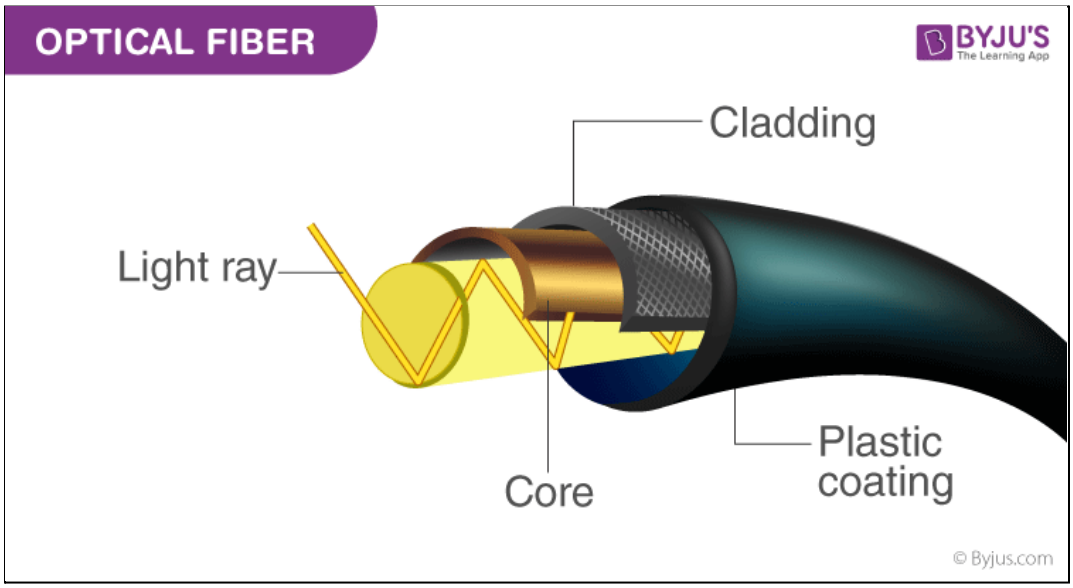
Topic of Interest

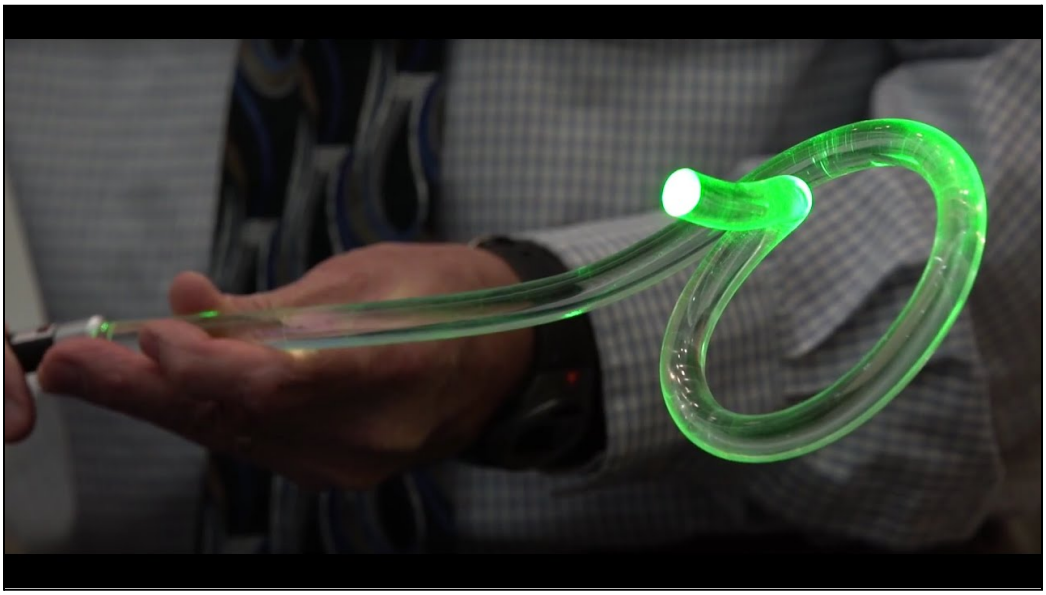
Fibre Optics

Fibre optics is a form of telecommunications that uses light to send data to different devices or nodes. Because light is being used to transfer data, the speed of data transfer is almost equal to the speed of light, 300 000 meters per second. To put this into perspective, a person travelling at the speed of light can circumnavigate the equator 7.5 times in one second. This makes fibre optics the fastest form of data transfer we have. One of the major advantages of fibre optics aside from its speed is its immunity to electromagnetic interference. The issue with copper wires is that their data transfer and functionality performance is determined by their environmental temperature and conditions. However, light is not a physical material, hence, physical conditions like the latter do not affect its performance. The significant impact of fibre optics is seen in Singapore's broadband network. The entire country is connected via fibre optic cabling and the average download speeds are around 1 Gb/s or 1000 Mb/s. Due to their speed, fibre optic cables are also used to connect Singapore and Australia via under-sea cabling as shown by the lime green line in the picture (image source).

Fibre optics rely on a phenomenon in Physics called total internal reflection. When light travels through a medium at an angle known as the critical angle, the light will bounce back into the medium instead of exiting it. Fibre optic cables rely on this phenomenon to

propagate light signals over long distances via a special wire known as a fibre optic wire shown below (image source 1, image source 2).



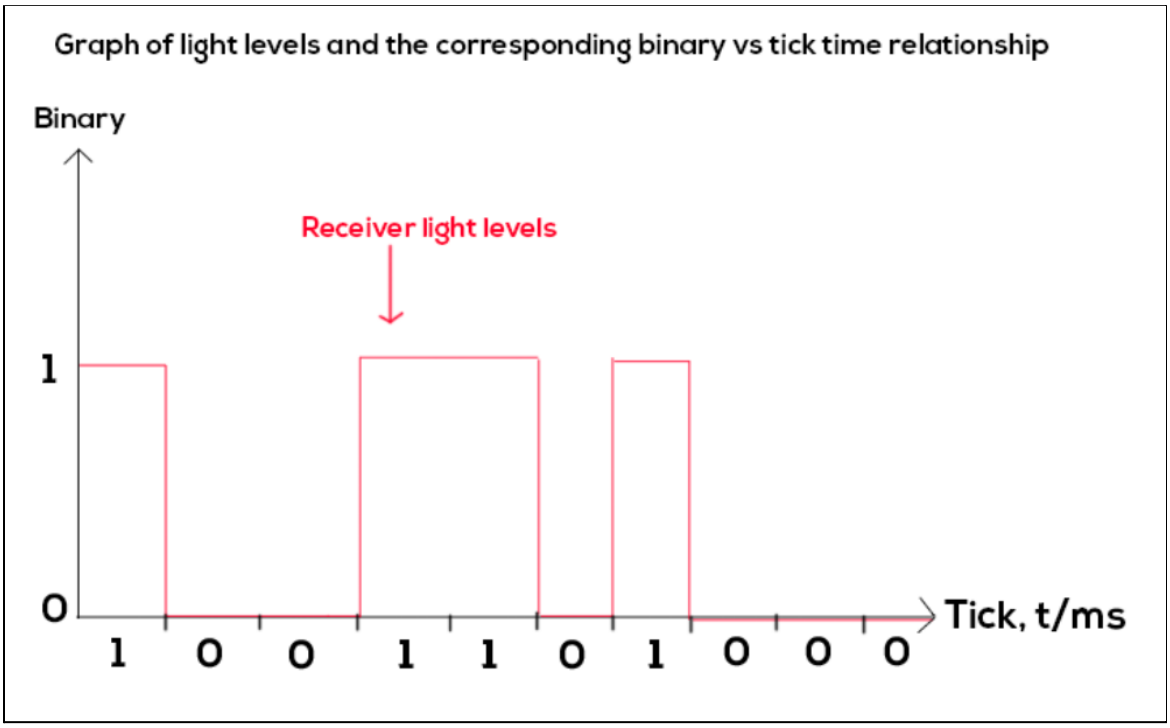


About The Project

I chose this topic for my negotiated study as I grew up in Singapore and have always wondered how light signals from a transmitter source can be converted into an electrical signal by a receiver and converted into meaningful readings. The following document contains my approach to the design of a fibre optic communications system. The final product will have two Arduino boards- one controlling the transmitter and the other, a receiver. The transmitter will send out timed intervals of light signals- an ON signal corresponds to a light signal reaching the receiver and an OFF signal corresponds to no light reaching the receiver (discussed later). The transmitter's job is to tell the receiver when to start reading the signal and when to stop reading, so a unique initializing signal will first be sent out and then the receiver will begin listening for the sequence of message signals. Once the message is finished, the transmitter will send out a unique terminating signal which will tell the receiver to stop reading and process the message.

Processing The Message

In computer science, it is common knowledge that a 1 corresponds to something true or something that is active and a 0 to something that is false or something that is inactive. For our purposes, the ON signal from the transmitter will be read by the receiver as a 1, and the OFF signal will be read as a 0. The resultant processed message will therefore be a binary sequence. The receiver will read (or listen to) the signals via an `analogRead` function by Arduino. There will be a light threshold, if the light detected is above this threshold, then the signal will be regarded as a 1, if it is below this threshold, then it will be regarded as a 0. The receiver's logic will read the message through a tick system where every set amount of time passed, it will listen to the state of the light level in the photodiode or photoresistor. If it is below the threshold, then it will log it as a 0. If it is above the threshold, it will log it as a 1. Below is a graph illustrating this.



Graph 1: Relationship showing the threshold light levels and the resultant binary vs tick time relationship. (graph by Farid Fadil)

Challenges

Syncing

One of the main things that will be a challenge is ensuring the syncing of the message signals so that the message sent out by the transmitter is read and matched exactly by the receiver. Because I have opted to use a tick system, this will undoubtedly be a pain to deal with because if the receiver misses a single signal tick by a bit, then it may misread an ON signal by the transmitter for an OFF signal.

Light Pollution

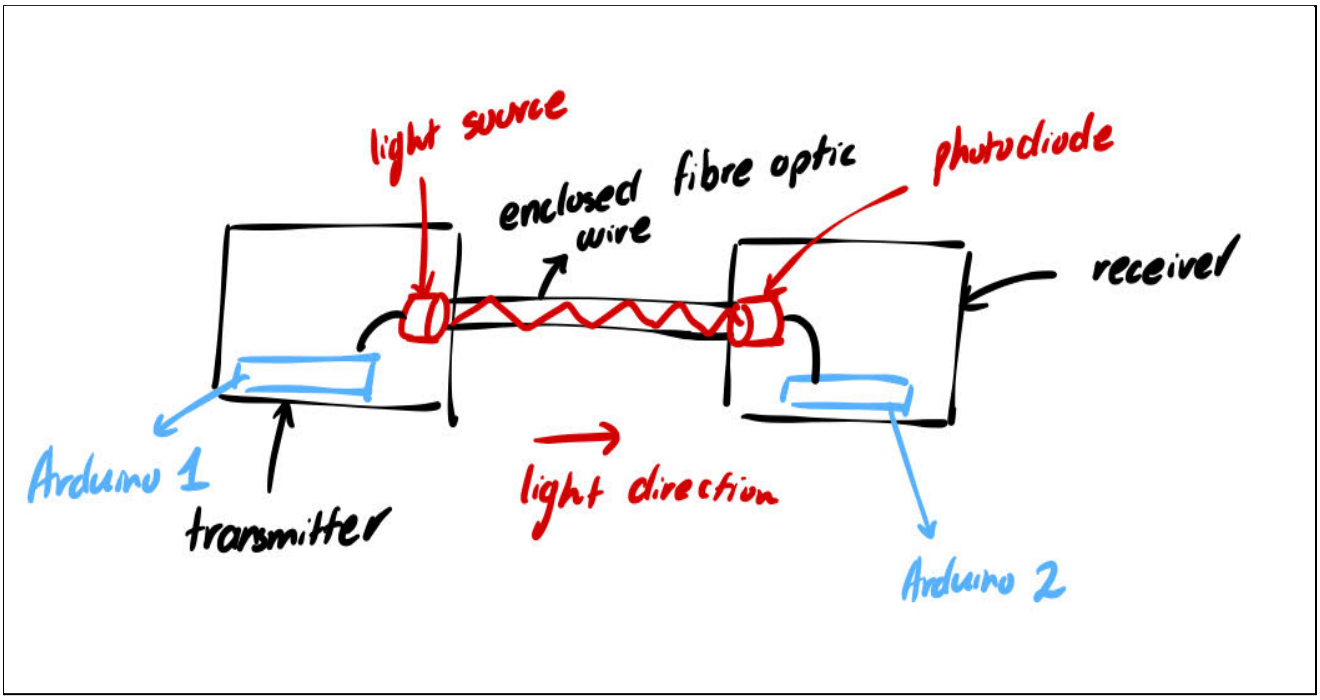
Another challenge is light leaking into the circuit. Ideally, no light should enter the circuit except the light source from the transmitter. The circuit must therefore be enclosed. See my hand-drawn diagrams below.

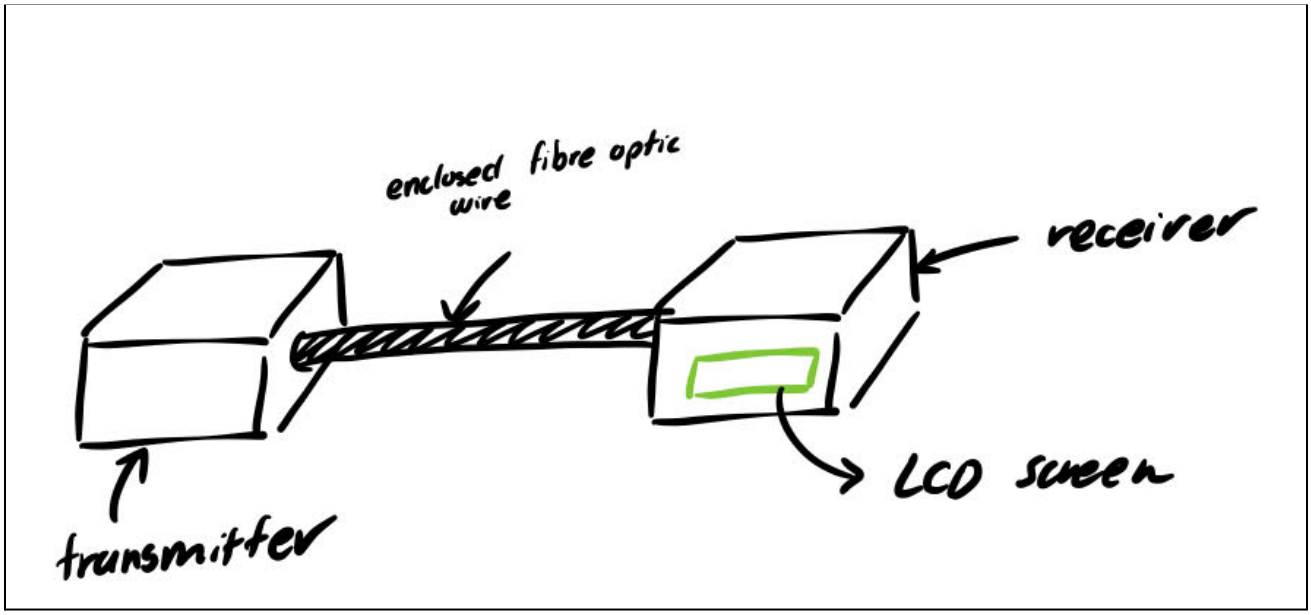
Binary to Message (and vice versa) conversion

Interfacing the code that talks to the Arduino with the more logical sided code related to the processing of the light signals into a “meaningful output” will be a challenge. Therefore, most of the heavy lifting will be in the pure C++ libraries that I will be writing as Arduino’s built-in functions do not have binary to character (or vice versa) conversion.

Pointers

I will be writing a lot of pure C++ code for an embedded system so pointers will be needed to ensure the most optimum manipulation and transferring of data for this project. I will aim to demonstrate and further my understanding of pointers by only using pointers in my source code.





Note the LCD screen is to display the decoded message. See '*Rough Receiver Code and Logic Outline'* below for an explanation.

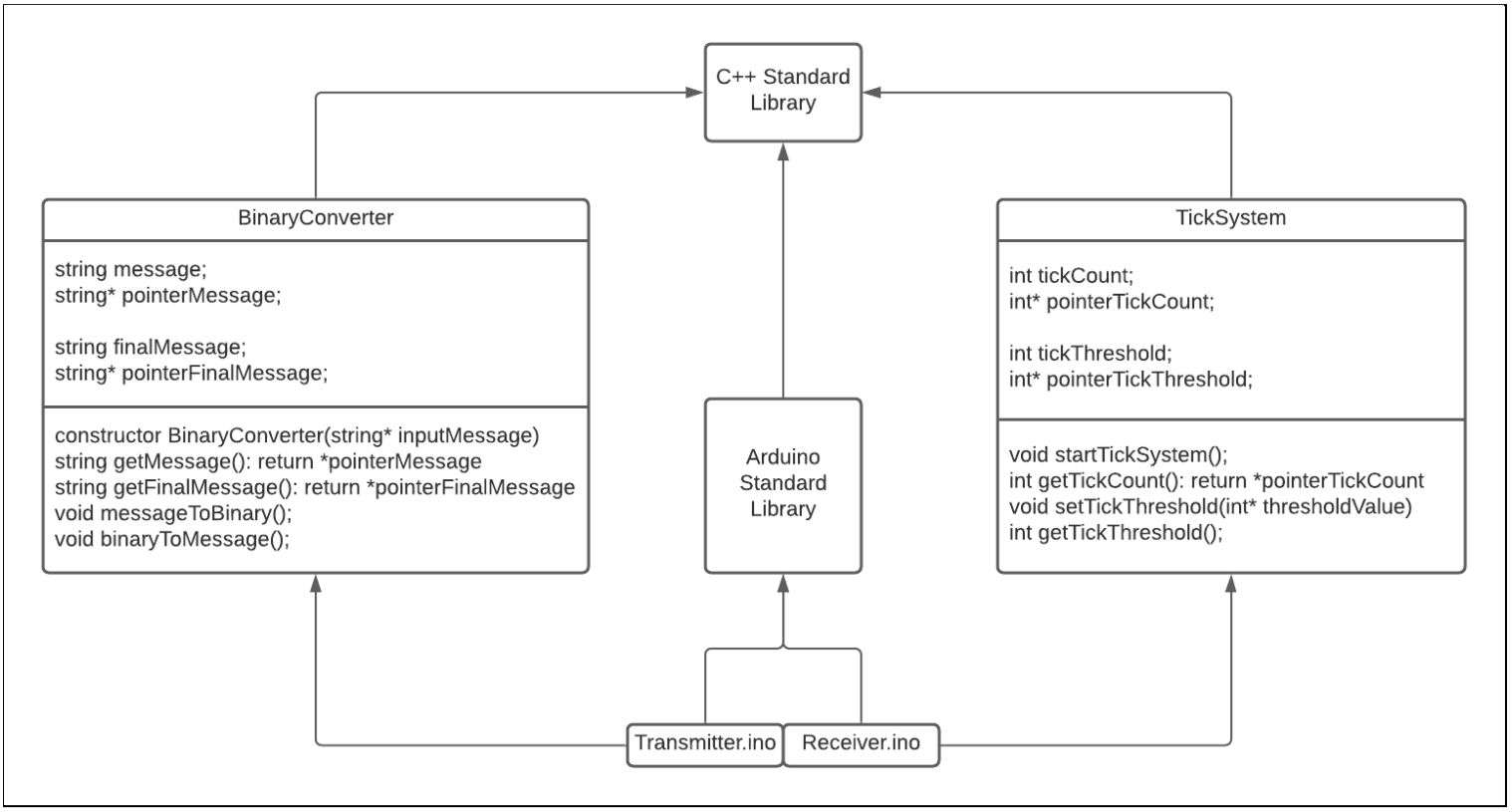
Rough Transmitter Code and Logic Outline

The code for the transmitter will have a string variable that stores a message in English. The string message will then be passed to a class that converts the individual characters into a binary sequence. The binary sequence will be taken and passed to a Transmitter class that will initiate the ticking system (by sending out the unique initializing signal to the receiver) where every tick, the corresponding light signal will be sent out. If it reads a 1 from the string, then the light source will turn on and if it reads a 0, then the light source will turn off. Ideally, when the initializing signal is sent out to the receiver, the code will aim to sync both the ticking systems of the transmitter and receiver.

Rough Receiver Code and Logic Outline

Once the receiver reads the initializing signal by the transmitter, it will initiate its tick system and listen for any incoming light signals. After every tick, it will listen to the state of the light level of the photodiode and append the status to a string. If the latter is above a set threshold value, then it will append a 1 to the string. If it is below the threshold value, it will append a 0 to the string. Once the receiver reads the terminating signal by the transmitter, the receiver will stop listening for any signals. The final string should have a full binary sequence appended to it. The string will be taken and passed to a class that decodes this binary sequence into an English message which will be displayed by the receiver’s LCD screen shown in the 3D illustration above.

UML Diagram



The BinaryConverter class, TickSystem class and the Arduino Standard library inherit from the C++ standard library, which acts as the “base” object in the UML diagram where everything comes from. All code in this project makes use of pointers as shown in the class attributes portion of the class diagrams.

BinaryConverter (BC)

This class contains the logic to convert a string message into a binary sequence and a binary sequence into a string message via the messageToBinary() and binaryToMessage() functions respectively.

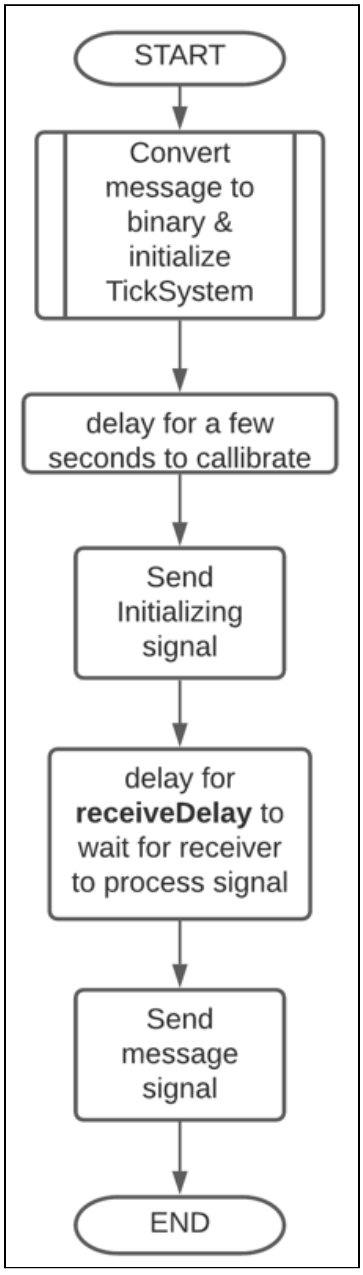
TickSystem (TS)

The TickSystem class contains functions to begin the tick system, get the current tick count, set the tick threshold (threshold after which the receiver will listen to the state of the photodiode or the threshold after which the transmitter will output the next signal) and to get the tick threshold.

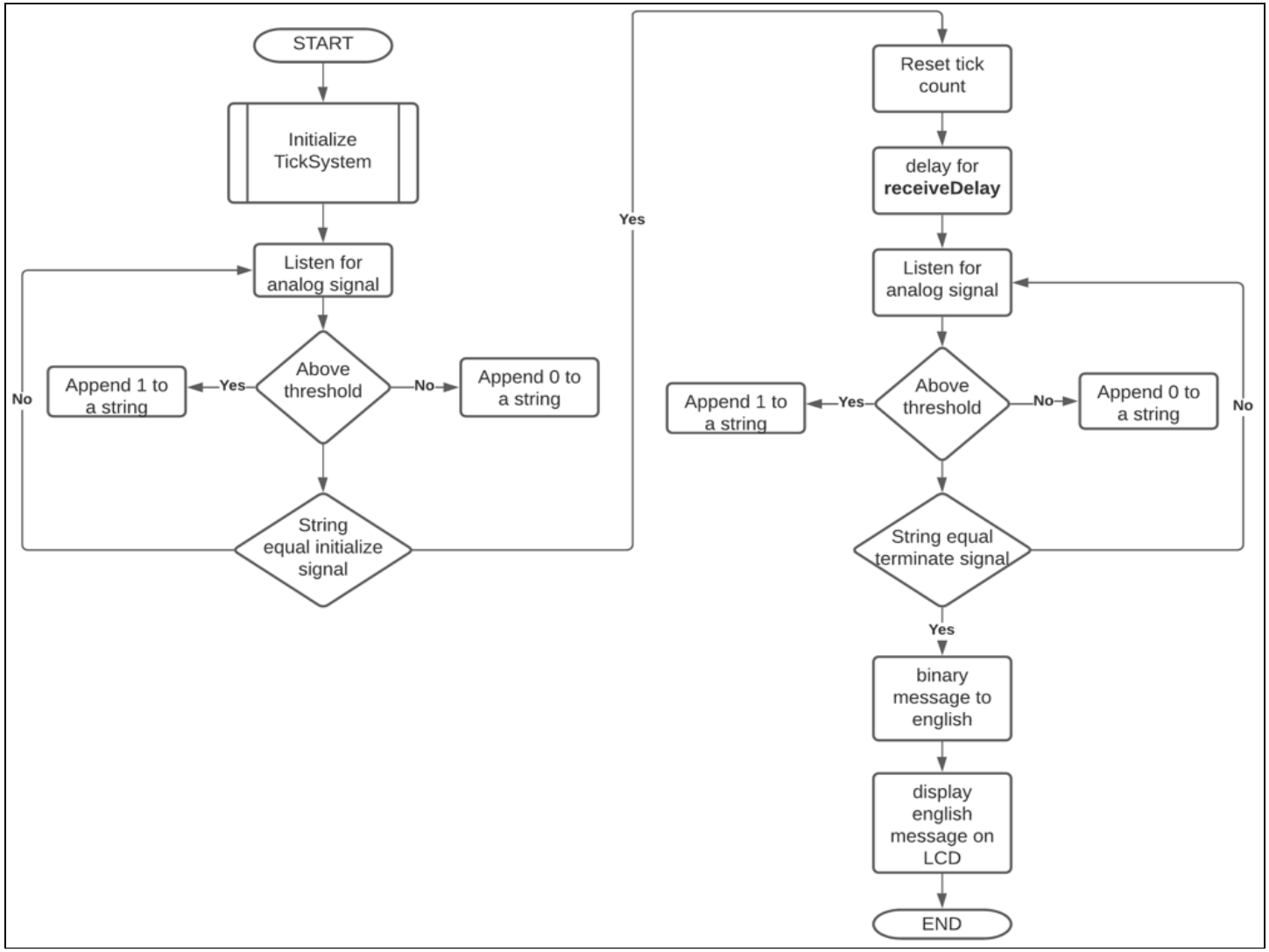
INO files

The INO files at the bottom inherit from the Arduino Standard Library, BC and TS.

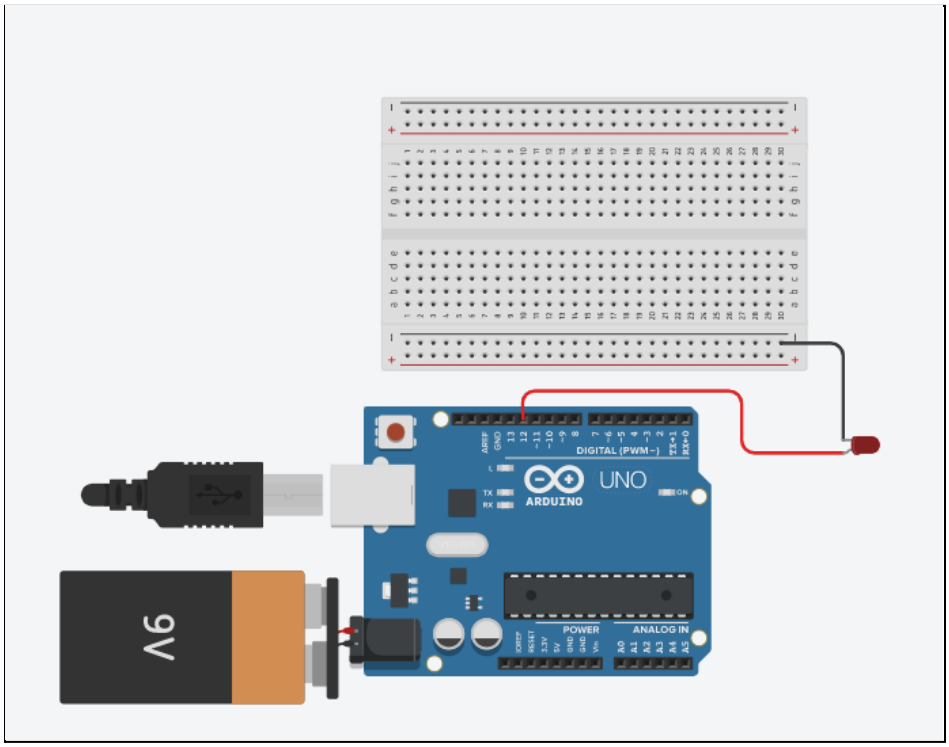
Transmitter Flowchart



Receiver Flowchart



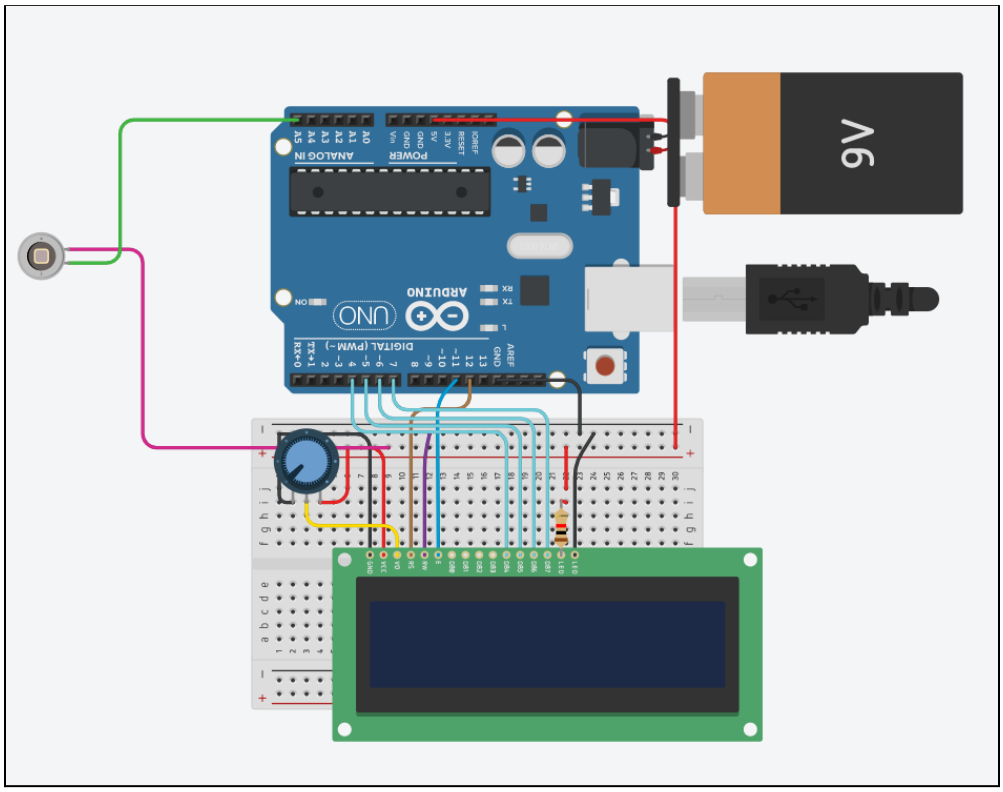
Project TinkerCad Diagrams



Transmitter Circuit

Despite the simple circuit, its underlying code is extremely complex and delicate which includes the TickSystem class and syncing of the light blinking with the receiver's TickSystem to ensure that no light "packets" so to speak, are lost. Taking inspiration from interdisciplinary concepts like clocking and synchronous tasks in my Networking and Security class, I decided to design the tick system around the latter. I didn't want to add sound or a random motor rotating to the circuit because frankly, what's the point? It simply wasn't practical. So I decided to build the circuit with the components I needed to make the transmitter a minimum viable product. What you don't see here is the connection of the fibre optic cable to the light source, represented by the LED. The real circuit utilizes a monochromatic (single wavelength) light source shown on the right. In order for total internal reflection to work, an LED would not suffice attributed to its relatively low light intensity juxtaposed to a concentrated beam of light.


Industrial Fiber Optics IF-E97 Fiber Optic Transmitter 660nm 1.7V 40mA Connector-less



Receiver Circuit

The receiver circuit contains a photodiode, an LCD, a potentiometer, a 200 Ohm resistor and a 9 Volt battery connected as shown above. The fibre optic cable mentioned above is connected to the photodiode. The idea is that the light source will send out a beam of light every time the Transmitter code reads a 1, and would not send out a beam of light if it reads a 0. Then, in order for the receiver to know when to read the status of the photodiode, its tick system will be used. By utilizing Arduino's loop() function, the receiver's tick system will run and once a particular value for its tickCount is reached, it will read the value from the photodiode. If the value is above a set threshold value, then it will append it to a string as a 1, otherwise, it will append a 0. Once the receiver reads a unique terminating signal from the transmitter then the receiver's code will pass the final string with the binary sequence into its local `BinaryConverter::binaryToMessage()` function and convert the binary sequence into an English text. The English text will then be returned and displayed on the LCD screen. Therefore, in order for the circuit to complete its purpose, the LCD screen **must** display the correct message.

Test Logs

Testing the TickSystem class

Testing to see how startTickSystem() would work in in a loop. This is pure C++ and is simulating the environment of Arduino's loop() callback function. Note the use of pointers.

```
#include <iostream>
#include "TickSystem.h"

TickSystem myTickSystem;
int tickThreshold = 3;
int* p_tickThreshold = &tickThreshold;

int loopCount = 10;

//main entry point of application
int main()
{
	myTickSystem.setTickThreshold(p_tickThreshold);

	for (int i = 0; i < loopCount; i++)
	{
		myTickSystem.startTickSystem();
		std::cout << myTickSystem.getTickCount() << std::endl;
	}

}
```

```
OUTPUT:
1
2
3
4
5
6
7
8
9
10
```

Testing string to binary conversion

Testing out the code that converts a string to an 8-bit binary number. C++ strings are basically char arrays so you can access the individual characters by looping through the string with a for a loop. I then use the bitset function in C++ to convert the individual characters into an 8-bit binary representation.

```
#include <iostream>
#include <bitset>
```

```
std::string message = "test";

//main entry point of application
int main()
{
	for(std::size_t i = 0; i < message.size(); i++)
	{
		std::cout << std::bitset<8>(message[i]) << " -> " << message[i]
<< std::endl;

	}
}
```

```
OUTPUT:
01110100 -> t
01100101 -> e
01110011 -> s
01110100 -> t
```

Testing char to binary sequence algorithm

After trying to carry over the above code from the main function into the BinaryConverter class, there were errors regarding the bitset function's parameters. I then resorted to implementing a pure C++ algorithm that converts an individual char into a binary sequence of numbers. The test below shows the algorithm's code and the tests to validate its functionality. (Courtesy of OrganicChemistryTutor).

```
//ALGORITHM CODE from BinaryConverter.cpp
BinaryConverter::BinaryConverter(std::string inputMessage)
{
    BinaryConverter::message = inputMessage;
}
void BinaryConverter::messageToBinary()
{
    int n = BinaryConverter::getMessage().length();
    std::string bin = "";
    for (int i = 0; i <= n; i++)
    {
        int val = int(BinaryConverter::getMessage()[i]);
        while (val > 0)
        {
            (val % 2) ? bin.push_back('1') : bin.push_back('0');
            val /= 2;
        }
        reverse(bin.begin(), bin.end());
    }
```

```
    BinaryConverter::finalMessage = bin;
}
```

```
#include <iostream>
#include "BinaryConverter.h"

std::string message = "momo";
BinaryConverter myBinary(message);

//main entry point of application
int main()
{
	std::cout << myBinary.getMessage() << std::endl;
	myBinary.messageToBinary();
	std::cout << myBinary.getFinalMessage() << std::endl;

	//std::string finalMessage = myBinary.getFinalMessage();
}
```

```
OUTPUT:
momo
11011011101101111101111111011
```