction of the IMU is also checked. If the direction of the IMU is a valid direction, then the bat/ball is moved in that direction. A valid direction is a direction that does not take the bat/ball out of bounds and where the ball is not blocked by a maze wall.

4.5 IMU interface

After the IMU boot procedure is completed, the device enters power-down mode. To turn on the device to gather acceleration data, it is necessary to select an operating mode through CTRL_REG1 and enable the axes. Normal power mode (100 Hz) is selected and all three axes (X, Y and Z) are enabled. Therefore, 0x57 needs to be written to register address 0x20 [6].

The IMU acceleration data is represented as 16-bit numbers and are left justified. The measured acceleration data is sent out to registers that respectively contain the most significant part or the least significant part of the acceleration data of the 3 axes (X, Y and Z). The complete data for each axis is given by the concatenation of the most significant and the least significant part and is represented as a 2's compliment number. For the described setup of the IMU, the acceleration data for 1g is represented as 16384_{10} [6].

Figure 9 shows a diagram of the IMU when it is tilted. If the angle θ is between 30° and 90°, then it

should be seen as a directional input. This is done by reading the acceleration data of the X-axis and the Y-axis. If the absolute value of the the acceleration data for an axis is higher than 8192_{10}, then the IMU is tilted at 30° or more with respect to the axis. The sign of the acceleration data determines the signage of the direction of the axis.

$$\begin{aligned} \sin(\theta) &= \frac{x}{16384} \\ \theta &= 30^\circ \\ \frac{x}{16384} &= \frac{1}{2} \\ \therefore x &= 8192 \end{aligned}$$

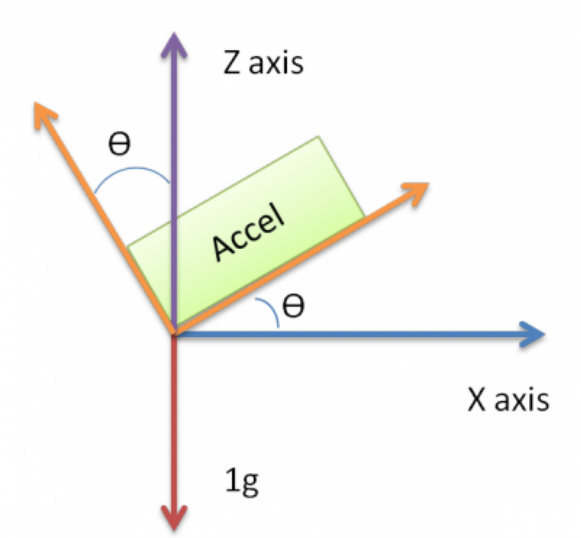


Figure 9: IMU tilt diagram

4.6 Peripheral setup

4.6.1 GPIO

The GPIO pins used are initialised in *main.c* under MX_GPIO_init(). The GPIO output pins are set to have an low initial output level, output push pull mode and no pull-up/pull-down. The GPIO input pins are set up as external interrupts with falling edge trigger detection.

4.6.2 UART

USART2 is set up to operate with a baud rate of 115200 Bits/s, have a word length of 8 Bits, no Parity Bit and 1 Stop Bit. It is set up in *main.c* under MX_USART1_Init().

4.6.3 ADC

The pin and mode of the ADC is configured in *main.c* under MX_ADC1_Init(). Listing 1 shows how the analog value of the slider is read through ADC and how the tennis bat position is updated.

Listing 1: Reading the ADC value

```
// Read the value of the slider through ADC and
// update the bat y position
void read_slider() {
    HAL_ADC_Start(&hadc1);
    HAL_ADC_PollForConversion(&hadc1, 200);
    uint32_t slider_adc_val = HAL_ADC_GetValue(&hadc1);
    HAL_ADC_Stop(&hadc1);
    bat_y_pos = (uint8_t) ( 6 - slider_adc_val / 585.14 );
}
```

4.6.4 I2C

The I2C is set up to have a clock speed of 100 Hz, with a 7-bit primary address length selection. Listing 2 shows the function that is used to set up CTRL_REG1 of the IMU and it shows the function that is used to read the measured data from the IMU.

Listing 2: IMU setup and reading

```
// Set up the IMU with a speed of 100Hz and and enable all
// three axes.
void imu_setup() {
    i2cData[0] = 0x20;
    i2cData[1] = 0x57;
    res = HAL_I2C_Master_Transmit(&hi2c1, 0x30, i2cData, 2, 10);
}

// Read the IMU axes values. Store data in global variables.
void read_imu() {
    i2cData[0] = 0xA8;
    res = HAL_I2C_Master_Transmit(&hi2c1, 0x30, i2cData, 1, 10);
    res = HAL_I2C_Master_Receive(&hi2c1, 0x30, i2cData, 4, 10);
    ax = *((int16_t*)i2cData);
    ay = *((int16_t*)(i2cData+2));
}
```

5 Testing

This section covers the measurements of the built hardware and the methods used to verify the functionality of the hardware.

5.1 Power supply

The 5 V and the 3.3 V power supplies were tested by varying the input voltage to the system and measuring the output of each power supply with a multimeter. The measured output is shown in Figure 10. This shows that both power supplies for the system remain constant at their desired values for input voltages above 7.2 V and below 14 V. The input voltage was not taken to the maximum input voltage [7].

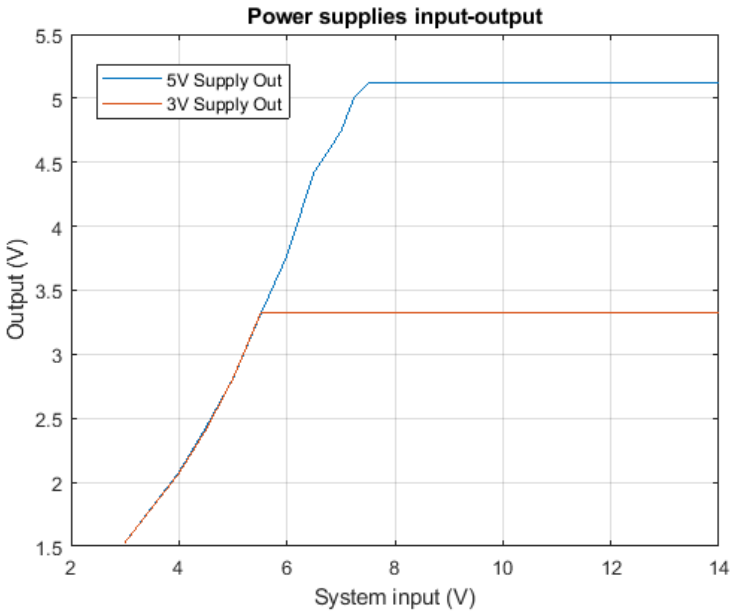


Figure 10: Power supplies measured output

5.2 UART communication

The UART communication is measured with an oscilloscope and through Tera Term software. Figure 11 shows the output of the UART channel when the student number is transmitted after the system starts. The measured baud rate corresponds with the designed baud rate. Figure 12 shows the Tera Term screen when the system is running. It is clear that the UART communication is working as designed.

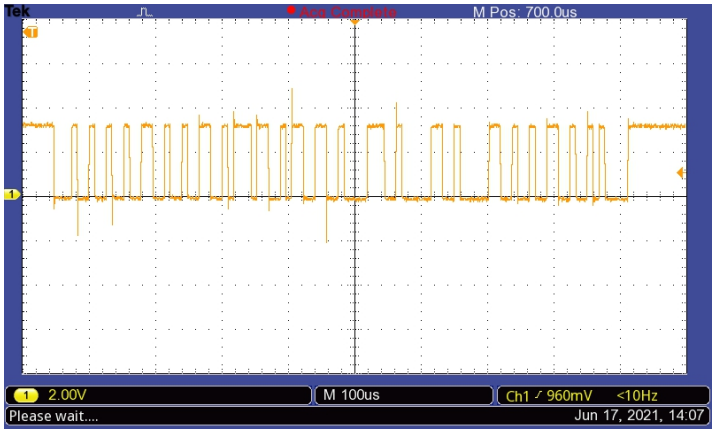


Figure 11: Student number UART transmission

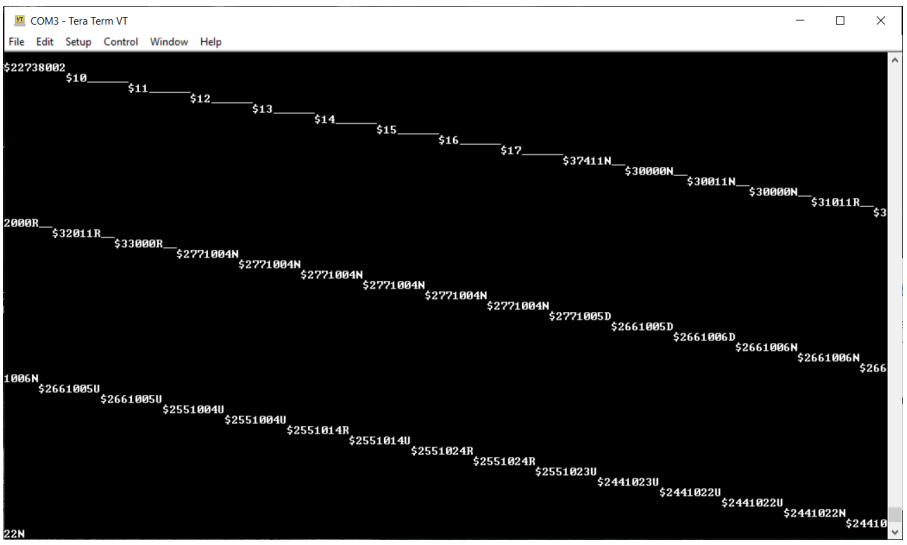
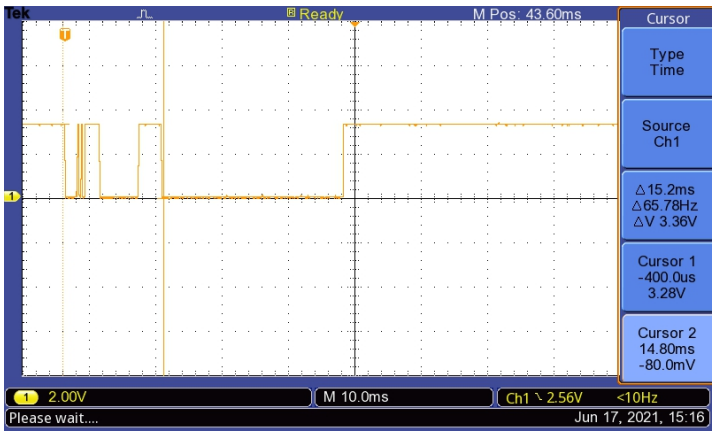


Figure 12: Tera Term Output

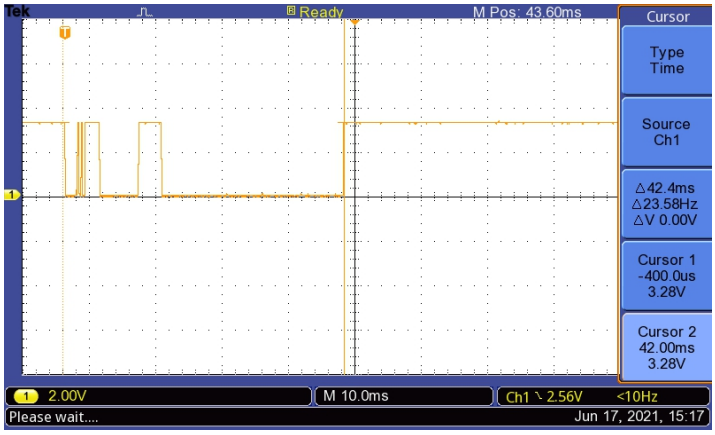
5.3 Buttons

The buttons are tested by clicking them fast and measuring the output with an oscilloscope. Figure 13 shows the oscilloscope measurements. The figure shows that the button bounces for 15.2 ms and the button stays pressed for 42 ms. A time delay for button debouncing in software of 20 ms should therefore be sufficient.

From the oscilloscope measurements in the figure it is clear that the buttons output logic-high when they are not pressed and logic-low when they are pressed. They are thus active-low, as was designed.



(a) Measure till bouncing complete



(b) Measure till button fully up

Figure 13: Button output measurement

5.4 Debug LEDs

The debug LEDs are measured with an oscilloscope. Figure 14 shows the output to a debug LED when it is switched on. In the software, all the debug LEDs are switched off and then the relevant debug LED is switched on during every cycle. This is clearly visible in the figure. The debug LEDs function as designed.

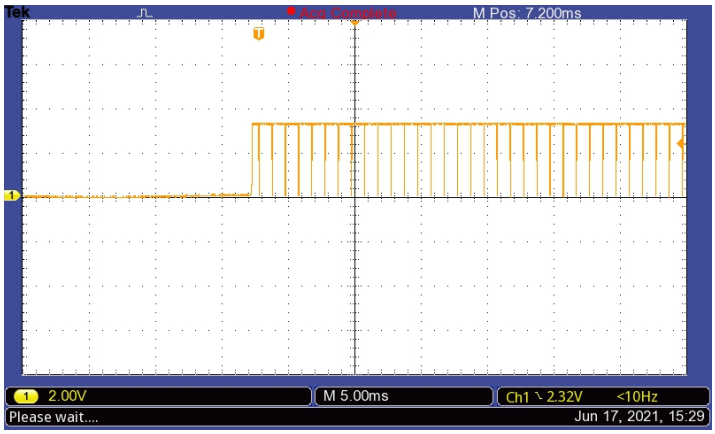


Figure 14: Output to debug LED

5.5 Dot matrix display

The dot matrix display is measured with an oscilloscope. Figure 15 shows the output of the microcontroller to a MOSFET that sinks an LED column in the dot matrix. Figure 16 shows the cursor measurements that are used to calculate the duty cycle of the LEDs. The measured duty cycle is 12.5%. This is the same as the designed duty cycle. The dot matrix functions as designed.

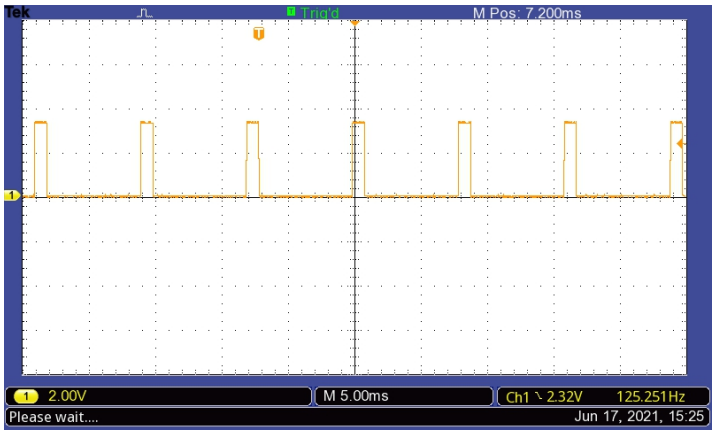
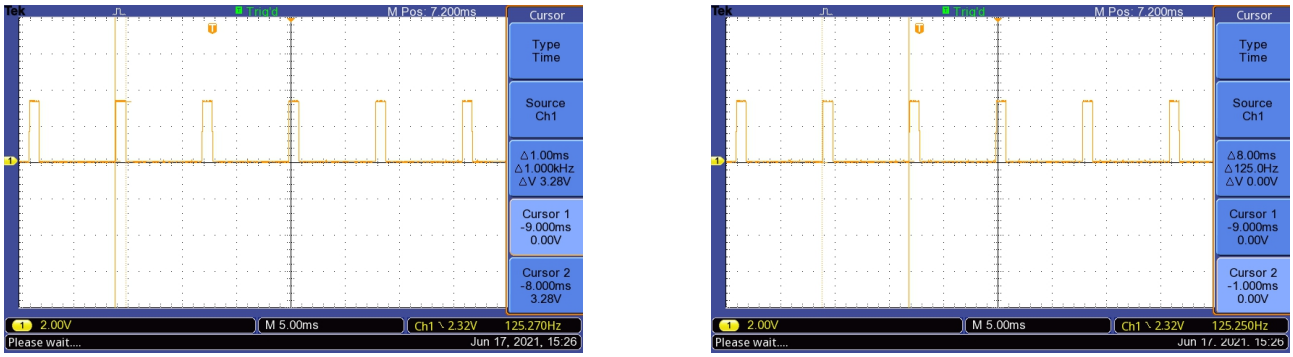


Figure 15: Dot matrix column switching



(a) Pulse width measurement

(b) Period measurement

Figure 16: Display duty cycle measurement

5.6 Slider

The slider is measured with an oscilloscope. Figure 17 shows the output when the slider is moved from end to end and back again. The figure shows that the slider outputs a linear voltage that can easily be sampled through ADC. The output of the slider varies between 0 V and 3.3 V. The functions as designed.

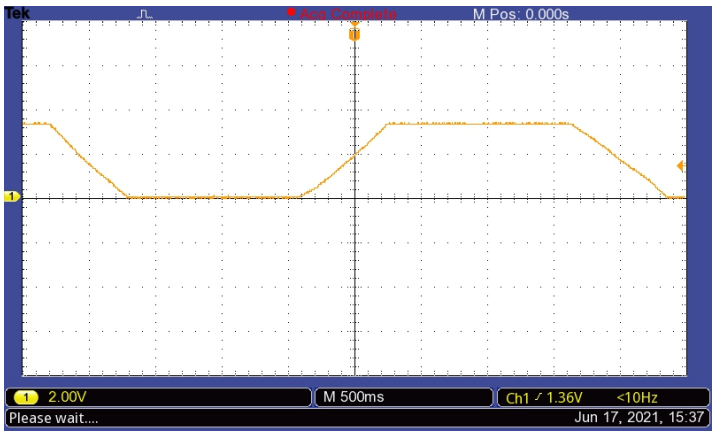


Figure 17: Slider output

5.7 IMU

The IMU is tested by measuring the SCL and the SDA line with an oscilloscope. Figure 18 shows the output of the SCL line and Figure 19 shows the output of the SDA line. From these two figures it is clear that the IMU functions as designed.

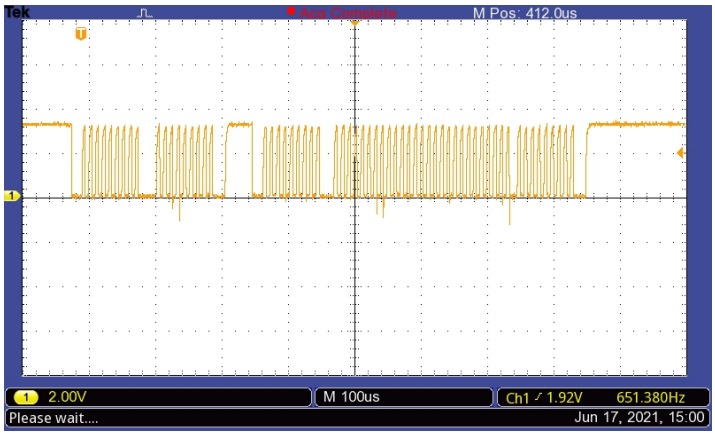


Figure 18: IMU SCL

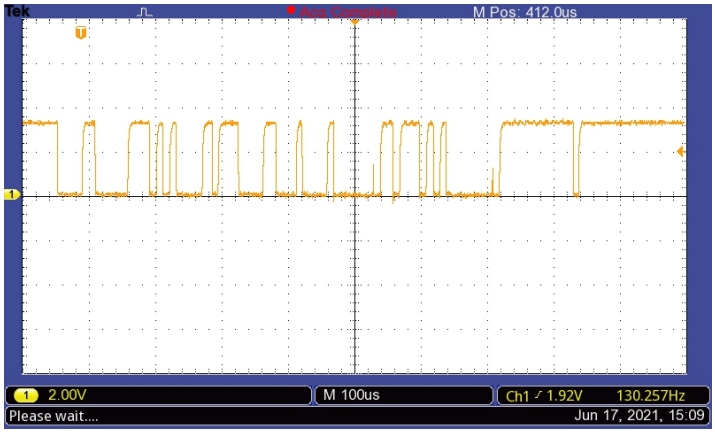


Figure 19: IMU SDA

6 Conclusion

The system complies with all the required specifications. There are no non-compliance's.

The 1N4007 diode of the power supply can be replaced with a Schottky diode. Most Schottky diodes have a voltage drop 0.15 V to 0.45 V. This is lower than the 0.6 V drop across the 1N4007 diode and will result in a lower power loss. An LCD screen can be used for the display rather than a LED matrix. This will result in a higher resolution for the display and allow for larger mazes to be displayed.

References

[1] *MCP1700 Low Quiescent Current LDO Data Sheet*, Microchip Technology Inc., Nov. 2013.

[2] *Medium-density performance line ARM-based 32-bit MCU with 64 or 128 KB Flash, USB, CAN, timers, 2 ADCs, 9 com. interfaces*, STMicroelectronics, Aug. 2015.

[3] *2N7000/2N7002/NDS7002A N-Channel Enhancement Mode Field Effect Transistor*, Semiconductor Components Industries, LLC, Oct. 2017.

[4] *PTA Series - Low Profile Slide Potentiometer*, Bourns, Mar. 2015.

[5] *MEMS digital output motion sensor: ultra-low-power high-performance 3-axis nano accelerometer*, STMicroelectronics, Dec. 2016.

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

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

Appendices

A Complete schematic

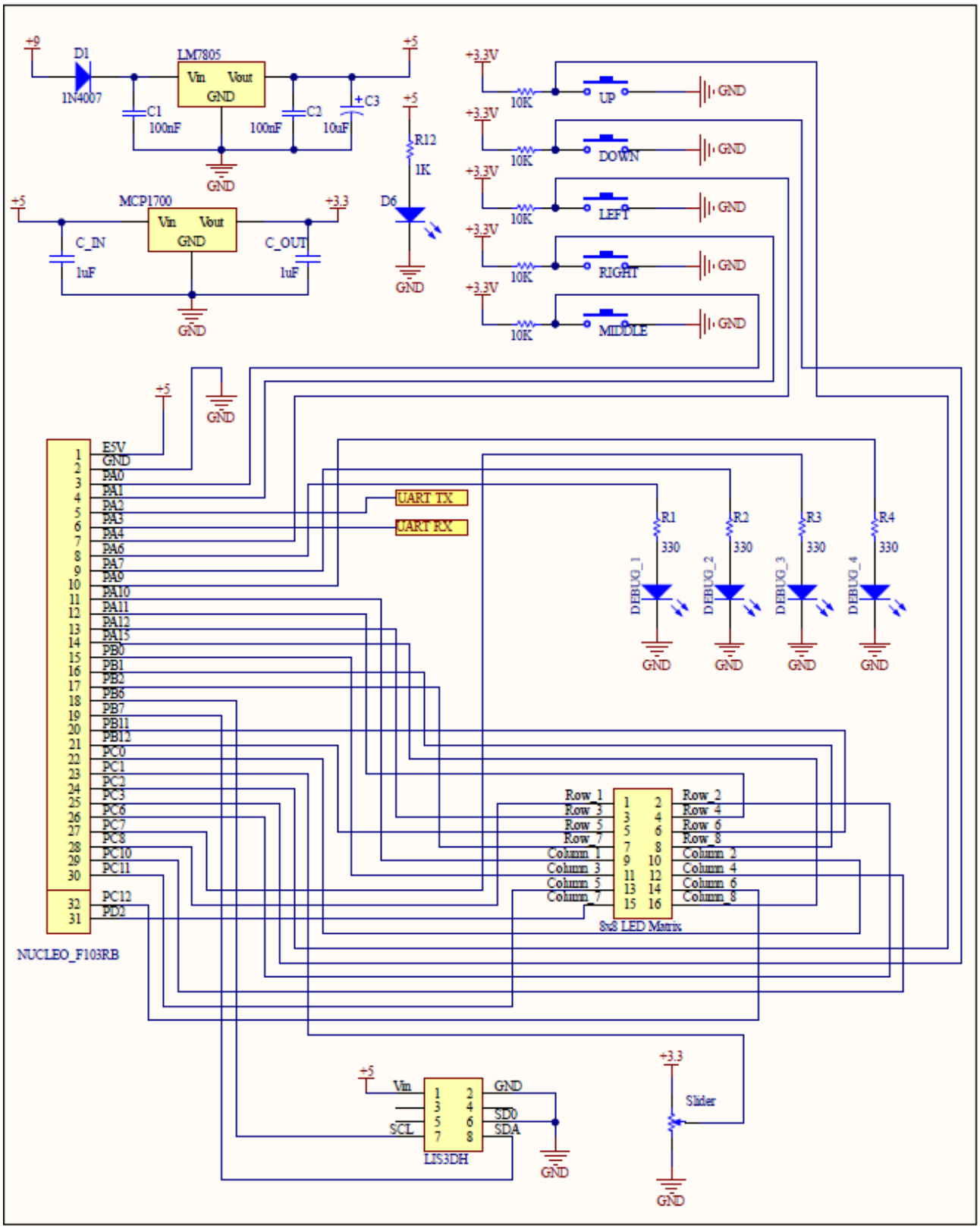


Figure 20: Complete schematic

B Pin Configuration

Pin	Configuration	Connected to
PA0	External interrupt	Middle button
PA1	External interrupt	Right button
PA2	USART2 TX	UART out
PA3	USART RX	UART in
PA4	External interrupt	Left button
PA6	Output	Debug LED 1
PA7	Output	Debug LED 2
PA9	Output	Debug LED 4
PA10	Output	Column 0
PA11	Output	Row 3
PA12	Output	Row 2
PA15	Output	Column 7
PB0	Output	Column 2
PB1	Output	Row 7
PB2	Output	Row 6
PB6	I2C1 SCL	IMU SCL
PB7	I2C1 SDA	IMU SDA
PB11	Output	Row 5
PB12	Output	Row 4
PC0	Output	Column 1
PC1	ADC1 IN	Slider output
PC2	External interrupt	Up button
PC3	External interrupt	Down button
PC6	Output	Row 1
PC7	Output	Debug LED 3
PC8	Output	Row 0
PC10	Output	Column 3
PC11	Output	Column 4
PC12	Output	Column 5
PD2	Output	Column 6

Table 3: Pin configuration