University of Cape Town

EEE3096S

Embedded Systems 2

Mini Project

Khavish Govind
GVNKHA001

Kam Norton
NRTKAM001

Ryan Jones
JNSRYA006

Natasha Soldin
SLDNAT001

26/09/2022

Contents

1 Introduction

The following mini project acts to demonstrate the knowledge accumulated from practical assessments performed in embedded systems 2 (EEE3096S). The project tasks students with the creation of a Light of Things (LoT) sensor system. The system will follow the structure shown by Figure 1.0.1 below:

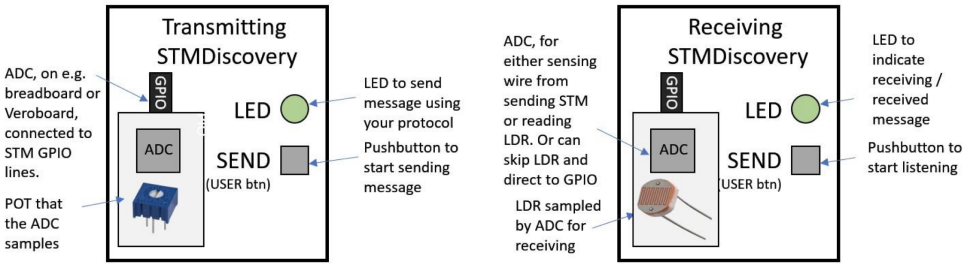


Figure 1.0.1: LoT Sensor System Overview [1]

From the overview in Figure 1.0.1 above it is evident that the system comprises of:

- A transmitting embedded system device (Discovery STM32F051R8T6 microcontroller) that samples analogue signals from a potentiometer (POT), converts the signals to digital using an analogue-to-digital converter (ADC) and transmits the data. Transmission occurs when the button (on the board) is pressed and is via either a wireless connection in the form of a light emitting diode (LED) or via a wired connection between GPIO pins.
- A receiving embedded system device (Discovery STM32F051R8T6 microcontroller) that received data either via a light dependent resistor(LDR) and light signal input or via a wired GPIO pin connection. Data intake occurs when the button (on the board) is pressed and an ADC is used to interpret the incoming message from the transmitting device.

The following report details the project and its various sections which have been broken down into 5 parts:

1. Requirements and LoT Message Protocol - which contains the project's requirements as well as the system's technical requirements. This section also details the LoT Message protocol and justifications for this structure.
2. Specifications and Design - which contains the quantitative system specifications and initial design including a flowchart of the system's operation, circuit diagram and description of C code classes and modules.
3. Implementation - which contains detailed explanation of the project's practical implementation and snippets of relevant C code.
4. Validation and Performance - Performance and Results of the System Designed
5. Conclusion - which contains a summary, potential market and plan for future work.

1.1 Major Design Choices

Some major design choices include the following:

1. The practical circuit external to the STM Discovery Boards will include a POT to be sampled by the on-board ADC of the transmitting device. The POT will be connected one one end to a 3.3V input, and on the other end, to ground. The output of the POT is what is sampled by the ADC and connected directly to the ADC of the STM.

2. The practical circuit will also include an LDR, which will have an output voltage across a resistor connected to the ADC of the receiving STM.

3. The external circuits and both the STMs should share a common ground for the sake of uniformity and consistency and so that there is a uniform reference.

4. The STM's will communicate through the GPIO pins using wires, as that method is both more reliant and stable when testing as many factors can change the reliability of transmitting data through LED's

1.2 Github Repository

Github was used extensively throughout the project's duration to aid the students in version control and collaboration. The git repository can be accessed here.

1.3 Contributions

The group members and their corresponding contributions can be found in table 1 below:

Group Member	**Section**	**Section Number**	**Page Number**
Khavish Govind	Transmitting System	N/A	N/A
	Introduction	1	1
	Design	3.2	6
Kam Norton	Transmitting System	N/A	N/A
	Requirements	2.1	3
	Specifications	3.1	6
Ryan Jones	Receiving System	N/A	N/A
	LoT Message Protocol	2.2	4
	Conclusion	6	19
Natasha Soldin	Receiving System	N/A	N/A
	Implementation	4	11
	Validation	5	18

Table 1: Contribution Table

2 Requirements and LoT Message Protocol

2.1 Requirements

2.1.1 Project Requirements

The project has the following broad requirements which are taken directly from the project outline document [1]:

- List of planned activities with the project and allocation to team members
- Description of the LoT message protocol, its message structure illustration, timing diagram and textual description. It caters at least for the means to send an ADC sample and a different checkpoint message.
- Circuit diagram for the system
- Plan to approach the project and justification / motivation for design choices.
- Design Diagram(s): hardware and software design aspects (flow charts).

2.1.2 System Requirements

The project has the following technical system requirements which are derived from the project outline document [1]:

- The two embedded devices are set up using micocontrollers.
- An analogue signal needs to be obtained and converted to a digital signal such that it can be transmitted.
- The digital signal needs to be structured into an appropriate message protocol.
- The devices need to transmit/ receive data only when the button is pressed on the respective devices.
- Data transmission can be wired or wireless.

2.2 LoT Message Protocol

The proposed message protocol structure can be seen in Figure 2.2.1 and into Figure 2.2.2 and further explained below. The structure has been cut into two lines in-order to improve readability.

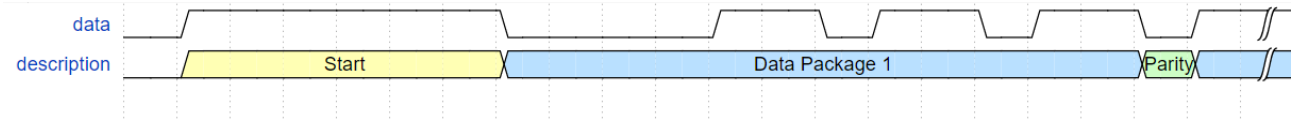


Figure 2.2.1: LoT Message Structure Part 1

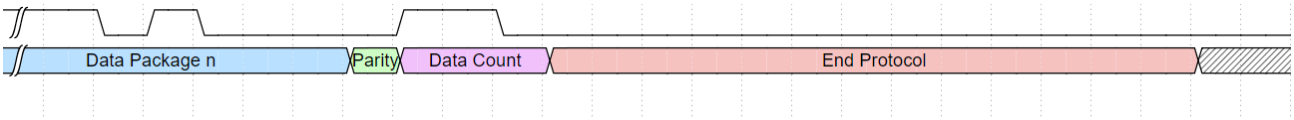


Figure 2.2.2: LoT Message Structure Part 2

The following conditions are required by the message protocol:

1. **Start Condition:** The LED is held high for 6-bits after idling low.
2. **Data:** The data is sent as a 12-bit signal with the MSB first, and the LSB last (i.e. in big-endian format).
3. **Parity Bit**: Odd parity is used. If the number of 1s in the data package is odd, the parity bit is set to 0. If the number of 0s in the data package is even, the parity bit is set to 1.
4. **Data Count:** A 3-bit binary value is used to acknowledge the total number of data sequences that have been sent in the message structure.
5. **End Condition:** The LED is switched low for 13-bits.

The following, corresponding, decisions were made about the message protocol:

1. **Start Condition:** As the POT can read values in the range $0_{10} \rightarrow 4095_{10}$, a 12-bit value is required to be used to transmit all values in the range of the POT. Practical limitations of the circuit meant that the POT would never reach its hypothetical maximum of 4096_{10}. This means, that a start condition of 111111_2 will never be confused with data as, in practice, this is not a possible data stream as in testing, the POT reached a maximum value of 3950_{10}. Treating the Start sequence as a 12-bit binary number (111111000000_2) gives a decimal value of 4032_{10}, which is higher than the actual value that the POT can output.

2. **Data:** As described above, the POT can output values of the range $0_{10} \rightarrow 4096_{10}$. This requires a 12-bit data stream as:

$$2^{12} = 4096$$
$$\therefore Range of data:$$
$$000000000000_2 \rightarrow 1111111111111111_2$$

3. **Parity Bit:** Odd parity was chosen as if the POT reads a value 0_{10}, the parity check is expected to add a 1_2 to the message protocol. This allows the transmitter to know that this is the expected message, as opposed to an error in the transmission algorithm or protocol. [2]

4. **Data Counter:** As 3-bits are used, this means that 7-bits of data can be sent in one package. This was decided to be adequate, as anything more than this would be excessive to package into one protocol.

5. **End Condition:** The end sequence of 13-bit 0's is adequate as it is preceded by the data bit which will never be 13-bit of 0's due to the use of an odd parity bit. If the 12-bit data signal comprises of all 0's then the 13th parity bit will be 1 which will not terminate the system's operation.

3 Specifications and Design

3.1 Specifications

The following system specifications quantitatively describe the System Requirements in Section 2.1 above.

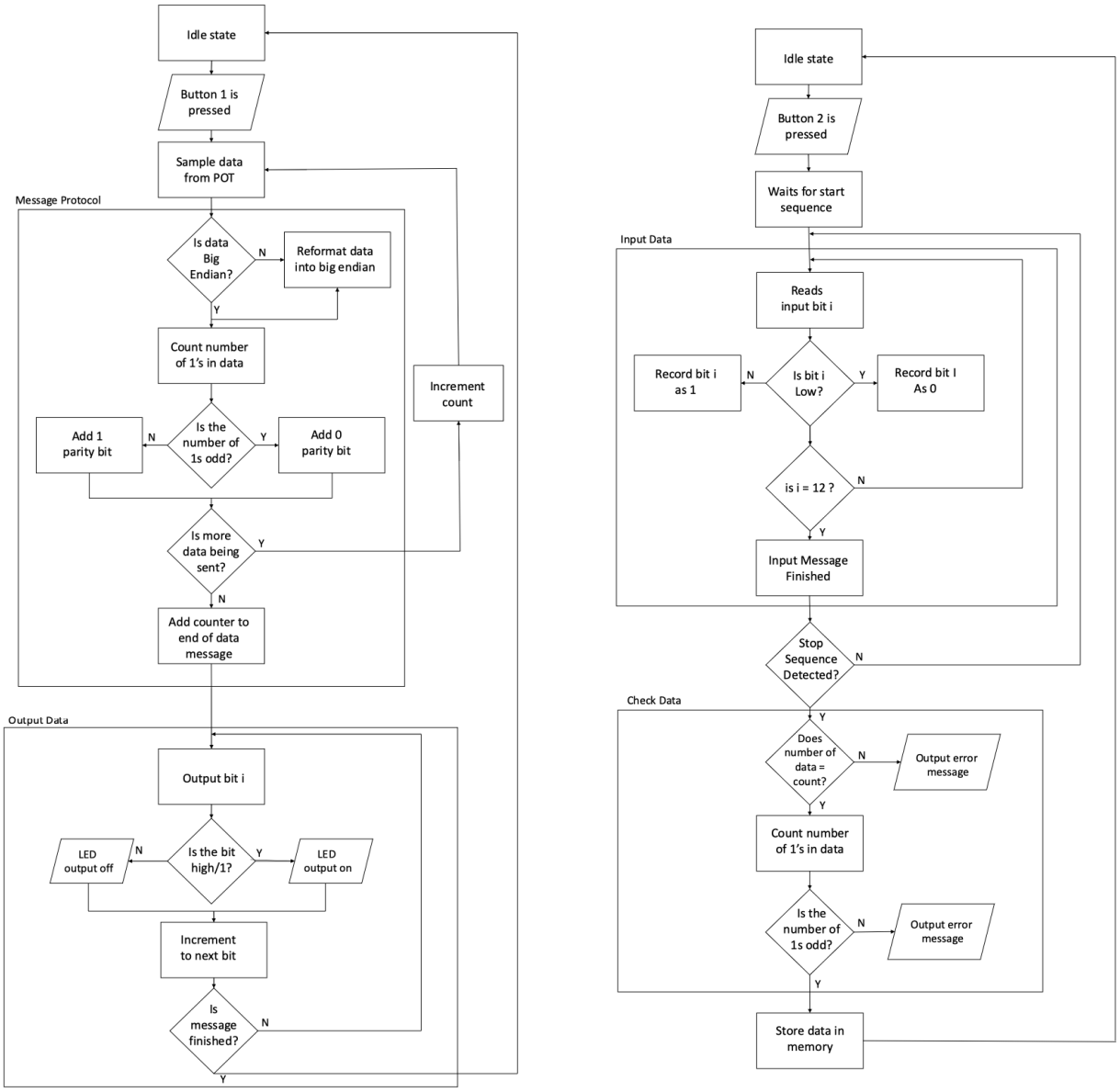
- STM32F051R8 micrcocontroller are used to create the embedded systems and code is written in C in STMCUBEIDE such that is it compatible.
- A POT is used to obtain an analog signal and the STM32F051R8's onboard ADC is used to convert the signal to a digital one.
- The data is structured into the following message structure: start sequence, data (in little endian), parity bit, end sequence.
- Interrupts in C will be used to implement the button pressing.
- Wireless data transmission occurs via an LED and received with an LDR and wired data transmission occurs using GPIO pin connections.

3.2 Design

The project and its required operation can be graphically shown by the flow charts in Figure 3.2.1 below. The operation of the transmitting device is represented in Figure 3.2.1a below and the operation of the receiving device is represented in Figure 3.2.1b below.

The operation of the transmitting system is as follows: the system is initially in an idle state waiting for input, the button being pressed acts as input signalling to the system that data is ready to be transmitted. Data is sampled from the POT and then the structure of that data is verified by first checking that the data is in little endian form and then attaching an odd parity bit. This repeats for however many data packages are sent (7 is maximum) and a count of the number of data packages is attached at the end. The data, being in the correct format, is then outputted by converting each bit into a high/ low voltage which translates to turning the LED on/ off respectively and also sent through the wired connection through the receiving system. Once data transmission is finished, the system returns to its idle state.

The operation of the receiving system us as follows: the system is initially in an idle state waiting for input when the button being pressed, this acts as the input signalling to the system that data is ready to be received, the start sequence of the data (i.e. 2-bit start sequence of 1's). Data is received bit by bit and translated into 0/1 for low/high light signals or through a wired connection. More than one data message can be received as the device will only stop receiving once the stop sequence (i.e. 13-bit stop sequence of 0's) is detected. The input data is checked to be of correct format using the odd parity bit and count to ensure the same amount of data is received as was sent. If the data is found to be of incorrect format, then an error message will be outputted to the user to signal that data corruption has occurred during transmission. Otherwise, the data is stored into memory or outputted and the system returns to its idle state.



(a) Transmitting System

(b) Receiving System

Figure 3.2.1: System Flow Chart

3.2.1 Circuit Diagram

The system can be configured according to the circuit diagram in Figure 3.2.2 below. The diagram was created using KiCad [3].

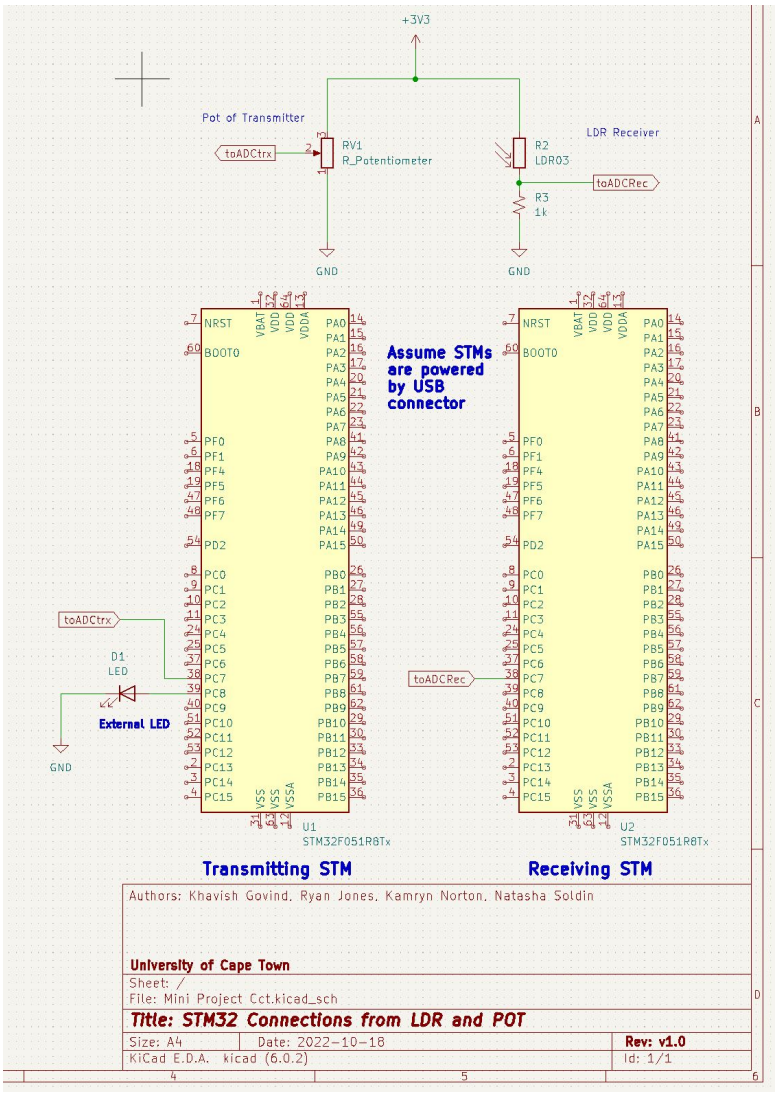


Figure 3.2.2: LOT External Circuitry

This diagram shows the two STM32F051R8 microcontrollers used to create the two transmitter and receiver embedded systems. Both systems utilize their respective on-board push buttons and ADCs.

The external circuit of the transmitter system includes a POT connected to the on-board ADC (CH1, pin PC7) which allows for analogue signals to be taken from the POT and digital converted signals to be created for transmission. The POT is connected to 3.3V from the transmitting board and GND on the other end. Data will be transmitted through two means:

1. **Wired Connection:** GPIO pin connection between the transmitter and receiver microcontrollers.

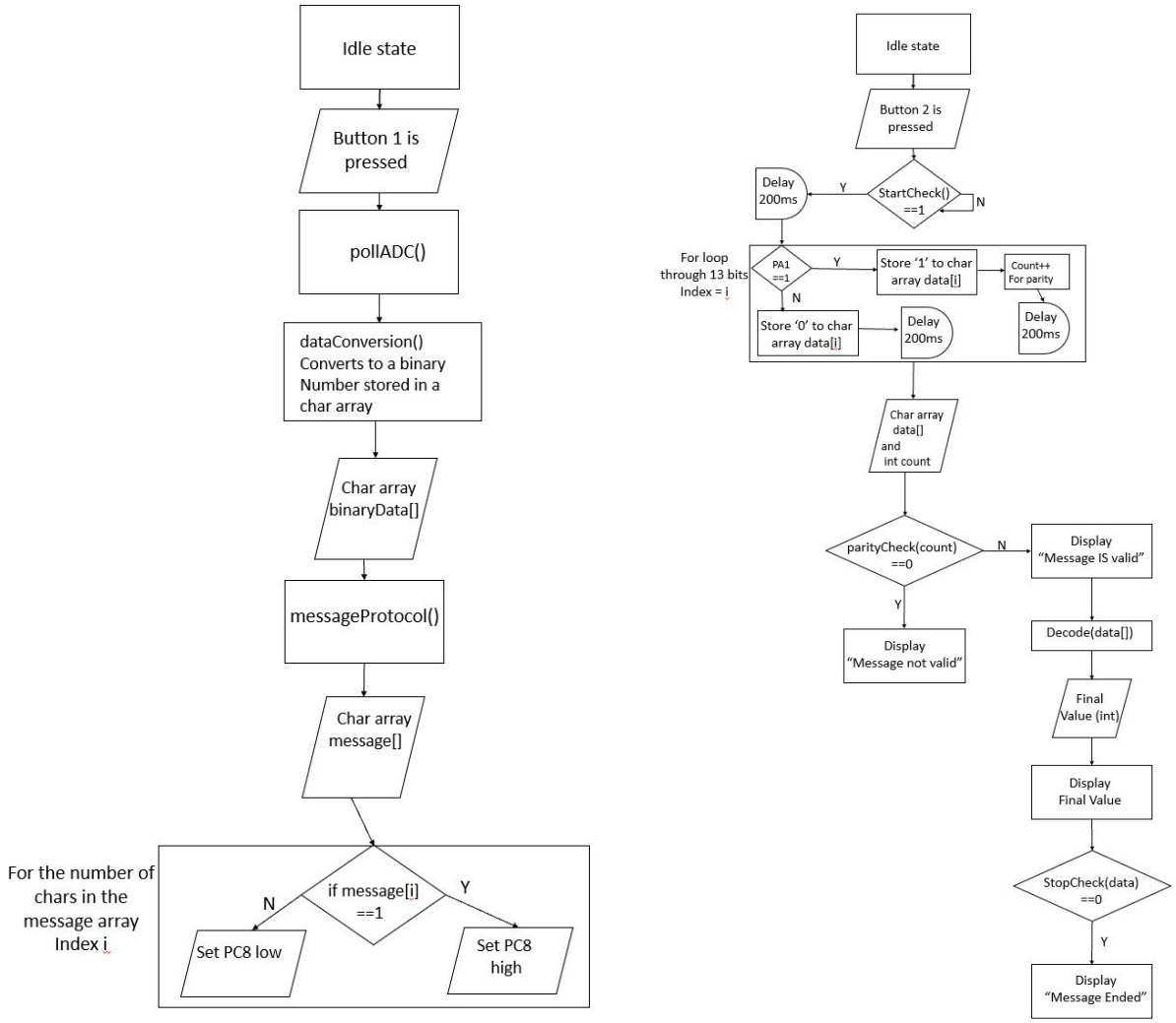
2. **Wireless Connection:** Binary data will be sent using an external LED on the transmitter microcontroller and received using an LDR connected to the receiving microcontroller.

The external circuit of the receiver system comprises of an LDR which will be used in the wireless-connected implementation of the project to detect light signals sent by the transmitter's LED. The output of the LDR is connected to the receiver system's onboard ADC (CH1, pin PC7) such that an analogue signal can be taken from an LDR and a digitally converted signal can be rendered for analysis on the receiver system. The LDR is connected to 3.3V from the receiving board and the output of the LDR is read across a $1k\Omega$ resistor connected to ground.

Although not explicitly shown, it is important to note that the STM32F051R8 microcontrollers are connected to external PCs through a USB connection to power them as well as through a USB to UART conversion circuit to allow for the use of a serial monitor to display the system's output. Another factor not shown in the diagram would be the GND connections of the systems. It is important to note that the systems will share a common GND connection to provide a common reference point.

3.2.2 UML Diagram

The following UML diagrams in figure 3.2.3 below follow a similar structure to the flowcharts in figure 3.2.1 above but differ as they include specific class and function names.



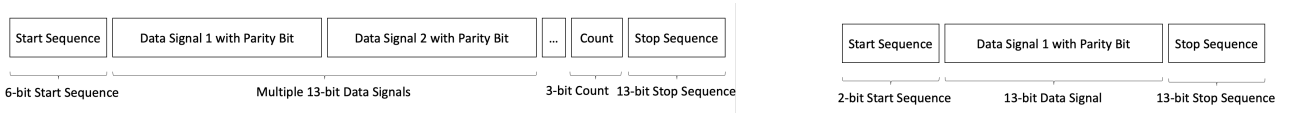
(a) Transmitting System (b) Receiving System

Figure 3.2.3: System UML Diagrams

4 Implementation

The following decisions were made before practical implementation commenced, these were made to keep in mind the project's duration time constraint which limited the students in what they are able to achieve.

- Although the implementation outlined in Specifications and Design above detailed a wired or wireless connected system. The students opted for a wired system where data can be sent from transmitter to receiver via a GPIO pin connection.
- The data will be transmitted bit by bit and reconstructed on the receiver side which would allow for ease of extrapolation when implemented via a wireless connection as stated in the Plan for Future Work section.
- The transmitted data will be package into a simplified message protocol - which is limited as only one data set (or one analogue signal sampled from the POT) is sent at a time. Where the data signal comprises of 12-bits of data and a 1-bit parity. The intended LoT Message Protocol as stated in Section 2.2 above is structured as shown in Figure 4.0.1a below but the chosen simplified stricture is shown in Figure 4.0.1b below. The simplified message protocol structure is shown below with an example bit representation as well in Figure 4.0.2.



(a) The Intended Message Protocol 2.2

(b) The Simplified, Implemented Message Protocol

Figure 4.0.1: Message Protocols

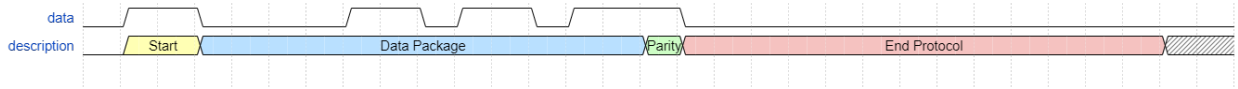


Figure 4.0.2: Simplified Message Protocol

The proposed practical implementation of the project was defined in Section 3, Specifications and Design, where the LDR and POT circuits were defined and the connections to the Discovery boards were outlined. This includes the LDR and POT circuits. The implementation of the circuits can be seen in the following annotated image in Figure 4.0.3 below:

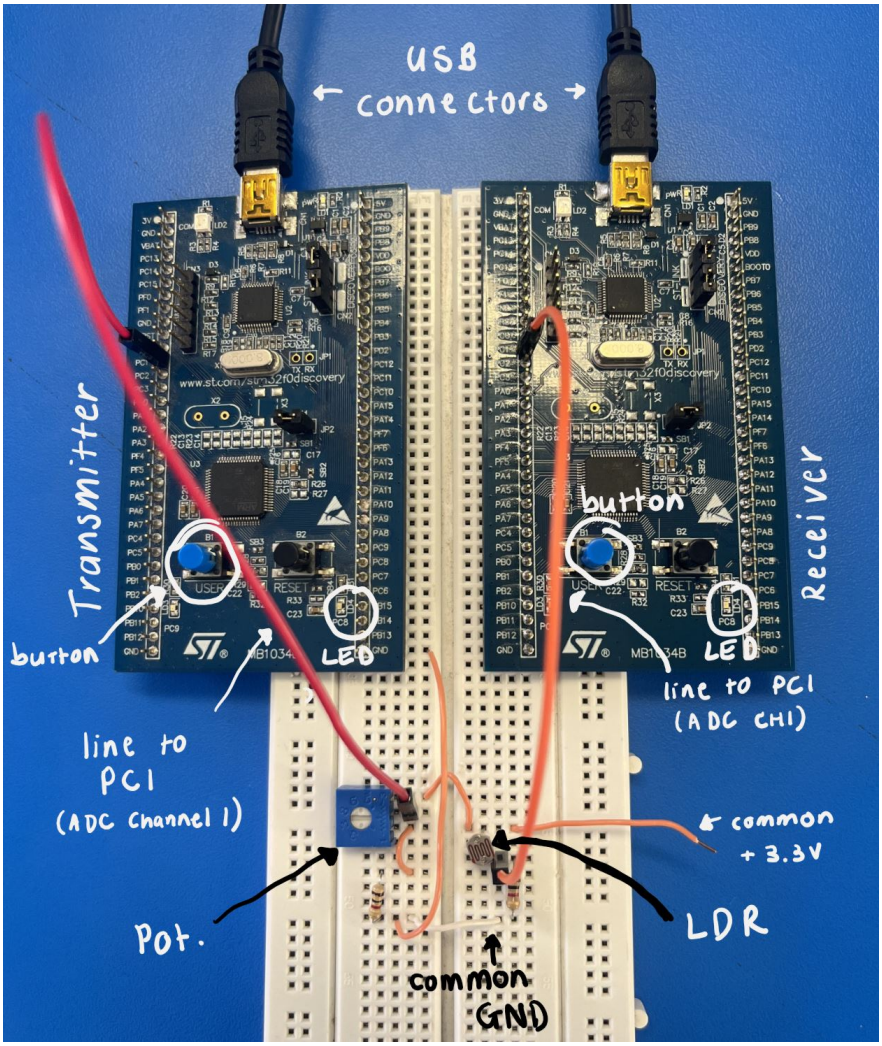


Figure 4.0.3: Practical Circuit Implementation

4.1 Important Methods

The following C code extracts are relevant and aid in system operation. The methods described in Section 4.1.1, 4.1.2 and 4.1.3 pertain to the transmitting system and the methods described in 4.1.4, 4.1.5, 4.1.6 and 4.1.7 pertain to the receiving system.

4.1.1 pollADC() Method

The following method extracts the digital signal output from the ADC. The input of which comes from the analog signal extracted from the POT [4]

```
/**
 * @function pollADC
 * @abstract Poll the ADC on PA7 on the transmitting STM. The input to this pin
    is the potentiometer
 * @param void
 * @result uint32_t A value that the ADC reads in from the potentiometer
 *
```

```
 */
uint32_t pollADC(void){

        HAL_ADC_Start(&hadc); //Start the ADC

        HAL_ADC_PollForConversion(&hadc, 1); //Poll for conversion

        uint32_t adc_val = HAL_ADC_GetValue(&hadc); //Get the ADC value

        HAL_ADC_Stop(&hadc); //Stop ADC

        return adc_val; //Return the ADC Value

        // Code referenced from: https://controllerstech.com/stm32-adc-single-
            channel/
}
```

4.1.2 dataConversion() Method

The following method takes an unsigned integer value of the analog signal and converts it into a 12-bit binary data format within a char array [5].

```
/**
 * @function dataConversion
 * @abstract Converts a unint32_t into a binary char array of length 12
 * @param uint32_t The ADC values that is polled using the pollADC method.
 * This is converted to a 12-bit binary char array as the resolution of the ADC
    is 12-bits
 * @return void The global variable binaryShort[12] is updated when this method
    is run
 */
void dataConversion (uint32_t adcVal){

        uint32_t i; //Counter Variable
        int counter = 0;
        char binary[32]; //Message Protocol output
        for (i = 1 << 31; i > 0; i = i / 2){
        //Loop through the 32-bit uint32_t variable and convert to a binary char
                (adcVal & i) ? (binary[counter] = '1') : (binary[counter] = '0')
                    ; counter++;
        }

        int dataCounter = 0;
        for(int i = 20; i <= 32; i++){
        //Loop through the 32-bit char array and add the last 12-bits to the
            binaryShort[] array
        //These 12-bits are the 12-bit ADC value
                binaryShort[dataCounter] = binary[i];
                dataCounter++;
        }
}
```

4.1.3 messageProtocol() Method

The following method takes in the data char array and packages it into the message protocol. This method adds the 2-bit start sequence, parity bit and 13-bit stop sequence.

```
/**
 * @function messageProtocol
 * @abstract Converts a 12-bit char array to the desired message protocol for
    transmission
 * @param 12-bit char array
 * @return void The global variable message[27] is updated when the method is
    run
 */
void messageProtocol(char binaryData[]) {

        int counter = 0; //Parity check counter of # of 1s

        //Add start bits
        message[0] = '1'; //Sets the first bit of the message to 1
        message[1] = '1'; //Sets the second bit of the message to 1

        //Loop through the polled ADC binary value and add them to the message
           array
        //in the correct position after the start bits
        for (int i = 0; i <= 12; i++) {
                message[i+2] = binaryData[i];
                //Count the number of 1s in the ADC binary message in order to
                   check parity
                if (binaryData[i] == '1'){
                        counter ++;
                }
        }

        //Add an odd parity bit after the message
        if (counter % 2 == 0) {
                // If there are an odd number of 1s, add a 0 to the parity bit
                   to satisfy odd parity
                message[15] = '0';
        }
        else if (counter % 2 == 1) {
                // If there are an even number of 1s, add a 1 to the parity bit
                   to satisfy odd parity
                message[15] = '1';
        }
        //Add the stop condition to the last 13 bits of the message
        for (int j = 16; j <= 32; j++) {
                message [j] = '0';
        }
}
```

4.1.4 StartCheck() Methods

This method checks to see if the start sequence is correct (i.e 2-bit sequence of 1's), if correct it returns a 1 and if incorrect it returns a 0.

```
/**
* @function StartCheck
* @abstract Checks if the start condition of the message protocol is met
* @param void Reads in the value from the appropriate GPIO Pin
* @return int 1 if the start protocol is met, 0 if it is not met
*/
int StartCheck(void)
{
        for(int a = 0; a < 2;a++)//Starts a loop to check for the first two bits
        {
                if(HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_8) == 1)//Checks if the
                    first bit is a 1
                {
                        HAL_Delay(200);//Delays of 200ms before second bit is
                            checked
                        if(HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_8) == 1)//Checks to
                            see if the second bit is a 1
                                        {
                                                return 1;//Start Sequence is
                                                    confirmed
                                        }
                        else
                                return 0;//Start Sequence  is wrong
                }
                else
                        return 0;//Start sequence is wrong
        }

        return 0;
}
```

4.1.5 ParityCheck() Methods

This method checks if the data message with parity bit has an odd number of 1's signalling that it meets an odd parity check requirement. This method returns 1 if the parity check yields an odd number of 1's (i.e. if data is correct) and a 0 if the parity check yields an even number of 1's (i.e. if data is incorrect/ lossy).

```
/**
* @function parityCheck
* @abstract Checks if the transferred parity bit is correct
* @param int The count of the total number of 1s in the message sequence
* @return int 1 if the parity is correct, 0 if not correct
*/

int parityCheck(int count)//Checks whether the parity bit of the data matches
    the data
{
        int parity = count%2;//stores the value of the number of 1's mod 2 into
            a variable
        if(data[12] == parity)//Compares the 13th bit of data(parity bit) to the
             the parity calculated from the data
        {
                return 1;//Returns 1 if the parity matches
        }
        else
                return 0;//Returns 0 if the partiy does not match
}
```

4.1.6 StopCheck() Method

This method checks to see if the stop sequence is correct (i.e 13-bit sequence of 0's), if correct it returns a 1 and if incorrect it returns a 0.

```
int StopCheck(char data[])//Check whether the stop sequence is activated
{
        int end = 0;
        for(int k = 0;k < 13;k++)//Runs loop through all 13 data bits
        {
                if(data[k] != '0')//Checks if the data is received is 0s
                {
                        end = 0;//Set end to zero is a zero is received
                }
                else
                {
                        end = 1;//Should a 1 be in the message, end will be
                            become 1
                        break;//Exits the the loop
                }
        }
        return end;//Return the value of end

}
```

4.1.7 Decode() Methods

This method accepts a char array representing a binary number and converts it into a decimal number.

```
/**
* @function Decode
* @abstract Converts a binary char array into the correct decimal value
* @param data[] char array to be converted to a double
* @return double The double value that corresponds to the char array that is
    given as a parameter
*/
double Decode(char data[])
{
        double total = 0;//creates a total varable for the binary value
        for(int a = 0; a <= 11; a++)//Runs a loop to run through the data is big
            endian format
        {

                if(data[a] == '1')//Checks when the vaue of the array is  1
                {
                        double power = pow(2.0,(double)(11-a));//Adds the 1 to
                           the correct power of 2 into the total variable
                        total = total + power;
                }
        }
        return total;//Returns the final decimal value
}
```

5 Validation and Performance

Using the configuration and implementation in Section 4 above. The following method can be used to test the system's performance and to validate its operation:

1. The POT should be set to a specified value using the wiper.
2. The polled digital signal should be read from the output of the ADC and should align with the expected converted value.
3. This polled digital signal is then converted into a binary value, and stored as a char array. This binary conversion should be validated using a serial monitor output to ensure proper conversion.
4. The message protocol is then applied to the binary data number and the entire message is stored as a char array. This includes the addition of an odd parity bit as well as the start and stop protocols. The structure as well as parity bit calculation should be validated.
5. This message is then transmitted via a GPIO pin connection between the two STM32F051R8 microcontrollers.
6. The data will then be read bit by bit and reconstructed on the receiving system. This would need to be validated to be the same by comparing the serial outputs on both the receiver and transmitter systems.
7. The data structure should then be deconstructed and the parity bit should be verified to ensure no data loss or tampering has occurred.
8. The data portion of the message should be converted back from binary to decimal to ensure the polled ADC value is the same as what was transmitted.

For instance: the POT wiper is turned completely to the one side and therefore has a polled ADC value of 3900, as discussed in Section 2.2 above this is the maximum value that the POT can achieved due to the non-ideal nature of the system - it cannot achieve its ideal maximum of 4095. The binary conversion of this is:

$$3900_{10} \rightarrow 111100111100_2$$

Applying the message protocol includes: adding the start sequence (2-bit 1's), adding the stop sequence (13-bit 0's) and adding the odd parity bit which in this case is a 1 because of the number of 1s. The final message protocol is:

$$11111100111100100000000000000_2$$

After transmission, the received data is: $11111100111100100000000000000_2$. Once compared to the transmitted data it can be seen that it is identical and there are no losses. Removing the start and stop sequence results in the following data structure: 1111001111001_2. Counting the number of 1's results in an odd number (9) which implies that the data is correct as odd parity holds true. Removing the parity bit and converting from binary to decimal results in:

$$111100111100_2 \rightarrow 3900_{10}$$

This is the initial/ correct polled ADC value abstracted from the POT
Therefore, the system's operation and overall performance is validated as correct.

6 Conclusion

The project was considered to be implemented correctly and efficiently given the projects description, system and project requirements, derived specifications as well as project's constraints and duration.

This system would be a viable product in scenarios where sampling is required to be sent back to a central point. Mining is an example of an instance in which different transmitters can be installed in different sections of the mine and can be interacted with by foreman to report air quality, heat or current depth of the mine. This data would be received by a central control unit. Other applications include the manufacturing industry, the technological development industry, or anywhere requiring short-range monitoring and communication.

6.1 Plan for Future Work

Due to the limited time duration of the project only a simplified version was produced which detailed the GPIO pin or wired connected implementation of the project. The future plan for the project would be to extrapolate the existing implementation to the LED and LDR or wireless connection. This would include:

1. Transmitting the message char array using the LED by looping though it and using the 0/1 values to signal low/high voltage and thus turning the LED off/on respectively.

2. Implement the proposed LoT Message Protocol which includes a count and the potential to transmit multiple data messages within a single transmission.

3. Include the means to receive the message sent over the LoT Message Protocol, using the implementation of polling the receiving STM's ADC, which will be connected to the output of the external LDR circuit as outlined in the specifications section. The ADC will effectively poll values it is reading from the LDR, and code could be used to process that data and place it back into the LoT Message Protocol.

4. Include the use of the pushbuttons for initiating the transmission and reception of messages on each respective STM Discovery Board. For the receiving system to have a button, which idles the system waiting for a message while also a push button function on the transmission system to instantly send a message. This was not done within the project because of time constraints.

References

[1] D. S. Winberg, *Embedded Systems EEE3096S/EEE3095S Mini-Project Part A& B*, 2022.

[2] TopCat, "Why bother with even parity?," 2019. https://electronics.stackexchange.com/questions/349216/why-bother-with-even-parity.

[3] KiCad.org, "Kicad eda," 2022. https://www.kicad.org/.

[4] Admin, "Adc single channel in stm32 using poll interrupt and dma," Oct 2022. https://controllerstech.com/stm32-adc-single-channel/.

[5] M. Bhojasia, "Decimal to binary in c," 2022. https://www.sanfoundry.com/c-program-binary-number-into-decimal/.