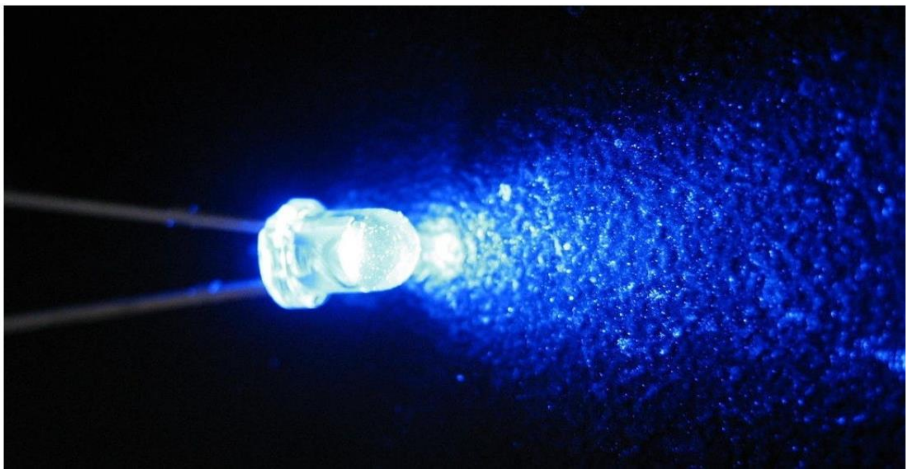


LOT MINI PROJECT

TSHTHA094, MRBMAT004. MSWGER001, NKHBES001

ABSTRACT
LED lights can be modified to transmit information over the air.

EEE3096S
UNIVERSITY OF CAPE TOWN

Table of Contents

INTRODUCTION

The system implemented is Light of Things (LoT). This system uses light to send messages or information across, using two microcontrollers, one being the transmitter, and the other acting as the receiver. The use of UART and a protocol such as RS232 was restricted, thus using a wired GPIO was the chosen method of data transmission between the two microcontrollers, to preserve the data integrity in the transmission and reduce external noise if having had used another external LDR circuit on the receiver. This LDR circuit would require that we use ADC and introduce noise from amplifiers (components) as amplifiers are needed to strengthen the signal.

PLANNED ACTIVITIES

The two main systems are the: Transmitter and Receiver.

The system was designed (transmitter and receiver), and the transmitter was implemented by, and report written by TSHTHA094 and MRBMAT004. The receiver system was implemented by MSWGER001 and NKHBES001.

The main systems were each broken into subsections of tasks that each had to be achieved to realise the full system. The subsystems will be discussed further in the Requirements and Implementations section and shown on the flowcharts. An external supporting circuitry diagram is included to better show some of the sub-sections of the sub-systems. The report structure follows the rubric guideline and layout of the report is presented in the table of contents above.

Design Choices

In the design process the team found it easier to implement an already working system that has its perimeters well defined. The team’s design of the LoT system will use am makeshift I2C protocol. The makes use of the transmitter as the master and the receiver as the slave. The subsystem specifications are as follows:

Transmitter system

The transmitter will make use of an Analog data set that well be sampled from A POT or Known as a Potentiometer. The signal will be passed through one of the selected GPIO pins of the STM and an Analog to Digital converter (ADC). The ADC value that we sample will then be converted to a binary array that will be passed as high and low voltages with the use of an external LED. This binary value will be encoded in start bits and end bits that will construct the data package the team will transmit.

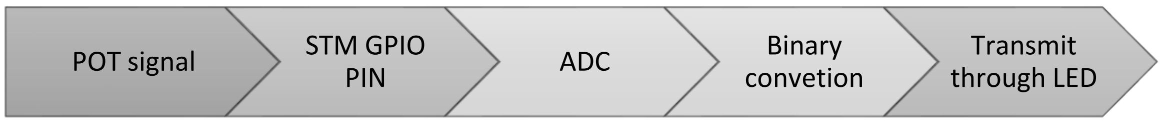


Figure 1: Transmitter system diagram

Start Bits

The team made use of this choice to imitate the away the master writes to the slave address and without his bit the transmitter system will not be able to communicate with the receiver.

End Bits

The uses of these bits were to have a way of saying that the transmitter has finished sending out its data so that the receiver knows that it must stop listening after the end of the bit or remain idle for the next data set.

Receiver system

The Receiver system makes use of a LDR system that will act as a voltage divider system that will change voltage in accordance with the light source. This change in voltage will be sampled by the GPIO Pin of the STM. The Light source will be a mixture of highs and lows that were constructed by the binary values that are being sent out by the transmitter. The receiver makes use of write address and it will contently wait for this message if the massage in not found the receiver will remain idle and not responded.

The received data will be taken to an ADC and for there the team used a form of comparator operation to set values to either one's or zero's that will reproduce the input from the transmitter.

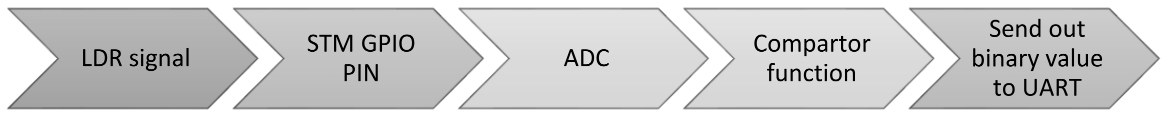


Figure 2: Receiver system diagram

Start bits and Acknowledge

When the correct start bit received it will grant access to the master to write to it and in the light of the I2C protocol the STM will flash an acknowledge signal to show it is read to take in data and the this is done by the on-board LED's. The STM also Turns on one of the other LEDs to indicate that it is read to be written to.

End bits and Acknowledge

When the receiver gets these bits, it will know that it is no longer required to listen and to show this it will turn off its receiver LED and send out an acknowledge and stay idle.

Requirements & LoT Message Protocol

Requirements

In the making of the team's system there are several design requirements that need to be implemented to have the system be deemed as functional. Below is a view of these system requirements and the subsystems that given them.

Transmission requirement

In summary the requirement of this section will be to take an Analog data set and transmit it through a light medium.

The system that must be built requires a means of taking Analog data and convert it to Digital data and that digital data will be sampled bit for bit to and LED out put that will receive high and low voltage values that will be regarded as the light of things data transmission. To implement the Analog conversion, the use of the STM's internal ADC protocol. The ADC value that is received from the ADC is a value that is between 0 and 4095. The value can be converted to a 12-bit value using decimal to binary algorithm. With this 12-bit value there is a means to be able to send out high and low voltage values by storing the 12-bit value in an array that will be called on to pass out one bit and output a high/low voltage to an STM pin that is connected to an external LED circuit.

Data representation

The from in with we had to represent the data came up as on of the subsystems when looking at data transmission. Though there is a means to access the 12-bit value and send it out as high/low voltages, questions of how long a pulse should last for and how will the duty cycle of the pulse look come to the table. The team decided to use the example given in the instruction sheet of having each pulse be 500ms long and have a 50% duty cycle. These values were good as they allowed for the team to detect changes with the naked eye, and it gives time for the receiver to collect the data this will be looked at in depth in the Receiver requirement.

Encoding of data

The encoding of the data was as subsystem that came to play when trying to send data. The reason for this was to be able to depict the beginning of new data and that end of that data package so that we do not sample the wrong packages. The application of this requirement allows for better means of error checking as it will allow to maintain the size of the data that in within the encoding and if it is wrong a form of interrupt can be applied.

Data transfer control

In the decision of this requirement, we took a page from the given instruction sheet as it would server no logical purpose that we cannot control. The implementation of this would be from an interrupt that we introduce to the system when we wish to start sending data and stop sending.

Data transfer speed

The system protocol should be fast to respond but must still be perceptible by the average user. The team decide that making the system extremely fast was not viable when we planned to use LED's and LDR's that have limited speed instead of the using direct GPIO transfer as we plan to stay true to the purpose of the project.

Receiver requirement

In summary what is required for this system is a means to get light based data that will incapsulate an ADC value from the transmitter and be able to reproduce that value showing the working of the LoT protocol and that it has retained data throughout the data transmission process.

The implementation of this system makes use of an LRD that will change the voltages drop across it as the light intensity changes. This change in voltage will be captured by the GPIO pin of the STM. The value will be run through an ADC to give a value between 0 and 4095. Keeping in mind that it will be an array that of the 12-bit data with the start and end bits. Each bit of the 12-bit data will have to go through a comparator operator to convert the ADC values to one's and zero's (Binary equivalent) and that data will run through a binary to digital converter algorithm and if the system is a success the out form the receiver must match that of the ADC value of the transmitter.

decoding of data

The Idea of decoding the data came from the fact that we are not just receiving the data as it is but it comes with other information that encloses it and the decoding of this package must not cause an data loses; to keep from losing data the implementation will be that the receiver knows what to expect and in a rare cause that happens the data is not correct…

Error checking

This subsystem is implemented to make sure that data that is received is always correct and if not, that data will be discarded. This was another concept that was taken from the instruction sheet. The error tracking is implemented using the start and end bits. The was it works is that there is a counter that starts up after the start bit and it ends at the end bits and if the counter of values in the data section does not match the pre-set value it has lost data.

Data receive control

The same reasoning applies for transmission of data here and it is useful to be able to stop receiving data when we wish a use cause of this would simply be we what data for a specific time only and not for all of time.

Data receiving speed

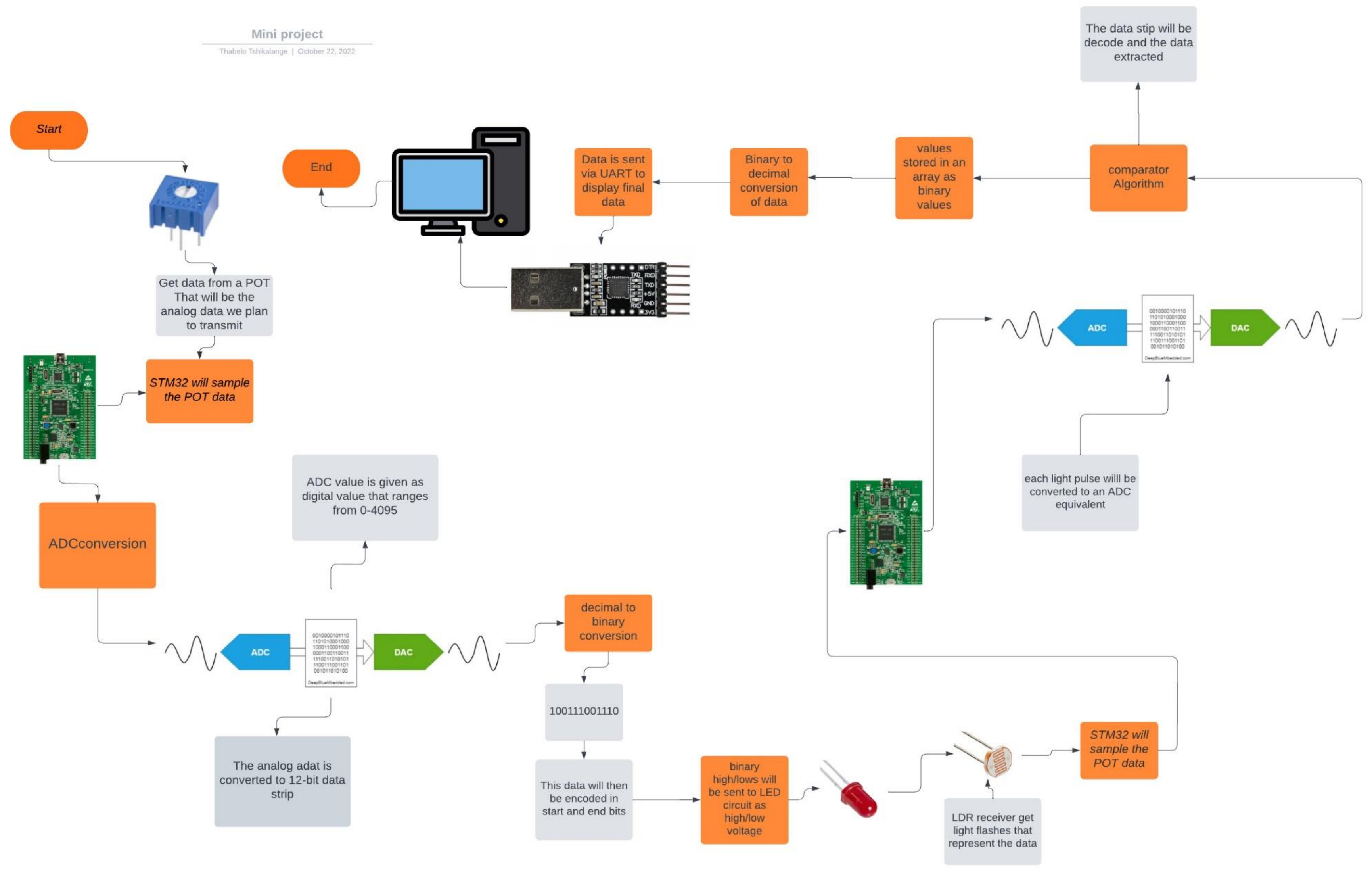
This subsystem for the receiver cause if the timing of that sampling rate is of a lot of data will be ruined and the hance, we require this to match up with the transmission of the data and this is by setting the same time stamp for all data transferred and received. And by making full use of the start and end bit.

One of the problems we face with speed is that the LDRs does not have infinite sampling time and the faster we make the light switching the LDR we end up reading constant output and results in data losses.

System requirement

The system in general must have seamless communication the Acknowledge massages and data massages must flow naturally without constant human input to the system.

Below is the full system diagram that summaries the above information:



Mini project
Thabelo Tshikalange | October 22, 2022
Start
Get data from a POT That will be the analog data we plan to transmit
STM32 will sample the POT data
ADCconversion
ADC value is given as digital value that ranges from 0-4095
ADC
DAC
The analog adat is converted to 12-bit data strip
decimal to binary conversion
100111001110
This data will then be encoded in start and end bits
binary high/lows will be sent to LED circuit as high/low voltage
LDR receiver get light flashes that represent the data
STM32 will sample the POT data
each light pulse willl be converted to an ADC equivalent
ADC
DAC
comparator Algorithm
The data stip will be decode and the data extracted
values stored in an array as binary values
Binary to decimal conversion of data
Data is sent via UART to display final data
End

Figure 3:System Diagram

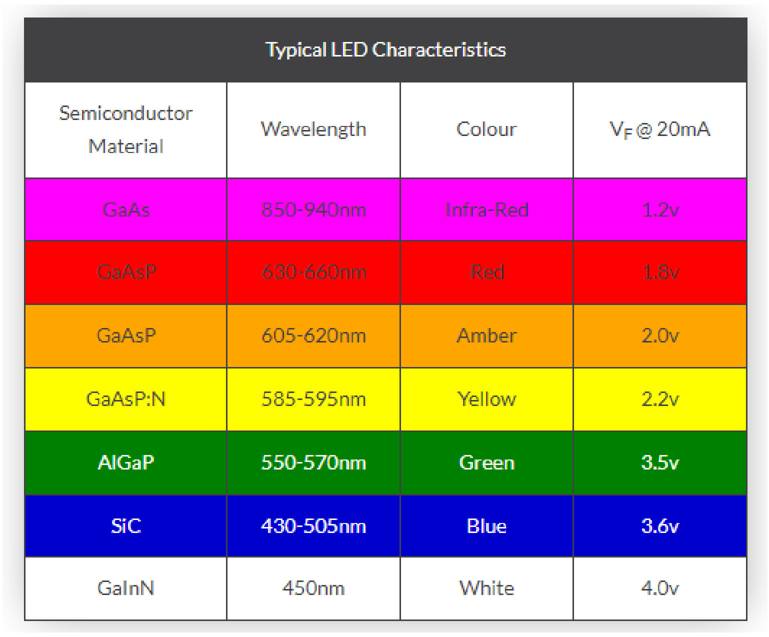
Specification and Design

Based on the requirements there were specifications that we needed to set up to design a working system.

Transmitter specification and Design concept

In designing the transmitter interface the objective was to send out given data to its binary equivalent that will be send out through an LED.

Some of the specs were to choose an LED that will offer a bright enough intensity that will be perceived by the receiver block. The LED was red. Reason for this decision was due to the properties it had as shown below.



Typical LED Characteristics			
Semiconductor Material	Wavelength	Colour	V_F @ 20mA
GaAs	850-940nm	Infra-Red	1.2v
GaAsP	630-660nm	Red	1.8v
GaAsP	605-620nm	Amber	2.0v
GaAsP:N	585-595nm	Yellow	2.2v
AlGaP	550-570nm	Green	3.5v
SiC	430-505nm	Blue	3.6v
GaInN	450nm	White	4.0v

Figure 4:LED specs [1]

Figure 4 shows the voltage required to power the LED's and their associated wavelengths. Form this we see that the voltage needed by the red led is 1.8v which is good for our system as we only have 3.3V that the GPIO pin can produce. Its wavelength is large enough and intense enough for the receiver to system discern. And again, the red LED is easily assessable.

With the hardware of the system determined the next line of action was to look at how the software that will bring the system to life will work.

The transmitter external circuitry

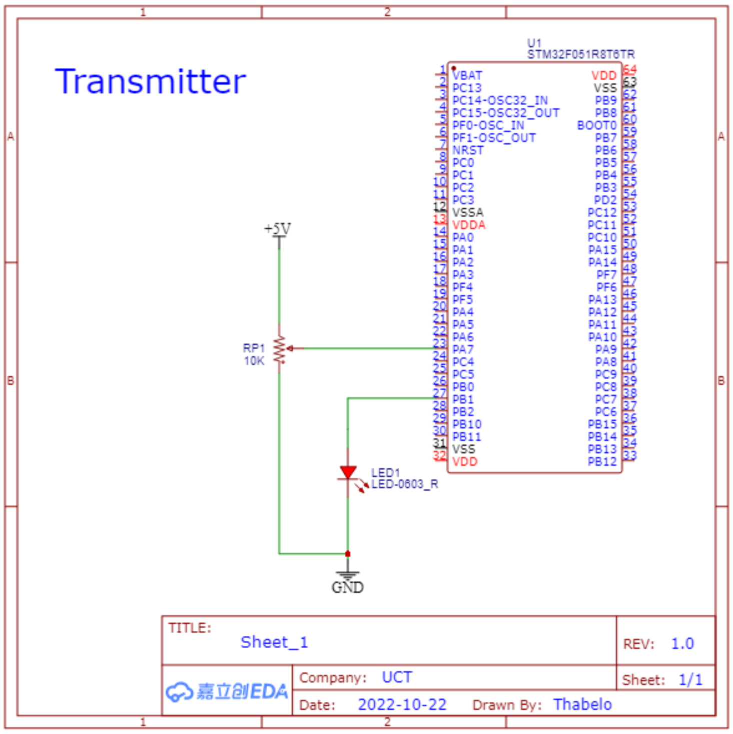


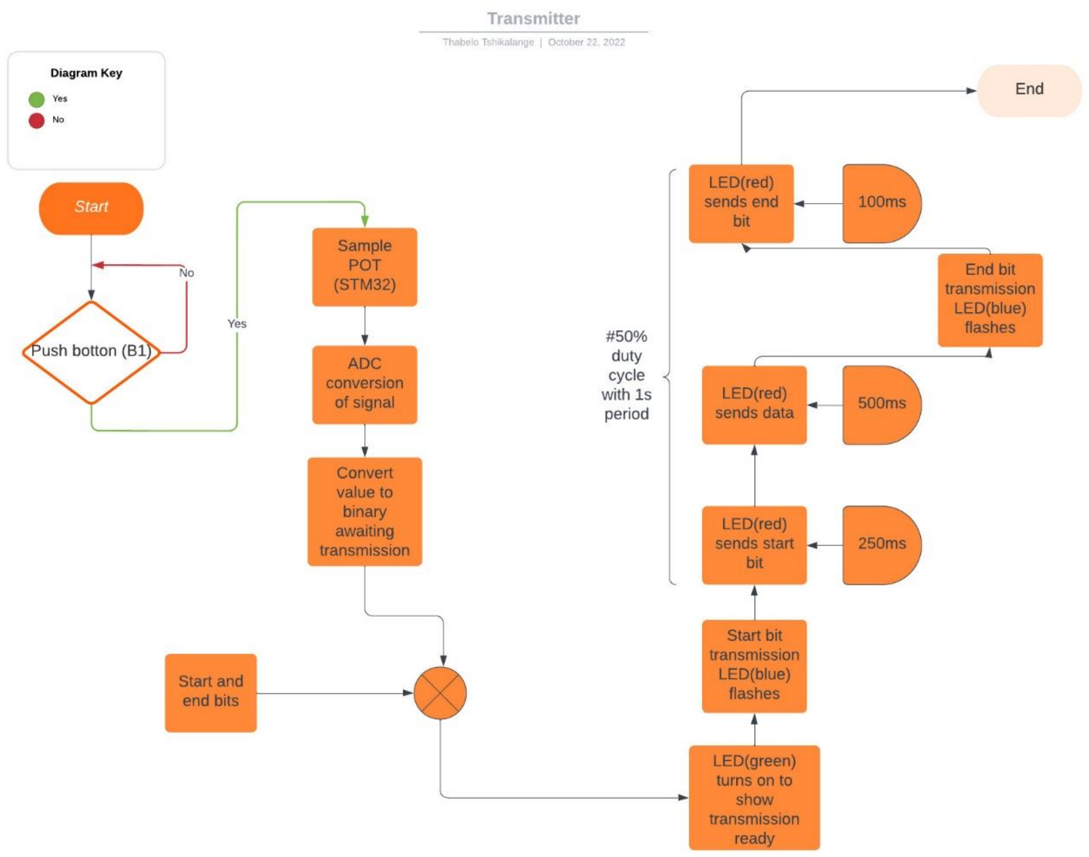
Figure 5: transmitter external circuitry

The software design is responsible for sending out a voltage high/low to the LED. The system must firstly know where it is in the transmission process whether it is sending the data, start or end the implementation of this system uses a delay that was chosen by the team. The coding of this process is such that when we are sending the start bit for example it has a short delay time so that it can be disfigured for all the other data values and the same is done for the stop and data bits. This implementation will now help the user see what is going on when data is begin transmitted using the LED. Then to give the system a way to confirm what has happen has indeed taken place an Acknowledge signal has to be given and this was the specifications we need when representing the data.

For the data transfer control requirement, the team specified that the use of a pushbutton will determine whether we will send data or not. The implementation of this use means of an interrupt when the pushbutton is engaged that will trigger to change state.

The speed of the system was determined by the overall data that comes though with the set delays. To maintain viewer readability and speed in the system the specified value for transmission was 8 seconds.

Transmitter flow chart



Transmitter
Thabelo Tshikalange | October 22, 2022
Diagram Key
Yes
No
Start
Push botton (B1)
No
Yes
Sample POT (STM32)
ADC conversion of signal
Convert value to binary awaiting transmission
Start and end bits
LED(green) turns on to show transmission ready
Start bit transmission LED(blue) flashes
LED(red) sends start bit
250ms
LED(red) sends data
500ms
End bit transmission LED(blue) flashes
LED(red) sends end bit
100ms
#50% duty cycle with 1s period
End

Figure 6:Transmitter flow chart

Receiver specification and Design concept

The components that were chosen for this set up were a LDR and resistor to form a voltage divider the spec that had to be insured had to be that the voltage difference between the on and off state of the LED had to be significantly different to use the comparator function. The use of a 1K-60KΩ LRD was best seated for our receiver.

The receiver external circuitry

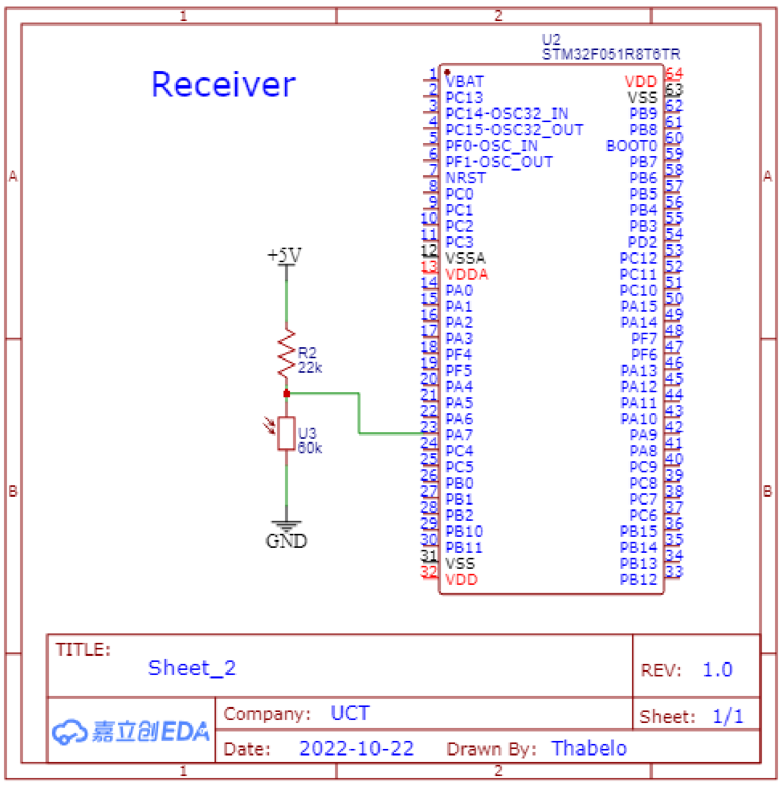
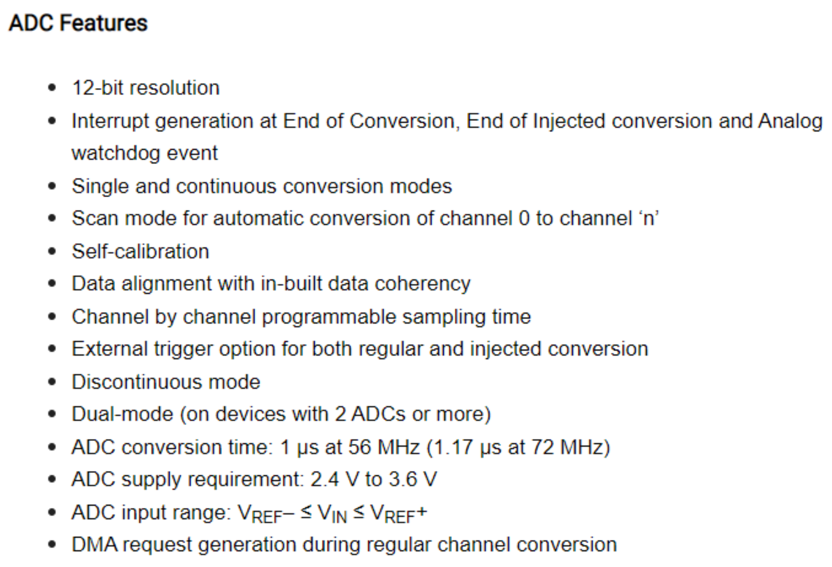


Figure 7: receiver external circuitry



ADC Features

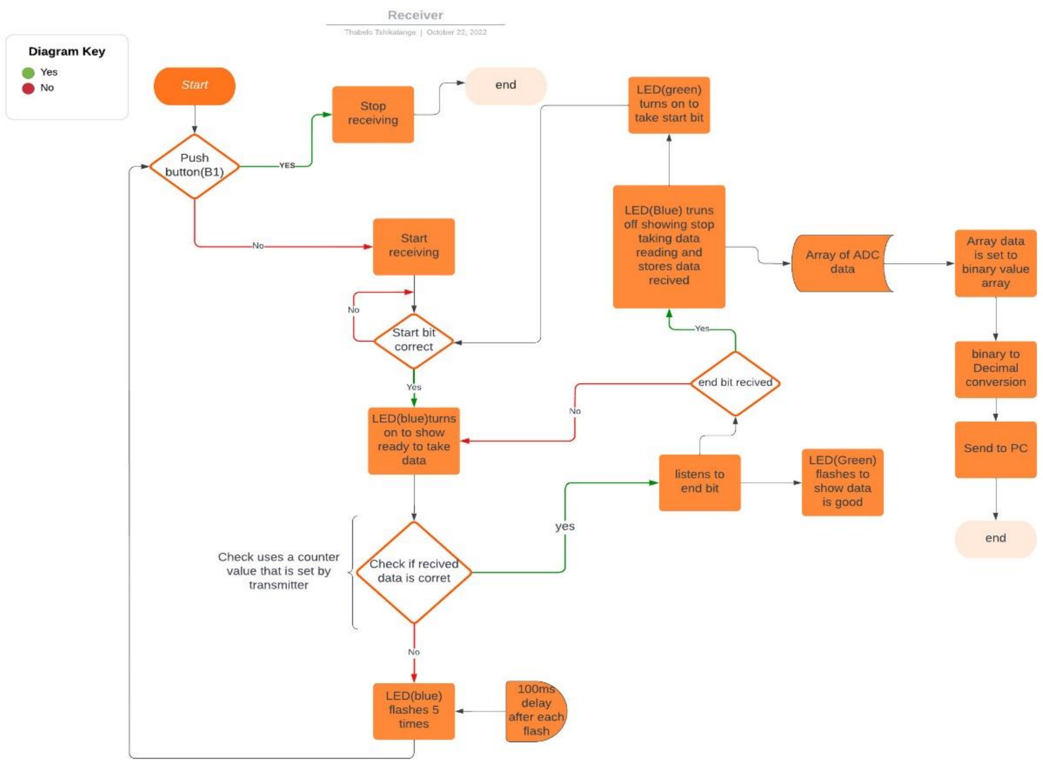
- 12-bit resolution
- Interrupt generation at End of Conversion, End of Injected conversion and Analog watchdog event
- Single and continuous conversion modes
- Scan mode for automatic conversion of channel 0 to channel ‘n’
- Self-calibration
- Data alignment with in-built data coherency
- Channel by channel programmable sampling time
- External trigger option for both regular and injected conversion
- Discontinuous mode
- Dual-mode (on devices with 2 ADCs or more)
- ADC conversion time: 1 µs at 56 MHz (1.17 µs at 72 MHz)
- ADC supply requirement: 2.4 V to 3.6 V
- ADC input range: $V_{REF-} \leq V_{IN} \leq V_{REF+}$
- DMA request generation during regular channel conversion

Figure 8: ADC spec [2]

The software implementation of the system only starts operating when it is accessed correctly, and it will remain active until it is signalled to go off by the LED. Using the delay of the start bit to our advantage for the receiver. The system will wait to receive this exact sequence of on off signal so in the coding we specified a delay of **200ms** and in that period two one states must be confirmed for the receiver to continue to the next sequence of data reading. From knowledge of our system, it is known that the ADC value is always a value that can be represented in a 12-bit binary reading and that each reading is **500ms** the Receiver can be configured to sample for **six seconds** after receiving the start bit. From there it waits for the transmission of the stop bit to reset to the receiving to wait for the start.

The error checking makes use of the implementation of the values at certain points of the data and if there’s a mismatch message of that is indicated. That data receives has a control system in place that is specified by the use of a pushbutton that will launch an interrupt.

Receiver flow chart



Receiver
Thabelo Tshikalange | October 22, 2022
Diagram Key
Yes
No
Start
Push button(B1)
YES
Stop receiving
end
No
Start receiving
No
Start bit correct
Yes
LED(blue)turns on to show ready to take data
Check uses a counter value that is set by transmitter
Check if recived data is corret
No
LED(blue) flashes 5 times
100ms delay after each flash
yes
listens to end bit
LED(Green) flashes to show data is good
end bit recived
No
Yes
LED(Blue) truns off showing stop taking data reading and stores data recived
LED(green) turns on to take start bit
Array of ADC data
Array data is set to binary value array
binary to Decimal conversion
Send to PC
end

Figure 9:Receiver flow chart

Implementation

Transmission implementation

Some of the code implementations will be explained below starting with the transmission block.

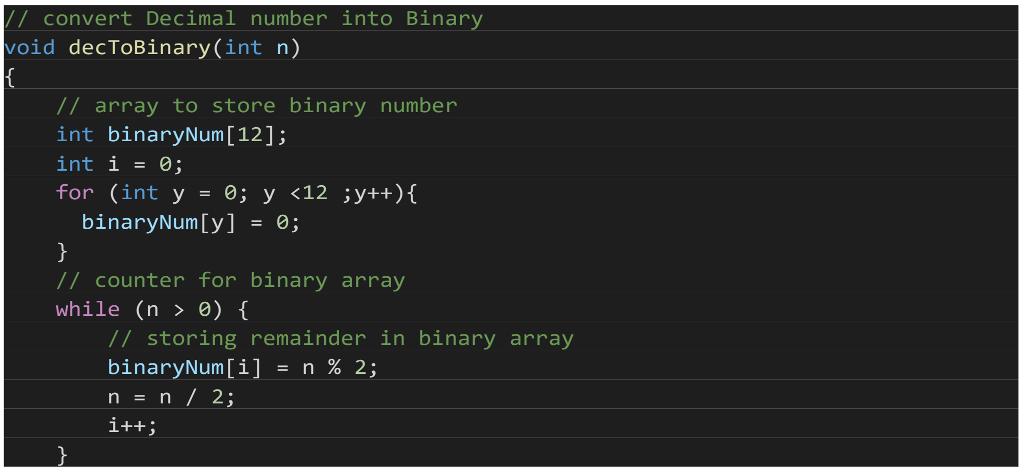

```
// convert Decimal number into Binary
void decToBinary(int n)
{
    // array to store binary number
    int binaryNum[12];
    int i = 0;
    for (int y = 0; y <12 ;y++){
      binaryNum[y] = 0;
    }
    // counter for binary array
    while (n > 0) {
        // storing remainder in binary array
        binaryNum[i] = n % 2;
        n = n / 2;
        i++;
    }
```

Figure 10: Decimal to binary conversion

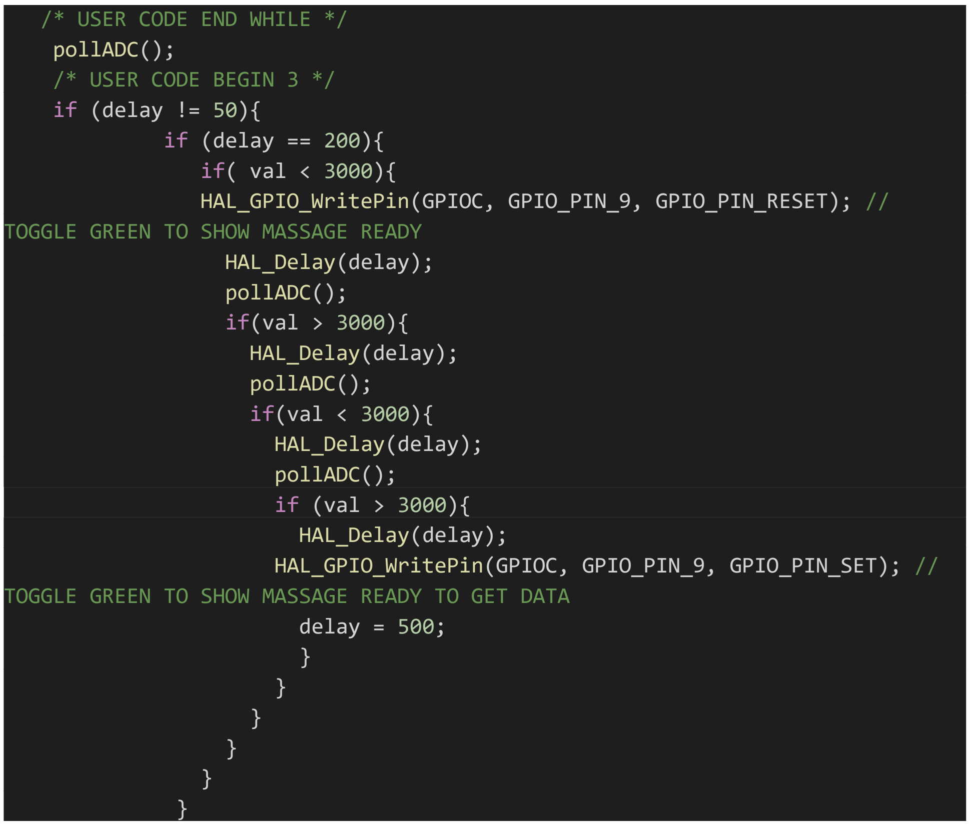
The figure above shows the implementation of the conversion from decimal to binary. From the main method this method is called to take the ADC value from the pollADC () method as a perimeter and run the conversion then it is stored in a binaryNum array that is a length of 12-bits and this is due to the nature of our ADC value begin a value that is 12-bits long. This is the binary array that is transmitted as light data in the continuation of the function below

```
    HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_8);// toggle pin 8 to show sending of
start bit
    for (int z = 0 ; z < 2; z++){
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_Delay(250);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET);
        HAL_Delay(250);
    }
    HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_8);// toggle pin 8 to show done sending
of start bit
    for (int j = i-1; j >= 0; j--){ // sending of binary values to light up
LED for 1 and 0 values
      if (binaryNum[j]==1){
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_Delay(500);
      }
      else{
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET);
        HAL_Delay(500);
      }
      counter++;
    }
    counter = 0;
    HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_8); // toggle Pin 8(blue led) to show
sending of end bit
    for (int z = 0 ; z < 5; z++){
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_Delay(100);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET);
        HAL_Delay(100);
    }
    HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_8);// toggle pin 8 to show end of end
bit
}
/* USER CODE END 4 */
```

Figure 11: Transmission block

Before the data is transmitted a set of start bits are sent out by pulling the port high and low twice with a delay of **250ms** for each flash and this will be the way the receiver is accessed ,from there we go over the array and for a **1** in the array we set the port high and for a **0** low this will the send voltages to the LED respectively. The data set has a 500ms delay for each reading after that the blue LED is toggled to show end of message. The end bit is sent out which is a set of **five** flashes at **100ms** delay completing the processes the LED flashes again to show that it is done sending the bit.

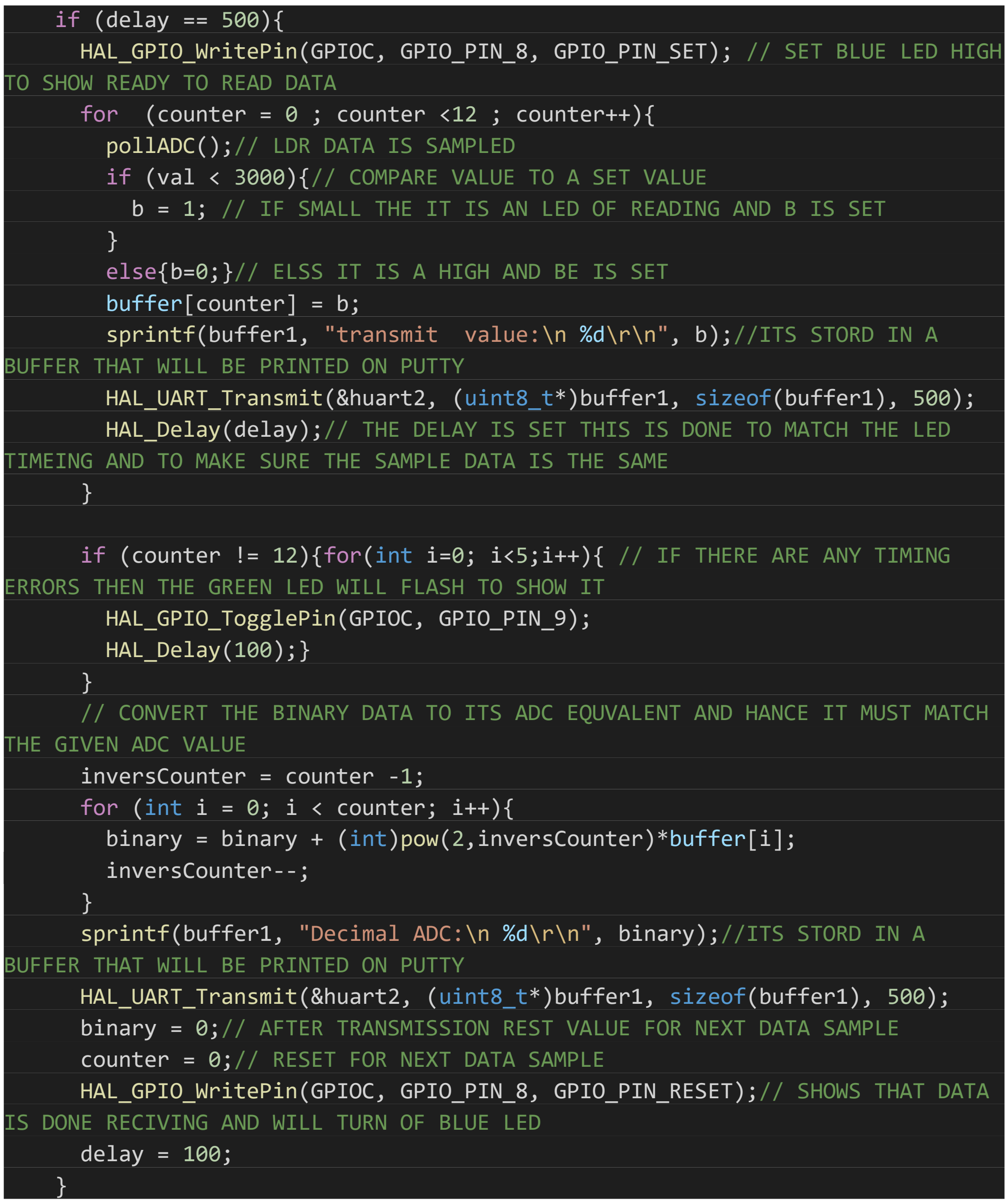
The figures below have the receiver code blocks.



```
   /* USER CODE END WHILE */
    pollADC();
    /* USER CODE BEGIN 3 */
    if (delay != 50){
             if (delay == 200){
                if( val < 3000){
                HAL_GPIO_WritePin(GPIOC, GPIO_PIN_9, GPIO_PIN_RESET); //
TOGGLE GREEN TO SHOW MASSAGE READY
                  HAL_Delay(delay);
                  pollADC();
                  if(val > 3000){
                    HAL_Delay(delay);
                    pollADC();
                    if(val < 3000){
                      HAL_Delay(delay);
                      pollADC();
                      if (val > 3000){
                        HAL_Delay(delay);
                      HAL_GPIO_WritePin(GPIOC, GPIO_PIN_9, GPIO_PIN_SET); //
TOGGLE GREEN TO SHOW MASSAGE READY TO GET DATA
                        delay = 500;
                        }
                      }
                    }
                  }
                }
              }
```

Figure 12: Receiver waiting for start bit

Figure 12 shows how the receiver waits for the start bit. Firstly it samples the ADC value it reads from the pollADC() function and compares it to its required value and if correct I will delay by a set delay value and samples the ADC value again and compares it again and this will continue till all cases are true then the green LED will flash back on to show that the correct start it was given and sets the new delay value for the next data set.

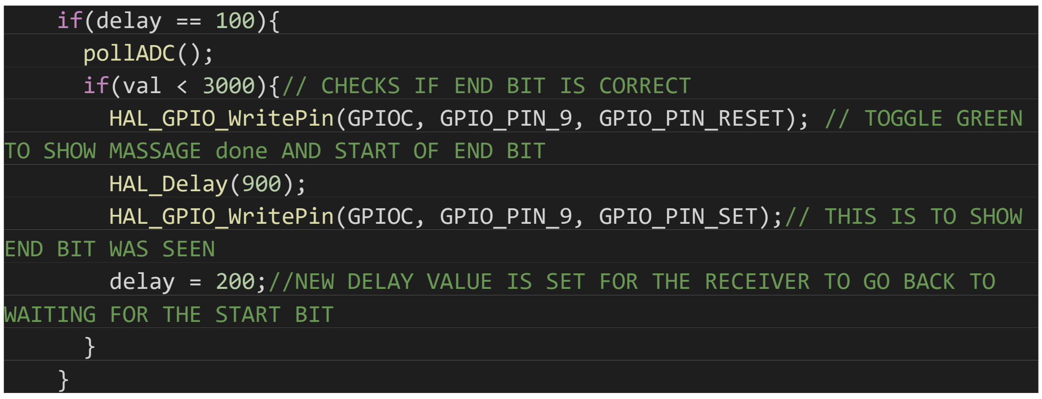


```
  if (delay == 500){
   HAL_GPIO_WritePin(GPIOC, GPIO_PIN_8, GPIO_PIN_SET); // SET BLUE LED HIGH
TO SHOW READY TO READ DATA
   for  (counter = 0 ; counter <12 ; counter++){
    pollADC();// LDR DATA IS SAMPLED
    if (val < 3000){// COMPARE VALUE TO A SET VALUE
     b = 1; // IF SMALL THE IT IS AN LED OF READING AND B IS SET
    }
    else{b=0;}// ELSS IT IS A HIGH AND BE IS SET
    buffer[counter] = b;
    sprintf(buffer1, "transmit  value:\n %d\r\n", b);//ITS STORD IN A
BUFFER THAT WILL BE PRINTED ON PUTTY
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer1, sizeof(buffer1), 500);
    HAL_Delay(delay);// THE DELAY IS SET THIS IS DONE TO MATCH THE LED
TIMEING AND TO MAKE SURE THE SAMPLE DATA IS THE SAME
   }

   if (counter != 12){for(int i=0; i<5;i++){ // IF THERE ARE ANY TIMING
ERRORS THEN THE GREEN LED WILL FLASH TO SHOW IT
    HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_9);
    HAL_Delay(100);}
   }
   // CONVERT THE BINARY DATA TO ITS ADC EQUVALENT AND HANCE IT MUST MATCH
THE GIVEN ADC VALUE
   inversCounter = counter -1;
   for (int i = 0; i < counter; i++){
    binary = binary + (int)pow(2,inversCounter)*buffer[i];
    inversCounter--;
   }
   sprintf(buffer1, "Decimal ADC:\n %d\r\n", binary);//ITS STORD IN A
BUFFER THAT WILL BE PRINTED ON PUTTY
   HAL_UART_Transmit(&huart2, (uint8_t*)buffer1, sizeof(buffer1), 500);
   binary = 0;// AFTER TRANSMISSION REST VALUE FOR NEXT DATA SAMPLE
   counter = 0;// RESET FOR NEXT DATA SAMPLE
   HAL_GPIO_WritePin(GPIOC, GPIO_PIN_8, GPIO_PIN_RESET);// SHOWS THAT DATA
IS DONE RECIVING AND WILL TURN OF BLUE LED
   delay = 100;
  }
```

Figure 13: Receive data

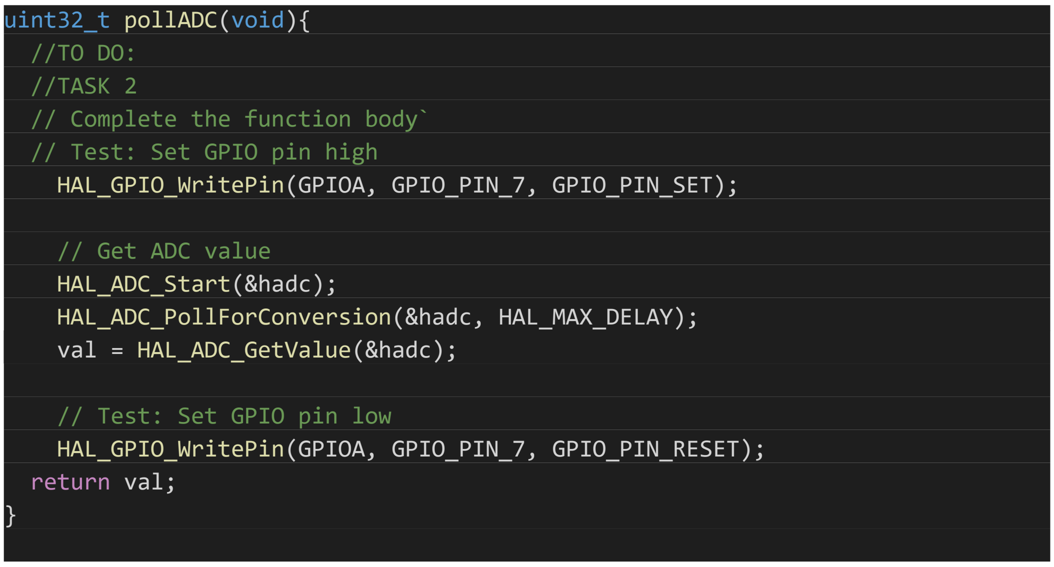
Figure 13 shows the receiving of data for the set delay for each ADC sample this value is then compared to set threshold that will determine if we are reading a LED high or low that will be shown on PUTTY. The toggling of the green LED will occur accordingly. Then a binary to decimal conversion data is ran and that value is shown using PUTTY. The new delay value is set accordingly.



```
    if(delay == 100){
      pollADC();
      if(val < 3000){// CHECKS IF END BIT IS CORRECT
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_9, GPIO_PIN_RESET); // TOGGLE GREEN
TO SHOW MASSAGE done AND START OF END BIT
        HAL_Delay(900);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_9, GPIO_PIN_SET);// THIS IS TO SHOW
END BIT WAS SEEN
        delay = 200;//NEW DELAY VALUE IS SET FOR THE RECEIVER TO GO BACK TO
WAITING FOR THE START BIT
      }
    }
```

Figure 14:END BIT

Above just checks if the end bit value is true and resets the delay value to get the receiver to wait for the start bit again and the green LED is toggled to show all this.



```
uint32_t pollADC(void){
  //TO DO:
  //TASK 2
  // Complete the function body`
  // Test: Set GPIO pin high
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_SET);

    // Get ADC value
    HAL_ADC_Start(&hadc);
    HAL_ADC_PollForConversion(&hadc, HAL_MAX_DELAY);
    val = HAL_ADC_GetValue(&hadc);

    // Test: Set GPIO pin low
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_RESET);
  return val;
}
```

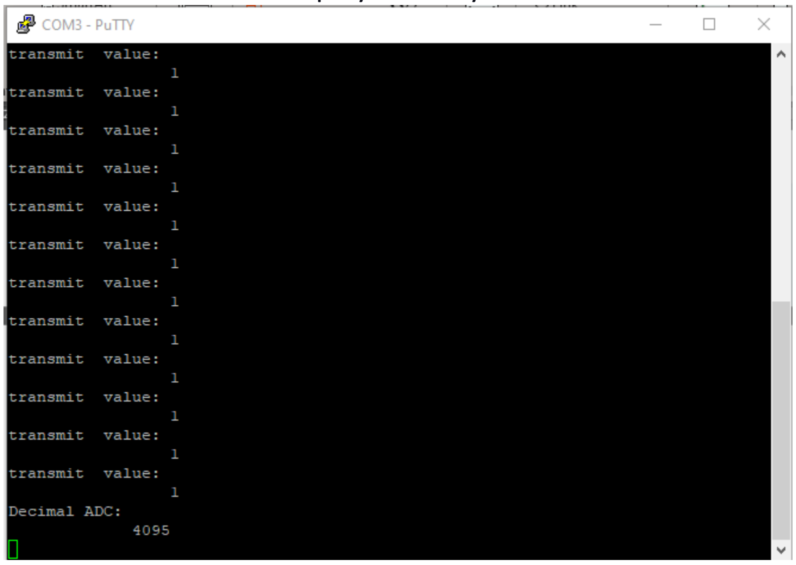
Figure 15: Sampling data

This is the famous pollADC () function that is called whenever data must be read from the port or the LDR of either system. The way it works is by setting the pin high to read data and it starts an ADC algorithm on that STM then gives a decimal equivalent.

Validation and Performance

To see the demo, use the provided link: https://youtu.be/_BfUzLLRCck
To see the code, use the provided link:
https://github.com/ThabeloR/TSHTHA094_MRBMAT004_EEE3096S/tree/main/Mini%20project

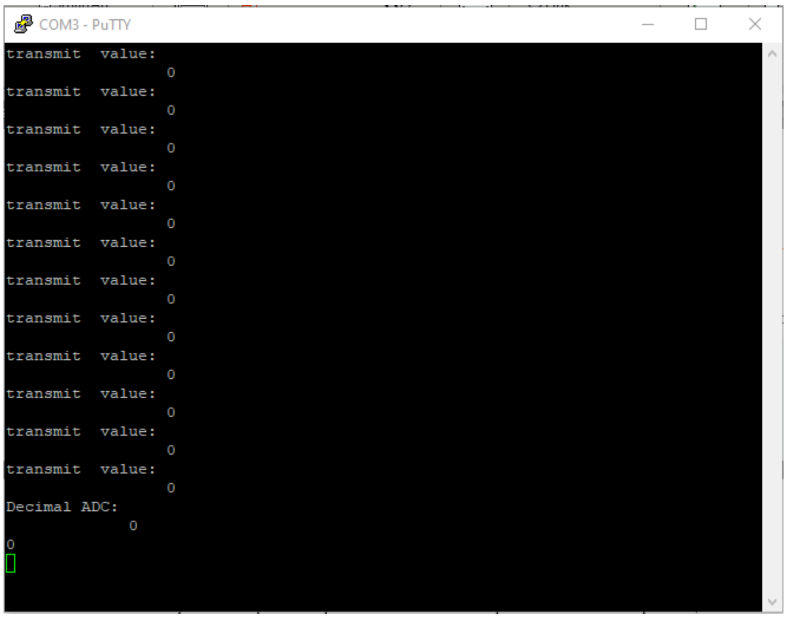
These are the PUTTY displays as they are unclear in the video



COM3 - PuTTY

```
transmit  value:
                 1
transmit  value:
                 1
transmit  value:
                 1
transmit  value:
                 1
transmit  value:
                 1
transmit  value:
                 1
transmit  value:
                 1
transmit  value:
                 1
transmit  value:
                 1
transmit  value:
                 1
transmit  value:
                 1
transmit  value:
                 1
Decimal ADC:
             4095
```

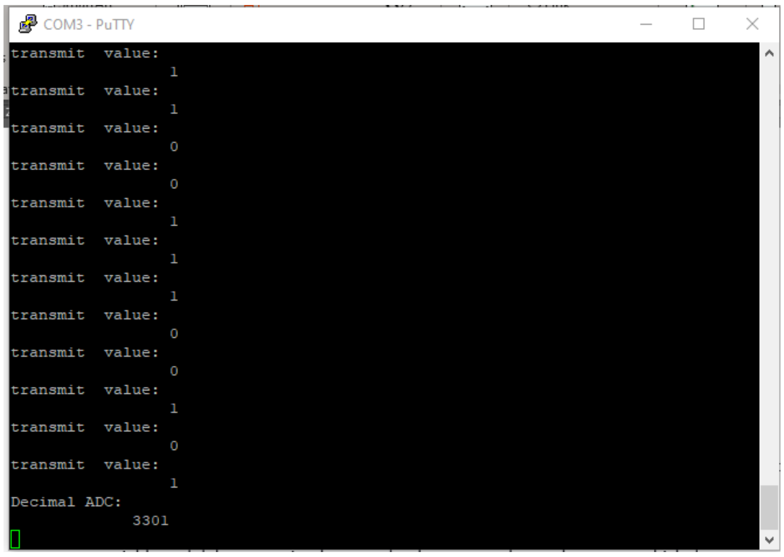
Figure 16: case 1 of system



COM3 - PuTTY

```
transmit  value:
                 0
transmit  value:
                 0
transmit  value:
                 0
transmit  value:
                 0
transmit  value:
                 0
transmit  value:
                 0
transmit  value:
                 0
transmit  value:
                 0
transmit  value:
                 0
transmit  value:
                 0
transmit  value:
                 0
transmit  value:
                 0
Decimal ADC:
             0
0
```

Figure 17: Case 2 of system



```
transmit  value:
                1
transmit  value:
                1
transmit  value:
                0
transmit  value:
                0
transmit  value:
                1
transmit  value:
                1
transmit  value:
                1
transmit  value:
                0
transmit  value:
                0
transmit  value:
                1
transmit  value:
                0
transmit  value:
                1
Decimal ADC:
             3301
```

Figure 18: case 3 of system

The system performed very well as the interface between the two main sub-systems, which is the transmitter and receiver worked perfectly. The transmitter sub-system worked successfully as it sampled the potentiometer on the pushbutton interrupt sent, converted the data into binary then sent the output data to the LED as visual output. The counter variable and delays were implemented to have control over the rate at which data was sampled and to keep track of the number of samples. The stop bit implemented was **100ms** toggling for **1s** to show the end of a data text.

The receiver sub-section worked mostly well; the pushbutton sub-section worked perfectly. The receiver could listen to the data signal from the transmitter. At first, the GPIO-linked wire was used, and the bit-bang GPIO was implemented but then the LDR method was preferred as it was easier and more direct to implement and was more in line with transmitting data using light which was the aim of this project. The LED from the transmitter then had to be made brighter for strength in signal and by not having a resistor before the LED.

However, when the receiver of the sub-system counter does not match the sample counter value of the transmitter sub-system. This case, which is, the checkpoint missing samples alert, worked successfully as the LED toggled for **100ms** for **1s** to show that there are samples lost in the transmission. The counter receiver == counter transmitter case is reached, and the LED fleshes and starts the process of waiting again.

The transmitter sub-system worked as expected as all the test cases produced the required results. The extremes:
In the zero-test case, it is seen that the LED does not transmit any data as it is off then after the data package is through, it flickers for **100ms** for one second, to indicate the end of text transmission. In the max-test case, the LED is at its brightest and toggles every **500ms** for the duration of data transmission and then flickers for **100ms** for one second to show the end of text transmission. The mid-test-case behaves like the max test case just less bright. The data can be seen from the putty.

The receiver sub-system works as expected having it only work when the pushbutton allows it and when the transmitter writes to it with **2, 200ms** delay signals (1010) which acts as an "address" to the receiver. Then after the next set of data is read and converted to its binary equivalent. This came out as a set of 12 on/off signals.

For the test case the data set was correct and was converted back to decimal perfectly showing that the data sent is received and not lost.

Testing the error cases by having the data come through incorrectly was perfectly detected by the receiver and the led flashes for at **100ms** for **1s**. The end bit causes the receiver to stop sampling as expected and this is indicated by the LED going off.

Conclusion

As seen and explained above, the transmitter worked perfectly, and the interface and integration of the sub-sections gelled together. The receiver sub-system was configured to listen to the incoming data signal from the receiver successfully and the push-button subsection collaborated perfectly with the receiver subsection that can be engaged by the sue of a pushbutton and wait for its write address. And the correct data is pushed through UART throughout.

This is not a useful product as its success depends on a controlled environment that is free from obstructions and requires supporting external circuitry that needs amplifiers to strengthen the signal, which is prone to noise with age and will require regular maintenance. A wired solution can be implemented using the UART/RS232, which is also limited by the distance allowable on the implementation to still can preserve data integrity and introduces latency and speed of data is limited by USB.

This system is good for light controlled environment (dark environments), within a short distance, with not much speed required unless using the I2C/SPI, and improved reliability can be achieved if modulation and filtering are implemented which are more involving.

References

[1] "Electronics Tutorials," AspenCore, 2022. [Online]. Available: https://www.electronics-tutorials.ws/diode/diode_8.html. [Accessed 22 10 2022].

[2] K. Magdy, "DeepBlue," DeepBlueMbedded.com, 2022. [Online]. Available: https://deepbluembedded.com/stm32-adc-tutorial-complete-guide-with-examples/. [Accessed 23 10 2022].